EXAMPLE-BASED RIGGING AND REAL-TIME ANIMATION OF CHARACTERS WITH LINEAR SKINNING MODELS

A Dissertation

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

By

Binh Huy Le

May 2014

EXAMPLE-BASED RIGGING AND REAL-TIME ANIMATION OF CHARACTERS WITH LINEAR SKINNING MODELS

Binh Huy Le

APPROVED:

Zhigang Deng, Chairman
Department of Computer Science

Guoning Chen
Department of Computer Science

Stephen Huang
Department of Computer Science

Tao Ju
Department of Computer Science and Engineering
Washington University in St. Louis

Weidong (Larry) Shi
Department of Computer Science

Dean, College of Natural Sciences and Mathematics

Acknowledgements

I would like to express my deepest thankful to my advisor, Dr. Zhigang Deng. Thank you for your trust, patience, motivation, and mentoring in the past six years. You have been teaching me all about scientific research, and most important, you let me be free to work on my own ideas. Thank you for your great support on building up my future career. Your persistent encouragement and valuable advices have always been making me better than before. I am proud that I was your student.

I would like to thank my committee members: Dr. Guoning Chen, Dr. Tao Ju, Dr. Larry Shi, and Dr. Stephen Huang for your insightful comments and constructive advices.

In my daily work, I have been blessed with friends in the Computer Graphics and Interactive Media lab: Chang Yun, Eric Gu, Qing Li, Mario Rincon, Xifeng Gao, Li Wei, Chrysanthi Chaleva Ntina (Sassa), Mingyang Zhu, Xiao Cheng, and Cheng Chen. Thank Xiaohan Ma and Nikhil Navkar for sharing with me memorable moments in my first three years. Your optimistic and your humor helped me survive thru the most difficult time in my PhD student life.

My special thanks go to my wife, Yen Le, for always being with me in this fabulous journey. Thank you for always keeping me joyful no matter if life is up or down. Thank you for taking care of our little family, and especially for your patience and gentle care for our lovely daughter. It is so difficult to be both a good mom and a successful PhD student and I adore that you made both.

Thank my little daughter, Phuong Le, for unintentionally motivating me moving forward. It is amazing to believe that my most productive working time came right

after you were born. I will never forget your emotional wow and your complementarity when you saw running animals in my demonstration videos of this dissertation.

Finally, I would like to thank my parents and my parents-in-law for supporting our family throughout this process. Without your great help and sacrifice, I and my wife might not be able to accomplish our study.

My PhD study was supported by Vietnam Education Foundation fellowship. During the time of this dissertation, I also received financial aids and scholarships from the University of Houston.

EXAMPLE-BASED RIGGING AND REAL-TIME ANIMATION OF CHARACTERS WITH LINEAR SKINNING MODELS

An Abstract of a Dissertation

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

By

Binh Huy Le

May 2014

Abstract

The creation of character animation of humans, animals, or plants plays one of the most important roles in computer graphics, with tremendous applications on creating motion pictures, video games, and virtual worlds. To this day, one of the most popular character animation techniques is skinning, where a set of low-dimensional control parameters is mapped to the high-dimensional geometric shape of the character model. Once the skinning model is setup, a manipulation on control parameters is propagated to the surface of the model accordingly, generating articulations and deformations of the character model. Skinning techniques allow effective controls of animation from different sources, such as artist input or motion capture data. In addition, skinning techniques can be used to accelerate the animation rendering or compress the motion data. This dissertation explores fundamental problems in the skinning pipeline including the model set up and the hardware accelerated rendering of animation. The scope of this dissertation only focuses on linear skinning models with using bone transformations as the controller.

To setup the skinning model, I propose example-based rigging techniques to extract the linear blend skinning (LBS) model from mesh sequence of the character model in different poses. The output LBS model consists of sparse, convex, and smooth skinning weights and rigid (orthogonal) bone transformations with or without a skeletal structure. The key contributions of my rigging technique are: (1) a fast iterative linear solver to optimize the orthogonal bone transformations, (2) a problem formulation and an optimization method to accurately constraint bone rotations around skeletal joints, (3) a rigidness Laplacian regularizer to constraint the smoothness of skinning weights, and (4) a robust method to extract the skeletal

structure from an over-completed initialization.

To accelerate the rendering of the LBS model during animation, I introduce a two-layer linear blend skinning model which can substantially reduce the computational cost of a dense-weight LBS model with insignificant loss of its visual quality. This two-layer model allows fast skinning animation without requiring the sparseness constraint on the skinning weights (dense-weight), which offers more flexibility on applications of the LBS model. The two-layer model is constructed by the sparse coding technique with directly using dense skinning weights or with using additional example poses to further improve its accuracy.

Contents

1 Introduction **1**

1.1 Contributions and Organization 3

1.2 Publications 5

2 Background on Character Skinning **8**

2.1 Geometric Skinning 8

2.1.1 Skeleton-based Skinning 8

2.1.2 Cage-based Skinning 14

2.1.3 Skinning from Single Shape 15

2.2 Anatomical Simulation Skinning 15

2.3 Example-based Skinning 16

2.3.1 Input Data 16

2.3.2 Skinning Weights Fitting 16

2.3.3 Skeleton Extraction 19

2.3.4 Basis Models 19

3 Example-based Rigging **21**

3.1 Notations 24

3.2 Problem Formulation 26

3.3 Algorithm Overview 28

3.4 Initialization 29
3.5 Joint Constraints and Joint Positions Solver 33
3.5.1 Joint Constraints 33
3.5.2 Joint Positions Solver 33
3.6 Skeleton Reconstruction 35
3.7 Skinning Weights Update 37
3.7.1 Non-negative Least Squares with Affinity Constraint 37
3.7.2 Rigidness Laplacian Regularizer 44
3.7.3 Iterative Local Solver 45
3.8 Bone Transformations Update 48
3.8.1 Absolute Orientation Problem 48
3.8.2 Rigid Transformations with Blending 49
3.8.3 Adding Joint Constraints 55
3.9 Skeleton Pruning 58
3.10 Results and Comparisons 60
3.10.1 With Skeleton 60
3.10.2 Without Skeleton 64
3.11 Discussion 68

4 Two-layer Blend Skinning Compression **80**
4.1 Problem Formulation 83
4.2 Sparse Compression Algorithm 87
4.2.1 Dictionary Update 89
4.2.2 Coefficients Update 90
4.3 Factorization from Example Poses 93
4.4 Results and Comparisons 95
4.5 Discussion 100

5 Conclusion **113**

Bibliography **116**

List of Figures

2.1 (a) Skeleton-based animation (images courtesy of [14] ©EUROGRAPHICS 2008). (b) Skeleton-based animation with bones represented by their transformations (images courtesy of [46] ©ACM 2013) 9

2.2 (a) Free-form deformation (images courtesy of [67] ©ACM 1986). (b) Cage-based deformation (images courtesy of [33] ©ACM 2005) 14

3.1 Only using a single set of parameters, my example-based method can accurately rig various models such as quadrupled animals (a), humans (b, c), and highly deformable models (d). My method can even generate bone structures for challenging parts, such as the mouth and the two ears of the cat (a), the skirt (b), and the elastic cow model (d). My method is robust; using only 9 frames, it can generate a skeleton with 28 bones for the cat model (a); note that even though the given example poses of the cat model have asymmetric poses, it still can generate an almost symmetric skeleton without imposing any symmetry constraints. 21

3.2 Examples to show two common limitations of current rigging 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 [25]. 23

3.3 The pipeline overview of our skeletal rigging approach. 28

3.4 An illustration of our cluster splitting strategy. The seed vertex is chosen as the off-center vertex with a large reconstruction error so that the resulting new clusters can maximally reduce the overall reconstruction error. 32

3.5 The effect of different weight smoothness regularizations. Compared to the graph Laplacian [57, 39], our rigidness Laplacian regularization offers better smoothness while still preserving the global shape. Note that how the graph Laplacian over-smooths the fracture while it starts to change the global shape (indicated by red circles). Our rigidness Laplacian is robust as demonstrated by the similar results with different ω values. 46

3.6 Comparison of the Weighted Absolute Orientation solution [26, 34] on the left and our iterative bone transformations update 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, our iterative 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). 51

3.7 (Left) Without the joint constraints, the bone transformations (bones) generated by [45] do not always rotate around the joints. (Right) With the soft joint constraints, bones rotate more strictly around the joints. Since the bone transformations are alternatively updated with the joint locations, our rigging model can converge to a local optimum with a good approximation of the input. 55

3.8 The redundant bones in the left panel are pruned to achieve the neat skeleton in the right panel. As illustrated in the two yellow boxes, the redundant bone j is identified by utilizing the weight regularization term to force its weights degenerate. 58

3.9 The skeletons extracted from 3 datasets by my approach (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, my approach can robustly determine the optimal number of bones in the skeletons thanks to the skeleton pruning. 61

3.10 The extracted **horse-gallop** skeletons with different cluster initializations. Despite very different numbers of initialized clusters, my 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. 62

3.11 Comparisons between my proposed method and three state of the art approaches. 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**. For a fair comparison, I set the same number of bones for all the four methods. Only my proposed method can generate sound outputs for all the 5 datasets. The issues in the results are indicated by red arrows. 63

3.12 Examples of visual distortion on the reconstructed **hand** poses with respect to different RMSE values (pay attention to the red circled areas). Fig. 3.11 shows the corresponding skeletons. 64

3.13 Results of my proposed LBS skinning without skeleton. My proposed model works well with both the articulated models (the top four models) and the elastic models (the bottom two models). 73

3.14 Comparisons of the skinning decomposition results among my proposed method, SMA [32], and LSSP [25]. 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 are rendered in blue. Significant distortion areas are indicated in red circles. 74

3.15 Detailed computation breakdown of my method on the **samba** model. The number inside a Skeleton-Reconstruction block denotes the number of bones at the current step. The number inside a Rigging-Iterations block denotes the number of iterations executed at the iterative rigging step. Each rigging iteration includes skinning weights update, joint positions update, bone transformations update, and skeleton pruning. The computational times of all the rigging iterations are approximately the same; one detailed breakdown is illustrated in the Last-Rigging-Iteration block. 76

3.16 Despite the high deformation on the skirt part of the **samba** model, my approach is still able to generate a reasonable skeletal structure where the skirt is rigged by some bones originated from the hip. Meanwhile, the three previous approaches cannot extract similar skeleton patterns. 77

3.17 My proposed method can even rig an elastic model such as this stretched **cow**. The top row shows the resulting skeletons in the rest pose. Compared with the reconstructed result by my method, the reconstructed poses by the three previous methods are not visually close to the ground-truth. 78

3.18 My approach with using skeleton LBS fails to remove 4 redundant bones in the upper arms and upper legs of the **scape** model, since the LBS model need them for a better approximation of muscle bulging on these parts (indicated by red arrows). To the end, my approach keeps the 4 bones to capture the non-linear deformation effect. The gray models are the ground-truth. 79

4.1 My compression model blends master bone transformations and caches them as virtual bone transformations (left most). It can compress a 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. 80

4.2 (a): Our skinning compression model can achieve a high performance with insignificant loss of visual quality. (b) and (c): Examples of exceptional vertices (illustrated in red color): the Cheb model (b) is rigged by [7], and a hand model (c) is rigged manually. 82

4.3 A conventional blend skinning model (top left) with a dense weight matrix W (bottom left) is approximated as a two-layer blending with virtual bones (top right). This is equivalent to factorizing W into a sparse dictionary D and a matrix of sparse coefficients A (bottom right). D has at most c non-zero elements, while A has at most 2 non-zero elements. 84

4.4 Some example poses of an animated mesh sequence (left) and its corresponding 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. Our model distributes virtual bones adaptively so that more virtual bones are employed for highly deformed regions (e.g., the legs or the tail). 86

4.5 On a manifold, two neighboring vertices typically have similar skinning weights and coefficients; thus, they most likely share the same optimum virtual bones (dictionary atoms). This assumption can be used to accelerate the update of coefficients. 91

4.6 Example results of our skinning compression model with different thresholds of the objective function (Δ_W and Δ_E). We can hardly observe visual distortions if the error threshold drops below 0.001. The input skinning model (camel gallop) has 15 bones with a dense weight matrix. For each compression example, its bone distribution is illustrated at the bottom-left corner. Master bone transformations are illustrated in red; virtual bone transformations are illustrated in blue. The fitting error on the example poses, E, is also reported. "V. Bones" denotes virtual bones; "b/vtx" denotes bones per vertex. . . . 105

4.7 Relation of the objective function and the fitting error on the example poses. The input skinning model has 15 bones with a dense weight matrix extracted from a camel gallop sequence. The relation between value of the objective function and the fitting error is quite similar for the both cases (i.e., with/without utilizing example poses). Compression utilizing the example poses produces smaller fitting errors than that without utilizing the example poses. 106

4.8 Comparisons between our method and several selected skinning weight reduction methods (i.e., k-largest weight reduction, smooth weight reduction [44], geometry weight reduction [32], and Poisson weight reduction [44]). The three used models are elephant-gallop with 15 bones (top-left), horse-collapse with 20 bones (top-right), and samba with 10 bones (bottom). All the methods reduce the skinning models with dense weights (obtained via the smooth skinning decomposition method [45]) to 4 bones per vertex. The methods noted with blue do not utilize example poses, while the methods noted with red utilize example poses. 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 [45]. . . . 108

4.9 Running time (seconds) of our skinning compression approach on CPU. The first row shows running time of our compression without using example poses and the bottom row shows the running time with using example poses. k denotes the number of master bones, n denotes the number of vertices of the input 3D model, and f is the frame number of example poses. 110

4.10 GPU Performance comparison between LBS and our approach. Both methods were implemented with Microsoft Direct3D 11 running on NVIDIA Geforce GT 540M. Normal vectors were transformed and then normalized to unit length. The performance was measured by animating 150 instances and reporting FPS for the entire pipeline including rendering. 111

4.11 The memory overhead required by our approach to cache virtual bone transformations, compared with the linear blend skinning model with the same number of bones per vertex. Here we assume each vertex is stored as four 32-bit floating point numbers, a transformation matrix is stored as sixteen floating point numbers, and a blending weight with index is stored as two floating point numbers. 112

List of Tables

3.1 Quantitative comparisons among all the four methods. The reported RMSE is normalized by the bounding volume diagonal [66, 25]. Specifically, $RMSE = 100 \times \sqrt{E_D}/d$, where E_D is the data fitting error in Eq. (3.1b), 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. 71

3.2 The test/evaluation datasets used. N denotes the number of vertices and F denotes the number of example poses. 72

3.3 Rigid bone skinning decomposition results of Skinning Mesh Animation with rigid bones (SMA) [32], Learning Skeletons for Shape and Pose (LSSP) [25], and my proposed skinning without skeleton.The result after rank-5 EigenSkin corrections [42] is also reported in the parentheses. 75

4.1 The test datasets used in this work. n denotes the number of vertices, and f denotes the number of example poses. 96

4.2 Comparison of the approximation errors on weight matrix (Δ_W): k-largest weight reduction, smooth weight reduction [44], and our method. In this comparison, example poses are not used. The number of bones k is shown as a subscript of input model name. The errors are multiplied by 1000 for convenience. 107

4.3 Comparison of the fitting errors on example poses (E) among our skinning compression method and several selected weight reduction methods: k-largest weight reduction, geometry weight reduction method [32], the smooth weight reduction method [44], and the Poisson weight reduction method [44]. "SD" denotes the employed original skinning decomposition method [45]. "b/v" denotes bones per vertex. "w/ Ex" (or "w/o Ex") denotes with (or without) utilizing example poses in our approach. 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 k 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. 109

Chapter 1

Introduction

Computer animation is the process of generating motion pictures with the assistance of computer. It plays a very important role in *computer graphics* with tremendous applications on creating movies, video games, and virtual worlds. Perhaps, the most important research subject in computer animation is animating characters such as human, animals, or plants. In most common cases, the rendering of a character to pictures uses the 3D surface geometry of the character such as a *3D triangle mesh*. In order to achieve the best visual effect, the geometry typically has very high spatial resolution. With a large amount of details, the major challenge for the animation pipeline is manipulating the complicated geometry in each time frame. In the early era of computer animation, this task was done by skilled artists with a tremendous amount of time spent on drawing or tweaking the surface geometry at the pixel or vertex level at every time frame. Later on, this problem is handled by *skinning techniques*, in which the idea is mapping a low-dimensional controller

to the high-dimensional 3D geometry shape of the character. The activation of the controller would be transferred to the surface of the character accordingly, generating its articulation and deformation. The controller is manually activated by animators but it can also use the input signals of popular animation control techniques such as *inverse kinematic* or *motion capture*.

Problems. The two fundamental problems of character skinning are *model setup* and *skinning animation rendering*:

- The model setup problem considers the design of the skinning model and the determination the parameters of the skinning model. In the modern animation pipeline, this skinning model setup is typically called *rigging*.
- The skinning animation rendering problem considers the integration of the skinning model into the rendering pipeline. In this pipeline, the skinning model acts as a computing model to take the input of control parameters, generate the 3D surface geometry, and then pass it to fragment shaders for rasterization and shading.

Challenges. Typically, designing a good skinning technique needs to deal with several challenges:

- *Deformation realism.* Reproducing the skinning deformation of creatures such as human and animals is very challenging due to their complicated anatomy structures. The skinning deformation is typically the resulting effect of interactions between many hard and soft layers, e.g. bones, tendons, muscles, flesh,

fat tissue, and skin. For this reason, deformations of some body parts, such as hand, arm, or face, become extremely difficult to model. With these body parts, skinning techniques would typically find challenges in generating effects such as muscle bulges or skin wrinkles.

- *Control effectiveness.* A simple and intuitive controller not only helps for reducing the manual effort on animation manipulation, but also helps for reducing the effort to setup the skinning model. For example, using the skeleton as the controller for articulated models such as a human is a natural way and it makes the setup and animation very effective. Unfortunately, the most simple and intuitive controller typically cannot achieve realistic deformation effects like muscle bulges or skin wrinkles. For many cases, the controller space need to be extended to enhance the deformation space.

- *Computational effectiveness.* Last but not least, the performance of skinning models is very important, especially with real time applications, e.g. animation editing, video games, or virtual worlds. Recent skinning models can take advantage of parallel computing processors such as GPUs or APUs. However, this typically requires special implementations to make them compatible with different graphics pipelines and graphics hardwares.

1.1 Contributions and Organization

This dissertation contributes to both fundamental problems of character skinning, including:

Contributions for skinning model setup. I propose a solution to example-based rigging (chapter 3), which extracts the linear blend skinning (LBS) model from a set of different example poses. The output LBS model consists of sparse, convex and smooth skinning weights and rigid (orthogonal) bone transformations with or without a skeletal structure. I formulate the skinning from examples problem as a minimization of the reconstruction error on the ground truth data with respect to constraints of the LBS model (§3.2). I also present an optimization framework for this problem (§3.3), which includes four technical contributions:

- A fast iterative linear solver to optimize the orthogonal bone transformations (§3.8).
- A problem formulation and optimization method to accurately constrain bone rotations around skeletal joints (§3.5).
- A rigidness Laplacian regularizer to constrain the smoothness of skinning weights (§3.7).
- A robust method to extract the skeletal structure from an over-completed initialized solution (§3.9).

Contributions for skinning animation rendering. I introduce a two-layer linear blend skinning model (chapter 4) which can substantially reduce the computational cost of a dense-weight LBS model with insignificant loss of its visual quality. The introduced model is friendly to the parallel implementation on graphics hardwares such as GPUs. It also allows fast skinning animation without requiring the

sparseness constraint on the skinning weights (dense-weight), which offers more flexibility on applications of the LBS model. The two-layer model is constructed by the sparse coding technique (§4.2) with directly using dense skinning weights or with using additional example poses to further improve its accuracy.

1.2 Publications

This dissertation presents my work published in the following papers:

ROBUST AND ACCURATE SKELETAL RIGGING FROM MESH SEQUENCES

Binh Huy Le, and Zhigang Deng

ACM Transactions on Graphics 33(4), July 2014

(Proceeding of ACM SIGGRAPH 2014)

TWO-LAYER SPARSE COMPRESSION OF DENSE-WEIGHT BLEND SKINNING

Binh Huy Le, and Zhigang Deng

ACM Transactions on Graphics 32(4), July 2013, article 124

(Proceeding of ACM SIGGRAPH 2013)

SMOOTH SKINNING DECOMPOSITION WITH RIGID BONES

Binh Huy Le, and Zhigang Deng

ACM Transactions on Graphics 31(6), Nov. 2012, article 199

(Proceeding of ACM SIGGRAPH Asia 2012)

In additional, the following papers have been published during the period of this

dissertation:

MARKER OPTIMIZATION FOR FACIAL MOTION ACQUISITION AND DEFORMATION

Binh Huy Le, Mingyang Zhu, and Zhigang Deng

IEEE Transactions on Visualization and Computer Graphics (TVCG), 19(11), Nov. 2013, pp. 1859-1871

LIVE SPEECH DRIVEN HEAD-AND-EYE MOTION GENERATORS

Binh Huy Le, Xiaohan Ma, and Zhigang Deng

IEEE Transactions on Visualization and Computer Graphics (TVCG). 18(11), Nov. 2012, pp. 1902-1914

PERCEPTUAL ANALYSIS OF TALKING AVATAR HEAD MOVEMENTS: A QUANTITATIVE PERSPECTIVE

Xiaohan Ma, *Binh Huy Le*, and Zhigang Deng

Proceedings of the SIGCHI International Conference on Human Factors in Computing Systems (CHI) 2011, pp. 2699-2702

STYLE LEARNING AND TRANSFERRING FOR FACIAL ANIMATION EDITING

Xiaohan Ma, *Binh Huy Le*, and Zhigang Deng

Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA) 2009, pp. 114-123

AN INTERACTIVE GEOMETRIC TECHNIQUE FOR UPPER AND LOWER TEETH SEGMENTATION

Binh Huy Le, Zhigang Deng, James Xia, Yu-Bing Chang, and Xiaobo Zhou
Proceedings of the 12^{th} International Conference on Medical Image Computing and Computer-assisted Intervention (MICCAI) 2009, pp. 968-975

Chapter 2

Background on Character Skinning

2.1 Geometric Skinning

Geometry-based methods are the simplest skinning techniques. Although their simplicity limits the deformation realism, geometric methods have several advantages such as easy for implementation, high performance, and well supported by mathematical models. For these advantages, geometry-based is still one of the most widely used skinning methods.

2.1.1 Skeleton-based Skinning

This skinning model uses the anatomic skeleton of the subject as the controller. For the sake of convenience, the skeleton can be simplified. In common practice, the skeleton of any articulated object is typically represented as a rooted tree, where

each tree node corresponds to a joint and each tree edge corresponds to a bone (Fig. 2.1(a)). Any two bones connected at a common joint are allowed to rotate relatively together around the common joint. The joints are either ball joints with 3 degree of rotations or hinge joints with 2 degree of rotations. Animation is then controlled by specifying the rotation angles of all the joints, or posing the skeleton in other words.

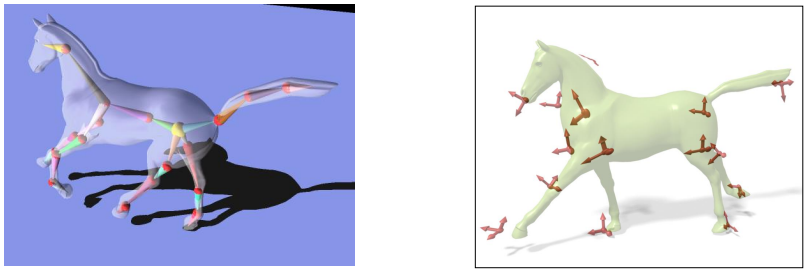

Figure 2.1: (a) Skeleton-based animation (images courtesy of [14] ©EUROGRAPHICS 2008). (b) Skeleton-based animation with bones represented by their transformations (images courtesy of [46] ©ACM 2013)

Linear Models

Among many proposed skinning techniques, *linear blend skinning* (LBS) is widely known to be the most popular skinning computational model due to its effectiveness, simplicity, and efficiency [50, 18]. It has many different names over the years. For example, in existing 3D modeling and animation tools, it is called *smooth skinning* (Autodesk Maya), *bones skinning* (Autodesk 3D Studio Max), or *linear blend skinning* (the open-source Blender). In the research community, this technique have other distinct names, including *skeleton subspace deformation*, *enveloping*, or *vertex blending*.

In the LBS model, skin deformation is driven by a set of bones. Every vertex is associated with the bones via a bone-vertex weight map which quantifies the influence of each bone to the vertices. The skin is deformed by transforming each vertex through a weighted combination of bone transformations from the rest pose. Assuming w_{ij} is the influence of j-th bone to the i-th vertex, p_i is the position of the i-th vertex at the rest pose, $|B|$ is the number of bones, and R_j^t and T_j^t are the rotation matrix and translation vector of the j-th bone at the t-th configuration, respectively, then the deformed i-th vertex, v_i^t, can be computed as follows:

$$v_i^t = \sum_{j=1}^{|B|} w_{ij}(R_j^t p_i + T_j^t) \tag{2.1}$$

Depending on specific applications, the above LBS model may impose certain constraints. The weight map w_{ij} is normally required to be convex, i.e., $w_{ij} \geq 0$ and $\sum_{j=1}^{|B|} w_{ij} = 1$. The first *non-negativity* constraint makes the transformation blending additive. The second *affinity* constraint normalizes the influences/weights to prevent over-fitting and deformation artifacts. The two constraints are critical for certain applications such as animation editing. In addition, the *sparseness* constraint on the weight map, which limits the number of non-zero bone weights per vertex, may be applied to take advantage of graphic hardware capabilities. Many applications also require R_j^t matrix to be orthogonal, e.g. animation editing, collision detection, and skeleton extraction. The *orthogonal* constraint avoids any shearing or scaling effect on the bone transformations, thus put the transformations into the rigid group. For this reason, the bone transformation with orthogonal rotation matrix is also called

the "rigid bone".

Non-linear Models

Despite its wide uses, the linear skinning model has certain limitations. For example, it suffers from artifacts such as the *collapsing elbow* or the *candy-wrapper* effects [18]. These artifacts are generated because the volume of the object is not preserved due to the linear blending of rotations create a non-orthogonal matrix.

One popular technique to handle this issue is interpolating the rotation and translation parts individually [38]. While the translation interpolation is done in the linear manner, interpolating the rotation part is non-linear. Kavan et al. [38] represent a rotation matrix as a *quaternion*. A quaternion is a four-dimensional vector space over the real numbers. The four bases of a quaternion are denoted as 1, i, j, k such that

$$i^2 = j^2 = k^2 = ijk = -1$$

With the four basis, any quaternion q can be represented by

$$q = a + bi + cj + dk$$

where a, b, c, d are four real numbers

The norm of quaternion q is defined by

$$||q|| = \sqrt{a^2 + b^2 + c^2 + d^2}$$

A quaternion q with norm $||q|| = 1$ is called a *unit quaternion*. A unit quaternion has 3 degree of freedoms. Any rotation in the 3D Euclidean space, i.e. $SO(3, \mathbf{R})$, can be represented by a unit quaternion. Given two rotations in form of quaternion q_1 and

q_2 and its center of rotation, q_1 and q_2 can be linearly blended by *spherical linear interpolation* (SLERP) [68]. Extending the problem to $n > 2$ rotations, Kavan et al. [38] first find the center of rotation by solving the least square of the drifting between any two rotations. Then, n rotations are blended by *quaternion linear interpolation* (QLERP). Although this method preserves the volume of the object by keeping the interpolation in the orthogonal space, it requires additional computation for finding the center of rotation.

Later in 2008, Kavan et al. [38] addressed the limitation of quaternion blending by employing the *dual-quaternion*. A dual-quaternion is a quaternion with four bases are *dual numbers*. A dual number $\hat{a}$ has two bases 1 and ϵ such that $\epsilon^2 = 0$. Thus, a dual-quaternion is determined by a tuple of eight real numbers. In other words, a dual quaternion $\hat{q}$ can be represented as the sum of two ordinary quaternion, a non-dual part q_0 and a dual-part q_ϵ, as follow

$$\hat{q} = q_0 + \epsilon q_\epsilon$$

The norm of dual quaternion $\hat{q}$ is defined by

$$||\hat{q}|| = ||q_0|| + \epsilon \frac{\langle q_0, q_\epsilon \rangle}{||q_0||}$$
$$\text{where } \langle q_0, q_\epsilon \rangle \text{ is the dot product}$$

A dual-quaternion $\hat{q}$ with norm $||\hat{q}|| = 1$ is called a *unit dual-quaternion*. Any rigid transformation in the 3D space can be represented as a unit dual-quaternion. In contrast with the quaternion, blending dual quaternions can be done directly in a linear manner, follow by a normalization to the unit dual-quaternion.

Compared with the original linear matrix blending, blending in a quaternion or

dual quaternion form preserves more rigidity of the transformation and thus reduces the linear deformation artifacts. However, this advantage goes with a performance trade off. The performance of quaternion blending is slowest due to the cost of calculating centers of rotation. The dual-quaternion blending is faster than quaternion blending but still slower than linear blend skinning due to the overhead of converting transformation matrices to dual-quaternions and vice versa.

Multi-dimensional Models

The intrinsic limitation of the linear skinning model can also be tackled by expanding the weight matrices into multi-dimensional.

The multi-weight enveloping technique [77] modifies the linear blend skinning by using an individual weight for each entry of the transformation matrix. As the result, the influence of a bone transformation to a vertex is represented by a 3×4 matrix. This extension allows the linear blending model to capture some complicated deformation such as muscle bulges. Due to its high dimension, the weight map is solved from examples by a linear least squares. The result can capture the deformation of the input well, but it often suffers from over fitting. In addition, the weight map is non-intuitive for the artist, thus making manual tuning challenging.

Jacobson et al. [31] proposed *stretchable and twistable bones* for skeletal shape deformation. On top of the linear blend skinning, this model adds a scalar weight function to each bone. The additional weights can be computed automatically from the skeletal structure, or they can be painted manually in an intuitive way. The additional weights have enriched the deformation space significantly. However, this method only works with skeletons with given structure, since the weight function

need to be calculated based on the two ends of each bone.

2.1.2 Cage-based Skinning

Cage-based skinning was inspired by the *free-form deformation* (FFD) [67] where the space of the object is embedded into a grid-based deformation control. By manipulating the grid nodes, the embeded object can be deformed (Fig. 2.2(a)). By extending the grid to an arbitrary triangular mesh, the controller can be shaped to approximate the model surface (Fig. 2.2(b)). The controller is the *cage* and its vertices are the control points. The position of each vertex on the model is then computed as a linear blend of positions of some cage vetices. The weights of this linear blending is sometime called the coordinates of the vertex.

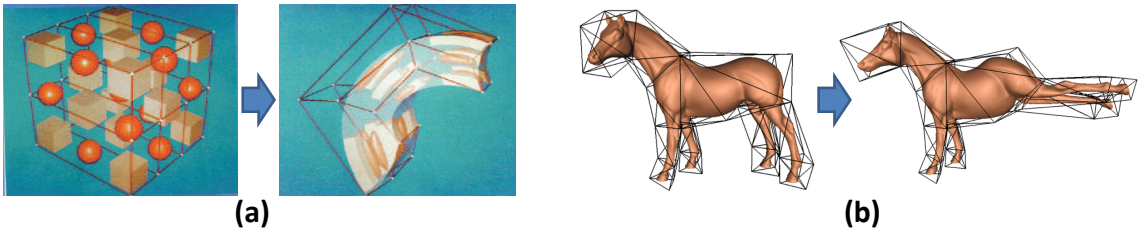

Figure 2.2: (a) Free-form deformation (images courtesy of [67] ©ACM 1986). (b) Cage-based deformation (images courtesy of [33] ©ACM 2005)

Compared to the sksleton-based skinning, cage-based skinning offers more flexible control on deformation. For example, muscle bulge effects can be easily created. However, buiding the cage for a particular model is time consuming. In addition, cage-based skinning cannot take the full advantage of the hardware-based implementation due to the dense influence of the cage vertices to the model vertices.

2.1.3 Skinning from Single Shape

Researchers have pursued two different directions to extract skeletons and the skinning weights from a single static pose, serving for different purposes. In one direction, curve skeleton extraction [6, 72, 52, 27] is typically focused on discovering the topology of the skeleton (e.g., handle or loop) rather than its exact shape. In the other direction, some methods focus on generating skeletons for animation [7]. Often, this application requires the skeleton to be simple, where each bone is typically a line segment.

Automatic methods have been also proposed to generate the skinning weights from a static 3D model [7, 30]. On top of these skinning weights, artists can also make manual adjustments (weight painting). However, it is non-trivial to obtain best sets of weights for realistic skinning, especially for highly deformable models, since the only input of these methods is a static shape of the object.

2.2 Anatomical Simulation Skinning

Realistic skinning can be achieved by setting up anatomical models, and then performing simulation [69, 48, 56, 22, 62]. Compared to the geometry-based skinning, simulation methods offer several advantages such as collision handling and secondary deformation. Although most of standard 3D modeling/animation tools, e.g. *Maya* or *Blender*, support physical simulation really well, designing an anatomical model as well as setting up simulation parameters is not an easy task. Acquiring a full scan

of a body with all the internal organs material properties is typically very costly or even impossible for many of non-human creatures. Recently, Ali-Hamadi et al. [3] proposed a semi-automatic approach to transfer anatomy structures that could be used for quick animation setup. However, this method only works with human-like creatures. Another limitation of simulation methods is performance. Although a lot of effort has been put for reducing the computational complexity [59, 29, 23], the performance of simulation methods are still very far away from the real-time requirement.

2.3 Example-based Skinning

2.3.1 Input Data

Several early related efforts were pursued to accurately model the high fidelity skinning deformation by using *example poses*. The example poses might be generated by artists, or achieved by running physical simulation.

2.3.2 Skinning Weights Fitting

Typically, this approach uses the example poses as the training data to find the optimum weights for the skinning model. The major works in this direction focus on linear skinning models.

For example, Wang and Phillips [77] employ linear least squares to calculate the

multi-dimensional weights given a set of examples poses and their corresponding bone transformations. Mohr and Gleicher [58] fit the affinity weights (the sum of weights for each vertex equals to 1) for linear blend skinning from examples.

The skinning from examples problem might be solved by regression technique. Wang et al. [76] use a linear regression model to learn the correlation between bone rotations and rotations of triangles on the surface of the object. The prediction triangle rotations are used as the input for *Poisson surface reconstruction* [5] on the skinning phase. Feng et al. [16] employ *Kernel Canonical Correlation Analysis* to learn the relation between control handle transformations and the surface deformation.

The first reported solution to compute both bone transformations and bone-vertex weights from example poses is the *skinning mesh animations* (SMA) method [32]. In the SMA model, the bone transformations are determined by first clustering the triangle rotation sequences and then associating each cluster transformation to a bone. Then, the bone-vertex weights are calculated by linear least squares with soft constraints. Although the SMA model provides a complete solution to the skinning decomposition problem, essentially, it is still a combination of two separate solutions of sub-problems rather than a unified framework. SMA is more effective for near-rigid (e.g., articulated) objects than non-rigid models since the triangle rotations can be clearly observed in the nearly-rigid objects. The SMA model is able to enforce hard constraints on bone rotation matrices; however, it can only enforce soft constraints on the bone-vertex weights (i.e., the affinity constraint). Due to this reason, it has to face a trade-off between guaranteeing the affinity constraint on the weights and

sacrificing the reconstruction error.

Later, following this direction, Hasler et al. [25] first segment the meshes of example poses into rigid parts by applying spectral clustering to the triangle rotations. Then, the bone transformations and bone-vertex weights are refined via an iterative process, where the bone transformations are optimized by linear least squares with soft constraints enforced on rotation matrices. The bone-vertex weights are solved by non-negative linear least squares with L1-norm minimization to achieve sparse solutions, followed by L1-norm normalization in order to enforce the affinity constraint. Since all the employed constraints are soft, the iterative process cannot guarantee its convergence in theory. In addition, the results need some corrections in order to fulfill the requirement of being rotation matrices as well as being a convex weight map.

Skinning for Hardware-accelerated Rendering

The skinning from examples technique can be used to accelerate the hardware-supported rendering by reducing the control data transferred between CPU and GPU. Kavan et al. [36] proposed an efficient solution to compute the compact set of bone transformations in the dual quaternion form based on an arbitrary set of example poses and the corresponding weight map computed by a simple smooth interpolation of an even distributed bones over the surface of the object. During the rendering phase, bone transformations are sent to the GPU instead of the original geometry, which can save a significant amount of data transfer.

Kavan et al. [37] target on the reduction of the control data transferred between CPU and GPU by a matrix decomposition manner. In their model, the vertices of

example poses, denoted as a matrix A, are decomposed to $A = TX$, where T represents transformations and X represents the combination of the weight matrix and the rest pose. Their approach does not enforce any constraints on the transformations T and thus their approach is suitable (and often limited) to compression and GPU-accelerated high performance rendering applications.

2.3.3 Skeleton Extraction

Using motion data, the skeleton and joint locations can be determined more easily than using a static pose alone. Anguelov et al. [4] proposed an early work to deal with unorganized point cloud data. Due to the lack of temporal coherence, their method is primarily designed to identify and track rigid components of the model. Kirk et al. [40] use marker-based mocap data as the input for skeleton extraction, exploiting the temporal coherence between mocap frames. Since both of the methods use low spatial resolution data, they only focus on solving the joint locations between two rigid body components, discarding the blending between bodies. Later, several methods have been proposed to work with animated mesh sequences [66, 14, 25]. Since an animated mesh sequence offers much higher spatial resolutions than mocap data, they can often produce quality skeletons along with the LBS model.

2.3.4 Basis Models

Skinning using basis models represents the shape or the deformation of the object as an interpolation of basis shapes.

Blend Shape is a linear interpolation technique where the object shape is then determined by the weighted linear combination of the basis [49].

One common way to build the bases shapes is performing *Principal Component Analysis* [63] on example poses. PCA maps the input data with possibly correlated variables into an uncorrelated variable space via an orthogonal linear transformation. PCA is widely known to be one of the most common dimension reduction techniques that attempt to minimize the reconstruction error. In computer graphics applications, PCA and its various variations have been used for segmenting human motion capture data into distinct behaviors [8], compressing animation sequences [64], human motion data retrieval [17], and recovering missing markers in motion capture data [61].

Pose-space Deformation. Lewis et al. [50] pioneered the *pose-space deformation* (PSD) that treats the deformation problem as a shape interpolation problem from a set of example poses, and it allows animators intuitively manipulate the set of control poses and add additional poses to correct the deformation artifacts. Compared to the blendshape model, PSD expands the deformation to the non-linear space, thus, it can easily capture the detailed deformation [10].

Skinning Correction Example poses can also be used to correct the skin deformation on top of any other skinning method. In the EigenSkin model [42], the deformation error is corrected by mapping back the per-vertex displacement error from each key pose to the rest pose. Then, the displacement error is represented in the Eigen space to reduce the storage. This compact representation allows the deformation correction to be calculated in an effective manner with a low band-width.

Chapter 3

Example-based Rigging

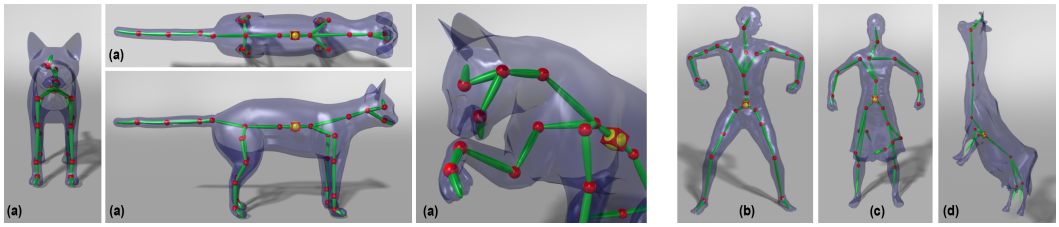

Figure 3.1: Only using a single set of parameters, my example-based method can accurately rig various models such as quadrupled animals (a), humans (b, c), and highly deformable models (d). My method can even generate bone structures for challenging parts, such as the mouth and the two ears of the cat (a), the skirt (b), and the elastic cow model (d). My method is robust; using only 9 frames, it can generate a skeleton with 28 bones for the cat model (a); note that even though the given example poses of the cat model have asymmetric poses, it still can generate an almost symmetric skeleton without imposing any symmetry constraints.

Setting up the skeleton-based animation (also known as *rigging*), has become

one of the major steps involved with tedious non-trivial manual interventions in industry practice. Rigging a model currently consists of two main steps: building a hierarchical skeleton with rigid bones connected by joints, and skinning the 3D model to define how joint rotations and translations would propagate to the surface during animation. In practice, animators typically repeat the two steps many times to refine the model for best results. This trial-and-error method is costly and time-consuming, since its two steps are often done manually or semi-automatically.

The concept of using example poses (i.e., a sequence of deformed mesh frames) for rigging [66, 14, 25] has become increasingly applicable and useful in recent years. In particular, deformed mesh sequences can be soundly reconstructed by performance capture [15, 74, 75, 70] or by dense motion capture based on commercial systems [61]. Since the skeleton extracted from example poses is typically compatible with game engines and popular animation software such as Autodesk Maya, it can be straightforwardly used for various animation editing, compression, and rendering applications, which helps to reduce production cost in industry practice.

The essential idea of these example-based rigging methods [66, 14, 25] is: first perform motion-driven clustering to extract rigid bone transformations, then estimate joint locations and bone lengths using linear or non-linear least squares, and finally optimize the bone transformations and skinning weights. However, they have the following limitations: first, nearly-rigid parts cannot be identified perfectly since motion-driven clustering algorithms model neither skin blending nor skeletal structure. As such, this step would either require non-trivial model-specific parameter tuning or result in an non-robust skeleton. 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, or after the joint locations are determined, joint constraints would change the bone transformations generated at the previous step. To the end, errors could be significantly accumulated (e.g., from the root joint to leaf joints). Due to these issues, these rigging methods have limited accuracy and robustness and therefore fall short of meeting the demand. Examples in Fig. 3.2 show two common limitations of these rigging methods.

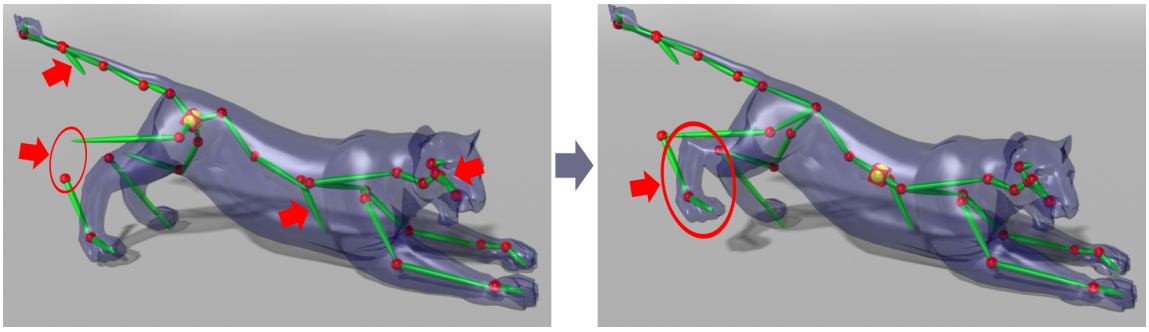

Figure 3.2: Examples to show two common limitations of current rigging methods: (i) an over-estimated bone initialization may generate inaccurate and redundant bones (indicated by red arrows); (ii) an inaccurate estimation of bone transformations (indicated by red arrows with ellipse) causes noticeable deformation errors when the bones are connected by joints (see the change of the legs). The examples in this figure are generated using [25].

In this chapter, I address the above limitations and introduce a robust and accurate rigging framework that takes a set of example poses as input and produces its corresponding *Skeleton-based Linear Blend Skinning* (LBS) model. The obtained

LBS model includes skeletal structure, skinning weights, joint locations, and bone transformations corresponding to all the example poses. This method can robustly generate high-quality rigging models from a small set of example poses. As shown in Fig. 3.1, my method can robustly generate high-quality rigging models from a small set of example poses. Compared to previous methods [66, 14, 25], it offers the following two advantages:

- **Robustness.** By automatically pruning redundant bones, my approach is more robust than the previous methods that often suffer from an over-estimated bone initialization.
- **Accuracy.** By formulating the solving of a LBS model with skeleton as a constrained optimization, my iterative approach can obtain more accurate joint locations, bone lengths, corresponding joint rotations, as well as the reconstruction of the example poses without accumulation of fitting errors.

The key component of my method is an *Iterative Rigging* algorithm (§3.3), which alternatively updates skinning weights (§3.7), joint locations, and bone transformations (§3.8), and automatically prunes redundant bones (§3.9).

3.1 Notations

Input. The input data for my rigging method is an animated mesh sequence, a.k.a. example poses, with temporal coherence between vertices, i.e. meshes in different frames share the same topology. Let F be the number of frames (or example poses)

in the sequence, N be the number of vertices in each mesh, and $v_i^f \in \mathbb{R}^3$ be the 3D position of the vertex i in frame f. The input mesh sequence also includes the rest pose mesh (a.k.a., the dressing pose), where the position of the vertex i in the rest pose is denoted by $u_i \in \mathbb{R}^3$.

Output. The output of the algorithm is the LBS model with skeletal structure, joint positions, skinning weights, and bone transformations with respect to the input poses.

The skeletal structure is an undirected tree $\mathbb{S}$ with B bones as nodes of the tree and $B-1$ joints as edges of the tree. For the sake of convenience, we write $(j,k) \in \mathbb{S}$ to denote (j,k) is an edge of the tree $\mathbb{S}$, or 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 C_{jk}, which is also the center of rotation of two bones j and k.

The skinning weights, a.k.a. bone-vertex influences, $W = \{w_{ij}\}$ is a $N \times B$ matrix, where w_{ij} denotes the influence of the j-th bone on the i-th vertex. The skinning weights W are constrained to be *sparse* and *convex*. The sparseness constraints allows no more than 4 non-zero weights per vertex, i.e. $\|w_i\|_0 \leq 4, \forall 1 \leq i \leq N$ where $w_i \in \mathbb{R}^N$ denotes the weights of vertex i. The convexity constraints includes non-negativity constraints, i.e. $w_{ij} \geq 0, \forall 1 \leq i \leq N, 1 \leq j \leq B$, and affinity constraints, i.e. $\sum_{j=1}^{B} w_{ij} = 1, \forall 1 \leq i \leq N$.

For each input example pose, my rigging algorithm also outputs the optimum bone transformations corresponding to the example pose. The transformation matrix of the j-th bone in the f-th example pose is $\left[R_j^f | T_j^f\right]$, where R_j^f denotes the 3×3

rotation matrix and T_j^f denotes the 3×1 translation vector. The rotation matrix R_j^f satisfies the orthogonal constraint, i.e. ${R_j^f}^\mathsf{T} R_j^f = I$ and $\det R_j^f = 1$, where I is a 3×3 identity matrix.

3.2 Problem Formulation

$$\min_{\mathbb{S}, C_{jk}, w_{ij}, [R_j^f | T_j^f]} E = E_D + \omega E_S + \lambda E_J \tag{3.1a}$$

$$\text{Where: } E_D = \frac{1}{NF} \sum_{i=1}^{N} \sum_{f=1}^{F} \left\| \sum_{j=1}^{B} w_{ij} [R_j^f | T_j^f] \begin{bmatrix} u_i \\ 1 \end{bmatrix} - v_i^f \right\|_2^2 \tag{3.1b}$$

$$E_S = \sum_{j=1}^{B} {w_j}^\mathsf{T} L w_j \tag{3.1c}$$

$$E_J = \frac{1}{F} \sum_{(j,k) \in \mathbb{S}} \sum_{f=1}^{F} \left\| \left([R_j^f | T_j^f] - [R_k^f | T_k^f] \right) \begin{bmatrix} C_{jk} \\ 1 \end{bmatrix} \right\|_2^2 \tag{3.1d}$$

$$\text{Subject to: } w_{ij} \geq 0, \ \sum_{j=1}^{B} w_{ij} = 1, \ \|w_i\|_0 \leq 4, \ \forall i, j \tag{3.1e}$$

$${R_j^f}^\mathsf{T} R_j^f = I, \ \det R_j^f = 1, \ \forall j, f \tag{3.1f}$$

We minimize the objective function in Eq. (3.1a) to find the optimized LBS model with skeletal constraints. The objective function E includes the following 3 terms:

- *Data fitting term* E_D (Eq. (3.1b)) 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. (3.1c)) favors the smoothness of skinning weights (§3.7) and drives the removal of redundant bones (§3.9). We derive this term from the fairness and smoothness conditions on the manifold, which is similar to [39]. $w_j \in \mathbb{R}^N$ is the column vector of N skinning weights of bone j, and $L \in \mathbb{R}^{N \times N}$ 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 (refer to Eq. (3.11a) for detailed definition of L).

- *Joint constraint term* E_J (Eq. (3.1d)) keeps any two connected bone transformations $\in \mathbb{S}$ rotate around their common joint. This soft constraint is derived from [4, 66]. If two bones j and k share the common joint (j, k), this constraint favors the rest-pose position of the joint, C_{jk}, going to the same position after bone j transformation and bone k transformation (also refer to the explanation of Eq. (3.5) in §3.6).

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. (3.1e)), and includes *orthogonal constraints* (Eq. (3.1f)) on the bone transformations R_j^f (all the bone transformations need to be rigid).

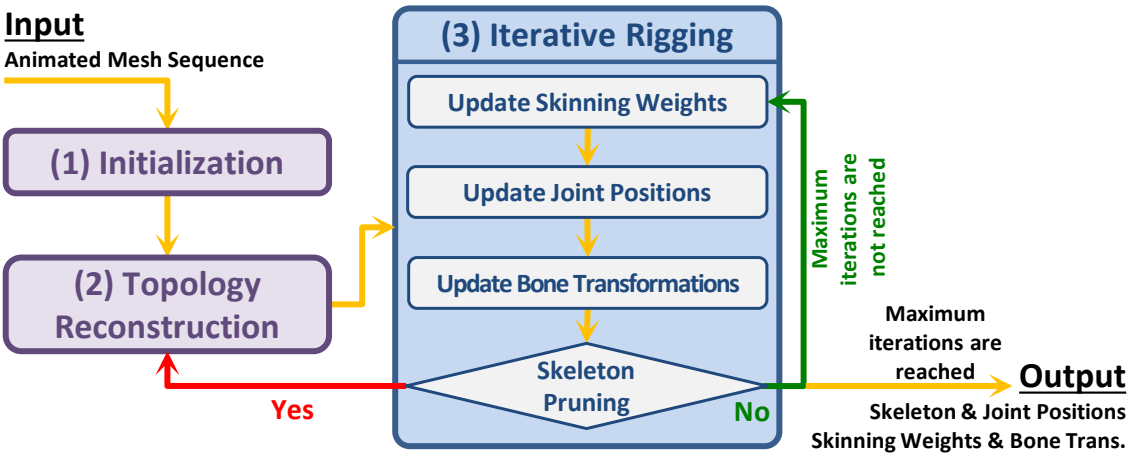

Figure 3.3: The pipeline overview of our skeletal rigging approach.

3.3 Algorithm Overview

From the input data, our method generates skinning weights, skeleton structure, joint locations, and corresponding bone transformations using the following three main steps, as illustrated in Fig. 3.3.

Initialization (§3.4). From the animated sequence, we first identify near-rigid parts and use each part to initialize one rigid bone transformation sequence. In this step, we favor to produce an over-estimated number of bones. Redundant bones will be later removed at the skeleton pruning step (§3.9).

Topology reconstruction. Using the initialized bone transformations, and skinning weights to obtain the LBS without skeleton. Then, we reconstruct the skeleton topology using minimum spanning tree algorithms on a weighted graph that is inferred from the LBS (§3.6).

Iterative rigging. The main component of our approach employs a block coordinate descent strategy [9]. This step alternatively updates skinning weights, joint positions, and bone transformations while keeping the remaining blocks fixed. Specifically, the weights update minimizes E with respect to the data fitting and weight regularization terms (§3.7); the joint positions update minimizes the joint constraint term using [4]; the bone transformations update minimizes the data fitting and joint constraint terms (§3.8). We repeat this alternative update process for a user-specified maximum number of times, which is empirically set to 20 in our experiments. During this process, we always prune redundant bones (§3.9) after each bone transformations update step. If a redundant bone is pruned, we will restart the iterative update process by resetting the iteration counter to zero. In Eq. (3.1a), ω is a constant determined by the Laplacian matrix (§3.7); λ starts with 1 and is multiplied by 1.5 after each iteration.

3.4 Initialization

Inspired by the work of [2, 28] that deformations should be *as-rigid-as-possible* in order to achieve realistic visual results, our initialization assumes no surface deformations appear. In other words, there is no bone-vertex weight blending. With this assumption, 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 clusters, and all the vertices in one cluster follows the same rigid transformation.

For each cluster j of the animated vertices, we fit a rigid bone transformation

$[R_j^f|T_j^f]$ to relate the vertex positions at the rest pose to the vertex positions at frame f, where $R_j^f \in \mathbb{R}^{3\times3}$ is an orthogonal rotation matrix and $T_j^f \in \mathbb{R}^3$ is a translation vector.

Motion-driven vertex clustering. We apply the *Linde-Buzo-Gray* (LBG) algorithm [51] to cluster vertices with similar rigid transformations. We first initialize one cluster to include all the vertices. Then, by extending the *cluster splitting-EM strategy* [51] to handle the mesh sequence data, we divide this initial cluster into two clusters. Afterward, the same cluster splitting-EM strategy is repeatedly called to further generate 4, 8, 16, and 32 clusters. We stop splitting at 32 clusters since all the datasets used in this paper have no more than 32 bones. However, it is noteworthy that the number of clusters can be more than 32 if needed (refer to Fig. 3.10), e.g., complicated models are inputted.

Cluster splitting-EM. This strategy consists of two sequential steps: (i) splitting one cluster into two, and then (ii) refining the resulting clusters using Expectation-Maximization (EM). The details of the two steps are described below.

- *Cluster splitting.* Since we perform the motion-driven clustering without explicitly computing the feature vector for each vertex, the feature-based cluster splitting step in the original LBG algorithm [51] cannot be directly applied. Instead, we design a new cluster splitting scheme by considering the following two criteria: (i) minimizing the approximation error, and (ii) keeping vertices in the same cluster close together. To split a cluster j, we first compute O, the rest-pose centroid of all the vertices belonging to j. Then, we find the seed

vertex s belonging to j such that the product of its reconstruction error, $e(s)$, and the distance from its rest-pose position to the rest-pose centroid, $d(u_s, O)$, is maximum. Eq. (3.2a) shows its computing process, where $\mathcal{L}(i)$ denotes the cluster label of vertex i. Finally, for all the vertices belonging to j, we use the Euclidean distances from their rest-pose positions to u_s as the criterion to evenly split them into two sets. Conceptually, we want to use the seed s and its neighbors to initialize the new cluster centers (i.e., bone transformations). The seed s is chosen as the off-center vertex with a large reconstruction error so that the resulting new clusters can maximally reduce the overall reconstruction error, as illustrated in Fig. 3.4.

$$s = \arg\max_i \{d(u_i, O)e(s)\} \text{ s.t. } \mathcal{L}(i) = j \tag{3.2a}$$

$$\text{where: } d(u_i, O) = \|u_i - O\|_2$$

$$e(i) = \sqrt{\frac{1}{F}\sum_{f=1}^{F} \left\| [R_j^f | T_j^f] \begin{bmatrix} u_i \\ 1 \end{bmatrix} - v_i^f \right\|_2^2} \tag{3.2b}$$

- *Cluster refining (EM).* Similar to the K-means clustering algorithm, we can refine the clusters after each split. Assuming each cluster j is represented as a sequence of F bone transformations $[R_j^f | T_j^f]$ ($1 \leq f \leq F$ corresponding to the F example poses), the refinement process includes two alternative steps, expectation and maximization . First, in the *maximization step* (a.k.a. assignment step), we associate each vertex to the bone which has the smallest squared reconstruction error. Then, in the *expectation step* (a.k.a. updating

step), we calculate the bone transformation based on all the associated vertices by the Kabsch algorithm [34]. In this way, we find the best transformation for pose f to relate the set of the associated vertices in pose f and the rest pose. This assignment and updating process is repeated for several times (e.g., 5 times in our experiments) to achieve a good feasible clustering. To ensure the robustness, we remove all the insignificant clusters after each updating step, each of which has members (i.e., vertices) fewer than 0.1% of the total number of the mesh vertices.

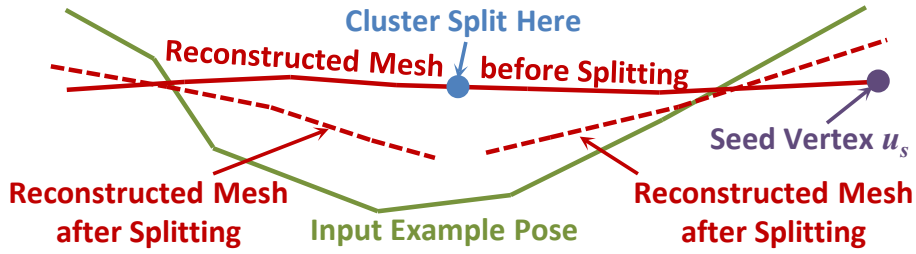

Figure 3.4: An illustration of our cluster splitting strategy. The seed vertex is chosen as the off-center vertex with a large reconstruction error so that the resulting new clusters can maximally reduce the overall reconstruction error.

Connected patches generation. The motion-driven clustering algorithm only assigns a vertex to the cluster using the smallest reconstruction error criterion, ignoring the mesh connectivity information. Thus, more than one rigid parts might be assigned to one cluster if they have similar motions. To keep only one rigid part per bone, we simply find all the connected components (i.e., patches of vertices) for each cluster, and then initialize a new rigid bone transformation for each connected component.

The LBG algorithm is as fast and simple as K-means clustering, yet it is still sufficiently robust compared to more costly solutions such as Mean Shift clustering [32] and Agglomerative clustering [66]. Having a few redundant clusters at the initialization step would affect our final result negligibly at most, thanks to our skeleton pruning step that removes redundant bones. Thus, an accurate estimation of the number of clusters is not required at the initialization step in our approach.

3.5 Joint Constraints and Joint Positions Solver

3.5.1 Joint Constraints

We use the constraints proposed by [4] to make bone transformations rotate around common joints. Assumming the two bones j and k share the same joint C_{jk}. Intuitively, C_{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. (3.3) will be close to zero.

$$\sum_{f=1}^{F} \left\| \left([R_j^f | T_j^f] - [R_k^f | T_k^f] \right) \begin{bmatrix} C_{jk} \\ 1 \end{bmatrix} \right\|_2^2 \rightarrow 0 \tag{3.3}$$

3.5.2 Joint Positions Solver

Given the two bone transformations sequences $[R_j^f | T_j^f]$ and $[R_k^f | T_k^f]$, we can find the common joint of bone j and bone k by minimizing the joint fitting error (3.3).

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 minimum solution of (3.3). To make the solution unique, we use the solution proposed by Anguelov et al. [4], which add a soft constraint to minimize the distance between joint position to the centroid of two bones:

$$\min_{C_{jk}} \sum_{f=1}^{F} \left\| \left([R_j^f | T_j^f] - [R_k^f | T_k^f] \right) \begin{bmatrix} C_{jk} \\ 1 \end{bmatrix} \right\|_2^2 + \mu \left\| G_j + G_k - 2C_{jk} \right\|_2^2 \tag{3.4}$$

Here, the two centroid G_j and G_k are the weighted centroid of all the vertices u_i with the weights are the skinning weights w_{ij} or w_{ik}. The parameter μ control the trade off between joint fitting error and joint position error. μ is set to a small number i.e. 0.01.

For the sake of robustness, we suggest to compute the center of rotation between two bones using [4], instead of [66]. We found that the pseudo-inverse solver in [66] tends to find the wrong subspace if the bone rotations are non-robust. In contrast, the regularization term in [4] only depends on the centroid of two bones, which is more robust to estimate. As the trade-off, [4] gives a slightly larger approximation error than [66].

3.6 Skeleton Reconstruction

We reconstruct the skeleton topology using both bone transformations and mesh connectivity information. First, we run the alternative skinning weights update (§3.7) and bone transformations update without skeleton (§3.8) for 10 iterations to refine the bone transformations and generate skinning weights from the initialization step. To improve the robustness, we start with only one non-zero weight per vertex and increase the number of weights-per-vertex by one every 3 iterations. The output of this step is bone transformations $[R_j^f|T_j^f] \in \mathbb{R}^{3\times4}$ and skinning weights $w_{ij} \in \mathbb{R}$, where i, j, and f denote a vertex index, a bone index, and a frame index, respectively.

We construct a weighted graph $\mathbb{G}$ with B nodes (corresponding to B bones). The weight $g(j,k)$ between bone j and bone k is computed by Eq. (3.5), 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, C_{jk}, after the two transformations of bone j and bone k. Here, the center of rotation C_{jk} between two bones j and k is defined at the rest pose, and its position is computed by [4] as described in §3.5. Since C_{jk} is the point such that its locations after both bone transformations are most similar in all frames; thus, the joint fitting error (the numerator in Eq. (3.5)) measures the quality of having the joint between bone j and bone k. The blending of two bones (the denominator in Eq. (3.5)) 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.

$$g(j,k) = \frac{\sum_{f=1}^{F} \left\| \left([R_j^f | T_j^f] - [R_k^f | T_k^f] \right) \begin{bmatrix} C_{jk} \\ 1 \end{bmatrix} \right\|_2^2}{\sum_{i=1}^{N} w_{ij} w_{ik}} \tag{3.5}$$

Finally, we determine the skeleton $\mathbb{S}$ as the *Minimum Spanning Tree* (MST) of $\mathbb{G}$ [41], which is similar to the idea of previous methods [40, 66, 14, 25]. For the sake of convenience, we write $(j,k) \in \mathbb{S}$ to denote (j,k) is an edge of the tree $\mathbb{S}$, or 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 C_{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. For the sake of visualization purpose, we simply set the root as the joint that is the closest to the centroid of the rest pose for visualization purpose. Given the root of the skeleton and each bone transformation, we can straightforwardly compute the joint rotations and bone lengths. However, to ensure the readability of this paper we only use the raw representation of the skeleton, that is, an unrooted skeleton with bone transformations.

For the sake of robustness, we suggest to compute the center of rotation between two bones using [4], instead of [66]. We found that the pseudo-inverse solver in [66] tends to find the wrong subspace if the bone rotations are non-robust. In contrast, the regularization term in [4] only depends on the centroid of two bones, which is

more robust to estimate. As the trade-off, [4] gives a slightly larger approximation error than [66].

3.7 Skinning Weights Update

3.7.1 Non-negative Least Squares with Affinity Constraint

The skinning weights w_{ij} are updated by fixing the skeleton, joint positions, bone transformations and find the optimized w_{ij} subject to the non-negatively and the affine constraint (Eq. (3.1e)). Thus, the building block for the weights update is the constrained linear least square (3.6).

$$W_i^{\mathsf{T}} = \arg\min_x \|Ax - b\|^2 \tag{3.6}$$
$$\text{Subject to: } x \geq 0$$
$$Jx = 1, J = (1, 1, \ldots, 1)$$

The problem (3.6) 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 $Cx = d$ and the inequality constraint could be in a form of $Ex > f$. One widely-used solution to the general linear least squares problem is the Active Set Method (ASM) [20].

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 [20, 11, 60], we make the following two major modifications:

1. The first modification is to initialize a feasible solution only in the first run: $x = 1/n$ and the corresponding active set $\mathcal{W}$. Later, our FASM algorithm uses the value of x from the previous run as the initial value in the next run. Since our block coordinate descent algorithm (Algorithm 3) updates the bone-vertex influences in every iteration and converges to the optimal solution, we assume the value of x and the active set will not be changed significantly in late iterations. Thus, re-using the solution in the previous step for initialization would save unnecessary computational efforts. Actually, a similar idea was successfully used for aircraft control allocation [24].

2. The second modification is to pre-compute the cross products $\mathcal{A} = A^{\mathsf{T}}A$ and $\mathcal{B} = A^{\mathsf{T}}b$, as proposed in [13], since the solution of the unconstrained linear least squares $Ax = b$ is the same as that of $A^{\mathsf{T}}Ax = A^{\mathsf{T}}b$. Thus, we can solve the least squares $p = \arg\min \|A(x + p) - b\|^2$ with equality constraint $Jp = 0$ and active set constraint $p_i = 0, \forall i \in \mathcal{W}$ via $\mathcal{A}$ and $\mathcal{B}$. By doing so, each iteration on active set needs to solve the linear least squares with the size of the unknowns 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 the $\mathcal{A}$ and $\mathcal{B}$ can also be used to calculate the Lagrange multipliers in equation (3.7).

As described in Algorithm 1, in the proposed FASM algorithm, we first initialize a feasible solution $x = 1/n$ and its corresponding active set $\mathcal{W}$ (line 3). In each iteration (line 5 to 25), we first solve the descent direction p for all p_i with i-th constraint active (line 6). Here we enforce the constraint $Jp = 0$ to ensure the moving of x in the direction of p would satisfy the equality constraint $J(x+\alpha p) = 1$. If $x+p \geq 0$, we move the current solution by p, i.e. let $x \leftarrow x+p$ (line 8), otherwise, we move the current solution by $x \leftarrow x + \alpha p$ 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 (3.7) [43, 9, 60] are used to check the optimum of the current solution x. Using these KKT conditions, we calculate the *Lagrange multiplier* λ_i for each constraint $i \notin \mathcal{W}$. 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 $Jx = 1$. Since the equality constraint $Jx = 1$ is always true, there are at most $n-1$ active inequality constraints. Thus, the system of equation (3.7) is never under-determined. In addition, the solution x always satisfies all the constraints, the system of equation (3.7) is always consistent.

$$\begin{bmatrix} J^\mathsf{T} & I \end{bmatrix} \begin{bmatrix} \mu \\ \lambda \end{bmatrix} = A^\mathsf{T}(Ax - b) \tag{3.7}$$

$$\lambda_i = 0, \forall i \notin \mathcal{W}$$

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

Input: $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$

Output: $x^* = \arg\min \|Ax - b\|^2$ s.t. $x \geq 0$ and $Jx = 1$

$\mathcal{A} \leftarrow A^\mathsf{T} A$
$\mathcal{B} \leftarrow A^\mathsf{T} b$
Initialize a feasible solution $x = 1/n$ if necessary
Initialize active (working) set $\mathcal{W} \leftarrow \{i | x_i = 0\}$
loop
    Solve $p = \arg\min \|A(x + p) - b\|^2$ s.t. $Jp = 0$ and $p_i = 0, \forall i \in \mathcal{W}$ via $\mathcal{A}$ and $\mathcal{B}$ by algorithm 2
    if $x + p \geq 0$ **then**
        $x \leftarrow x + p$
        Compute the Lagrange multipliers λ asscociated with $\mathcal{W}$ by equation (3.7)
        if $\lambda_i \geq 0, \forall i$ **then**
            return $x^* \leftarrow x$ is the optimal solution
        else
            $i^* \leftarrow \arg\min_{i \in \mathcal{W}} \lambda_i$
            $\mathcal{W} \leftarrow \mathcal{W} \backslash i$
        end if
    else
        $\alpha \leftarrow -\max_{i \notin \mathcal{W}} x_i / p_i$
        if $\alpha \leq 0$ **then**
            return $x^* \leftarrow x$ is the optimal solution
        else
            $x_i \leftarrow x_i + \alpha p_i, \forall i$
            $\mathcal{W} \leftarrow \{i | x_i = 0\}$
        end if
    end if
end loop

3.7.1.1 Handling Equality Constraints

In the above Algorithm 1, we need to solve a sub-problem of linear least squares with equality constraint $Jp = 0$ (zero-sum) (line 6). 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 [58]. However, this solution would change the cost function and thus may break the convergence of our main algorithm. Instead, we can use an orthogonal projection to preserve the cost function [11].

Suppose we need to solve for x' in the linear least squares with h equality constraints (Equation (3.8)).

$$\min_{x'} \|A'x' - b'\|^2 \text{ s.t. } C'x' = d' \qquad (3.8)$$
$$x' \in \mathbb{R}^k, C' \in \mathbb{R}^{h \times k}$$

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

$$C'^{\mathsf{T}} = QR = [\underbrace{Q_1}_{h} | \underbrace{Q_2}_{k-h}] \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \text{ s.t. } QQ^{\mathsf{T}} = I$$

Let

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

We have

$$d' = C'x' = R^{\mathsf{T}}Q^{\mathsf{T}}x' = R^{\mathsf{T}} \begin{bmatrix} y \\ z \end{bmatrix} = {R_1}^{\mathsf{T}}y$$

and

$$A'x' = A'QQ^{\mathsf{T}}x' = A'[Q_1|Q_2] \begin{bmatrix} y \\ z \end{bmatrix} = A'Q_1y + A'Q_2z$$

Then, the problem (3.8) becomes an unconstrained linear least squares

$$\min_z \|A_2z - (b' - A_1y)\|^2$$

Where: $A_1 = A'Q_1$

$A_2 = A'Q_2$

$R_1^{\mathsf{T}}y = d'$

$$x' = Q \begin{bmatrix} y \\ z \end{bmatrix}$$

The solution of this unconstrained linear least squares is

$$z = (A_2{}^{\mathsf{T}} A_2)^{-1} A_2{}^{\mathsf{T}} (b' - A_1 y) \tag{3.9}$$

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

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

$$\begin{gathered} J^{\{k\}\mathsf{T}} = Q^{\{k\}} R^{\{k\}} = \left[Q_1^{\{k\}} | Q_2^{\{k\}} \right] R^{\{k\}} \\ \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}}_{J^{\{k\}\mathsf{T}}} = \underbrace{\begin{bmatrix} \alpha \\ \alpha \\ \alpha \\ \alpha \\ \vdots \\ \alpha \end{bmatrix}}_{Q_1^{\{k\}}} \underbrace{\begin{bmatrix} \alpha & \alpha & \alpha & \cdots & \alpha \\ \delta & \gamma & \gamma & \cdots & \gamma \\ \gamma & \delta & \gamma & \cdots & \gamma \\ \gamma & \gamma & \delta & \cdots & \gamma \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \gamma & \gamma & \gamma & \cdots & \delta \end{bmatrix}}_{Q_2^{\{k\}} \in \mathbb{R}^{k \times (k-1)}} \underbrace{\begin{bmatrix} \rho \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{R^{\{k\}}} \\ \text{Here: } \alpha = -1/\sqrt{k}, \delta = \gamma + 1, \gamma = \frac{-1}{k + \sqrt{k}}, \rho = -\sqrt{k} \end{gathered} \tag{3.10}$$

In Algorithm 2, we can represent the two matrix products $A_2{}^{\mathsf{T}} A_2$ and $A_2{}^{\mathsf{T}}(b' - A_1 y)$ in (3.9) via the pre-calculated matrices $\mathcal{A} = A^{\mathsf{T}} A$ and $\mathcal{B} = A^{\mathsf{T}} b$. The final

simplified calculations are shown in lines 1 to 7 of Algorithm 2, where $\tilde{\mathcal{A}} = {A_2}^{\mathsf{T}} A_2$ and $\tilde{\mathcal{B}} = {A_2}^{\mathsf{T}}(b' - A_1 y)$.

Algorithm 2 Linear Least-Squares with Zero-Sum Constraint

Input: $\mathcal{A} \in \mathbb{R}^{n \times n}, \mathcal{B} \in \mathbb{R}^n, x \in \mathbb{R}^n, \mathcal{W} \subseteq \{1 \dots n\}$

Output: $p = \arg\min \|A(x+p) - b\|^2$ s.t. $\mathcal{A} = A^{\mathsf{T}} A$,

$\mathcal{B} = A^{\mathsf{T}} b$, $Jp = 0$ and $p_i = 0, \forall i \in \mathcal{W}$

1: $\mathcal{B}' \leftarrow \mathcal{B} - \mathcal{A}x$

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 $Q_2^{\{k\}} \in \mathbb{R}^{k \times (k-1)}$ by the QR decomposition in equation (3.10), where $k = n - |\mathcal{W}|$

5: $\tilde{\mathcal{A}} \leftarrow {Q_2^{\{k\}}}^{\mathsf{T}} \mathcal{A}^{\overline{\mathcal{W}}} Q_2^{\{k\}}$

6: $\tilde{\mathcal{B}} \leftarrow {Q_2^{\{k\}}}^{\mathsf{T}} \mathcal{B}^{\overline{\mathcal{W}}}$

7: $\tilde{p} \leftarrow \tilde{\mathcal{A}}^{-1} \tilde{\mathcal{B}}$

8: $p^{\overline{\mathcal{W}}} \leftarrow Q_2^{\{k\}} \tilde{p}$

9: **return** $p \leftarrow$ Corresponding rows of $p^{\overline{\mathcal{W}}}$, other rows are 0

3.7.2 Rigidness Laplacian Regularizer

Conventional least squares (LS) skinning weights solvers [32, 66] are sensitive to noisy data. When the sparseness constraint is imposed, the LS solver might generate skinning fractures, i.e., visible discontinuities on the surface (examples are shown as the $\omega = 0$ case in Fig. 3.5), as some neighboring vertices are associated with different bones. Thus, we add a weight regularization term to make our algorithm robust.

Instead, I propose a rigidness Laplacian L computed by Eq. (3.11a), where the rigidness weight d_{ik} penalizes the maximum change of the Euclidean distance between vertex i and vertex k with respect to the rest pose (Eq. (3.11b)). Intuitively, if vertex i and vertex k belong to the same nearly-rigid part, the distance between them should not change much, resulting in a large rigidness weight d_{ik} and highly similar skinning weights for vertex i and vertex k (i.e., w_i and w_k).

$$L_{ik} = \begin{cases} 1 & \text{if } k = i \\ -\frac{d_{ik}}{\sum_{h \in \mathcal{N}(i)} d_{ih}} & \text{if } k \in \mathcal{N}(i) \\ 0 & \text{otherwise} \end{cases} \tag{3.11a}$$

Where: $\mathcal{N}(i)$ denotes all the 1-ring neighbors of vertex i

$$d_{ik} = \frac{1}{\sqrt{\frac{1}{F}\sum_{f=1}^{F}\left(\left\|v_i^f - v_k^f\right\|_2 - \left\|u_i - u_k\right\|_2\right)^2}} \tag{3.11b}$$

With the proposed matrix L, we empirically set $\omega = 10^{-3}$ for all the experiments in this paper. This parameter is determined by first resizing all the datasets to tightly fit their rest poses in a unit sphere and then observing the deformation smoothness for different ω values, examples of which are shown in Fig. 3.5.

3.7.3 Iterative Local Solver

Without the smoothness term E_s, skinning weights can be updated per-vertex [32, 45] since the weights of each vertex are independent of those of other vertices. In our

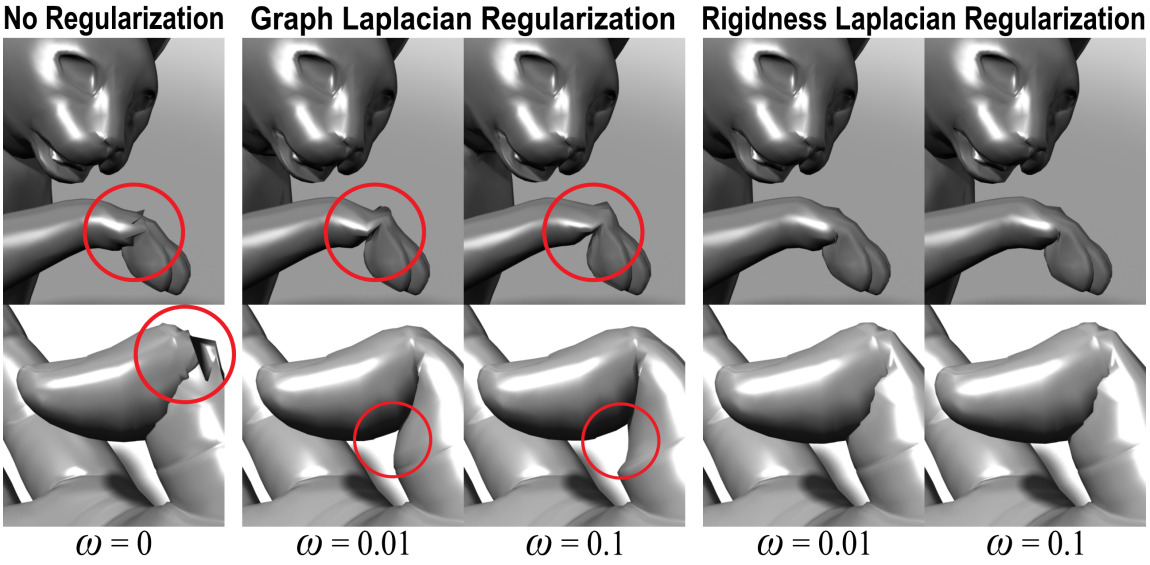

Figure 3.5: The effect of different weight smoothness regularizations. Compared to the graph Laplacian [57, 39], our rigidness Laplacian regularization offers better smoothness while still preserving the global shape. Note that how the graph Laplacian over-smooths the fracture while it starts to change the global shape (indicated by red circles). Our rigidness Laplacian is robust as demonstrated by the similar results with different ω values.

case, the weights of all the vertices are related to each other in the smoothness term E_s (refer to Eq. (3.1c)). Unfortunately, a global weights update of all the vertices is impractical with a very large number of unknowns ($N \times 4$). Instead, we use an iterative local optimization strategy similar to the one used in [44]. Specifically, we iterate all the vertices one by one (with the given order in the input). For each vertex $\hat{i}$, we fix the weights of its one-ring neighbors and minimize the objective function, Eq. (3.1a), with respect to weights $w_{\hat{i}} \in \mathbb{R}^B$. Then, this problem becomes a linear least squares problem with convex and sparseness constraints as follows:

$$\min_{w_{\hat{i}}} \frac{1}{NF} \sum_{f=1}^{F} \left\| \sum_{j=1}^{B} w_{\hat{i}j} [R_j^f | T_j^f] \begin{bmatrix} u_{\hat{i}} \\ 1 \end{bmatrix} - v_{\hat{i}}^f \right\|_2^2 + \omega \sum_{i=1}^{N} L_{\hat{i}i} \left\| w_{\hat{i}} - w_i \right\|_2^2 \tag{3.12}$$

The above least squares problem can be solved in a similar way as [32]. Specifically, we handle the sparseness constraint by greedily selecting 4 weights with smallest residuals when used to individually estimate the objective function, Eq. (3.12). Then, we perform non-negative linear least squares with the affinity constraint on the set of 4 selected weights to compute their values (§3.7.1). Note that its size is relatively small since L is a sparse matrix. As suggested by Landreneau and Schaefer [44], we iterate the local solver a few times proportional to the number of mesh vertices. Since these iterations are mixed with the iterations of the global block coordinate descent, a small number of iterations are typically sufficient to reach the convergence in our experiments, e.g., 0.05 percent of the total number of the mesh vertices.

Note that similar solutions of using *graph Laplacian* for skinning weights regularization have been proposed [57, 39]. However, in their methods, the Laplacian matrix is computed only based on the graph of a single mesh; it is insensitive to the deformation over the whole mesh sequence. In contrast, our rigidness Laplacian regularization takes the deformation of the whole mesh sequence into consideration (Fig. 3.5).

3.8 Bone Transformations Update

3.8.1 Absolute Orientation Problem

The *Absolute Orientation* problem, a.k.a. procrustes problem, is the problem of finding the best rotation and translation to transform one set of points to another set of points. Assumming we want to find the transformation $[R|T]$ to transform the set of N points p_i to another the set of N points q_i, the general strategy [26, 34] is minimizing the sum of square error beween all q_i and all p_i after applying $[R|T]$. This is equivalent to minimizing the objective function (3.13) with respect to the orthogonal contraint $R^{\mathsf{T}}R = I$ of the rotation matrix.

$$\min_{R,T} \sum_{i=1}^{N} \|(Rp_i + T) - q_i\|^2 \tag{3.13}$$

The solution for this minimization problem is proposed by Kabsch [34]. We first need to subtract both points sets to their centroid in order to remove the translation from the objective function. Then, we find the best rotation by *Singular Value Decomposition*.

Specifically, the centroids of points sets are:

$$p_* = \frac{1}{N}\sum_{i=1}^{N} p_i, q_* = \frac{1}{N}\sum_{i=1}^{N} q_i \tag{3.14}$$

Then, we perform translations for all the set of vertices as follows:

$$\overline{p}_i = p_i - p_*, \overline{q}_i = q_i - q_* \tag{3.15}$$

Finally, the optimal transformation is computed by performing Singular Value Decomposition:

$$R = \vartheta\mu^{\mathsf{T}}; \quad T_j^f = q_* - Rp_* \tag{3.16a}$$

$$\text{Where: } \mu\varsigma\vartheta^{\mathsf{T}} = \sum i = 1^N \overline{p}_i\overline{q}_i^{\mathsf{T}} \tag{3.16b}$$

3.8.2 Rigid Transformations with Blending

In this step, we need to fix the bone-vertex influences and minimize the objective function (4.1a) with respect to the bone rotations R and the bone translations T over the set of example poses. Since the bone transformations for every pose are independent, we can solve the minimization for each pose individually. For the pose t, the problem then becomes finding the set of bone transformations to associate the vertices in the rest pose $\{p_i\}$ to the vertices $\{v_i^t\}$ in pose t through minimization of the objective function (Equation (3.17)).

$$\min_{R^t,T^t} E^t = \min_{R^t,T^t} \sum_{i=1}^{N} \left\| v_i^t - \sum_{j=1}^{B} w_{ij}(R_j^t p_i + T_j^t) \right\|^2 \tag{3.17}$$

The above quadratic optimization problem (3.17) can be solved by the Levenberg-Marquardt algorithm [55]. However, this algorithm is inefficient since it requires

many iterations to find the accurate optimum. Many other approaches have been proposed to approximate the result or tackle similar problems. For example, Horn [26] proposed a closed-form solution for the Absolute Orientation problem, which is a particular case of (3.17) with only one bone transformation to be solved. Schaefer et al. [65] slightly change the cost function and solve the new problem by the Moving Least Squares (MLS) framework. The MLS solution, however, is not the exact solution to the original problem (3.17). Recently, Kavan et al. [36] proposed a closed-form solution by transforming (3.17) to the Dual Quaternion space. Unfortunately, due to the non-linear nature of this transformation, it cannot preserve the error metric in the original space. In order to update the bone transformations, we can apply all these methods, however, none of them can guarantee the convergence of our main algorithm, except the high computational cost Levenberg-Marquardt algorithm [55].

To overcome this limitation, we update the bone transformations one by one instead of updating all of them at once (Fig. 3.6). However, our updating can guarantee the non-increasing of the global objective function (4.1a). By doing so, we essentially break the problem with multiple transformations to simpler problems where we need to only deal with a single transformation. Then, the simple problem of finding the optimal transformation of the bone $\hat{j}$ for the example pose t can be solved by minimizing the following objective function:

$$E_{\hat{j}}^{t} = \sum_{i=1}^{N} \left\| v_i^t - \sum_{j=1, j\neq\hat{j}}^{B} w_{ij}(R_j^t p_i + T_j^t) - w_{i\hat{j}}(R_{\hat{j}}^t p_i + T_{\hat{j}}^t) \right\|^2 \tag{3.18}$$

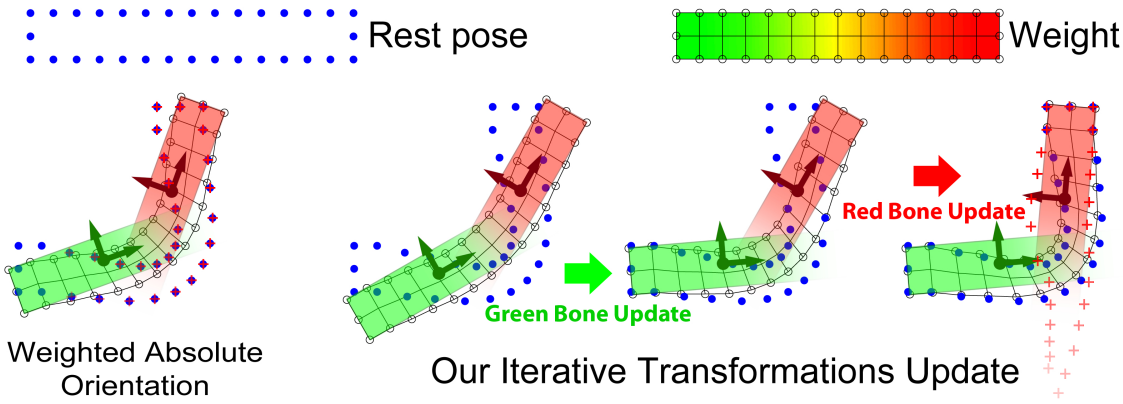

Figure 3.6: Comparison of the Weighted Absolute Orientation solution [26, 34] on the left and our iterative bone transformations update on the right. The blue dots indicate vertices in the example pose. The red plus signs (+) indicate the target positions for the red bone transformation fitting. While the Weighted Absolute Orientation only provides an approximate solution due to the deformation of the target positions in the example pose, our iterative 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).

Let

$$q_i^t = v_i^t - \sum_{j=1, j\neq\hat{j}}^{B} w_{ij}(R_j^t p_i + T_j^t) \tag{3.19}$$

The problem of finding the optimal transformation then becomes:

$$\begin{aligned}\{R_{\hat{j}}^t, T_{\hat{j}}^t\}^* &= \arg\min_{R_{\hat{j}}^t, T_{\hat{j}}^t} E_{\hat{j}}^t \\ &= \arg\min_{R_{\hat{j}}^t, T_{\hat{j}}^t} \sum_{i=1}^{N} \left\| q_i^t - w_{i\hat{j}}(R_{\hat{j}}^t p_i + T_{\hat{j}}^t) \right\|^2 \\ \text{Subject to: } & R_{\hat{j}}^{t\mathsf{T}} R_{\hat{j}}^t = I, \det R_{\hat{j}}^t = 1\end{aligned} \tag{3.20}$$

The problem (3.20) is similar to the original Absolute Orientation problem [26], in which we need to find the transformation to relate the two sets of vertices. The only difference is the scope of the weight term. Inspired by the work of [26], we can first remove the translation $T_{\hat{j}}^t$ and then solve for the optimum rotation. This is done by translating all the vertices to bring the center of rotation to the origin for each set of vertices. Let

$$p_* = \frac{\sum_{i=1}^{N} w_{i\hat{j}}^2 p_i}{\sum_{i=1}^{N} w_{i\hat{j}}^2}, q_*^t = \frac{\sum_{i=1}^{N} w_{i\hat{j}} q_i^t}{\sum_{i=1}^{N} w_{i\hat{j}}^2} \tag{3.21}$$

Then, we perform translations for all the set of vertices as follows.

$$\overline{p}_i = p_i - p_*, \overline{q}_i^t = q_i^t - w_{i\hat{j}} q_*^t \tag{3.22}$$

The cost function (3.20) yields

$$E_{\hat{j}}^t = \sum_{i=1}^{N} \left\| \overline{q}_i^t + w_{i\hat{j}} q_*^t - w_{i\hat{j}} [R_{\hat{j}}^t (\overline{p}_i + p_*) + T_{\hat{j}}^t] \right\|^2$$
$$= \sum_{i=1}^{N} \left\| \overline{q}_i^t - w_{i\hat{j}} R_{\hat{j}}^t \overline{p}_i - w_{i\hat{j}} [T_{\hat{j}}^t - q_*^t + R_{\hat{j}}^t p_*] \right\|^2 \tag{3.23}$$

Let $\tau = T_{\hat{j}}^t - q_*^t + R_{\hat{j}}^t p_*$, (3.23) yields

$$E_{\hat{j}}^t = \sum_{i=1}^{N} \left\| \overline{q}_i^t - w_{i\hat{j}} R_{\hat{j}}^t \overline{p}_i - w_{i\hat{j}} \tau \right\|^2 = \sum_{i=1}^{N} \left\| \overline{q}_i^t - w_{i\hat{j}} R_{\hat{j}}^t \overline{p}_i \right\|^2$$
$$- 2\tau \sum_{i=1}^{N} (w_{i\hat{j}} \overline{q}_i^t - {w_{i\hat{j}}}^2 R_{\hat{j}}^t \overline{p}_i) + \sum_{i=1}^{N} \left\| w_{i\hat{j}} \tau \right\|^2 \tag{3.24}$$

From equations (3.21) and (3.22), we have $\sum_{i=1}^{N} w_{i\hat{j}} \overline{q}_i^t = 0$ and $\sum_{i=1}^{N} {w_{i\hat{j}}}^2 R_{\hat{j}}^t \overline{p}_i = 0$. Simplifying equation (3.24) yields

$$E_{\hat{j}}^t \to \min \Leftrightarrow \begin{cases} \sum_{i=1}^{N} \left\| \overline{q}_i^t - w_{i\hat{j}} R_{\hat{j}}^t \overline{p}_i \right\|^2 \to \min & (3.25) \\ \sum_{i=1}^{N} \left\| w_{i\hat{j}} \tau \right\|^2 \to \min \Leftrightarrow \tau = 0 & (3.26) \end{cases}$$

From equation (3.26), we can calculate the optimum translation $T_{\hat{j}}^t$ as follows if the optimum rotation $R_{\hat{j}}^t$ is known.

$$T_{\hat{j}}^t = q_*^t - R_{\hat{j}}^t p_* \tag{3.27}$$

After the translation $T_{\hat{j}}^t$ is removed, we find the optimum rotation $R_{\hat{j}}^t$ by minimizing (3.25). First, we construct two matrices $P = [w_{1\hat{j}}\overline{p}_1^t \dots w_{N\hat{j}}\overline{p}_N^t]$ and $Q = [\overline{q}_1^t \dots \overline{q}_N^t]$ ($P, Q \in \mathbb{R}^{3\times N}$). The objective function in (3.25) becomes

$$\begin{aligned}\sum_{i=1}^{N} \left\| \overline{q}_i^t - w_{i\hat{j}} R_{\hat{j}}^t \overline{p}_i \right\|^2 &= \sum_{i=1}^{N} \left\| \overline{q}_i^t - R_{\hat{j}}^t (w_{i\hat{j}} \overline{p}_i) \right\|^2 \\ &= \left\| Q - R_{\hat{j}}^t P \right\|_F = tr((Q - R_{\hat{j}}^t P)^{\mathsf{T}} (Q - R_{\hat{j}}^t P)) \\ &= tr(Q^{\mathsf{T}} Q) + tr(P^{\mathsf{T}} {R_{\hat{j}}^t}^{\mathsf{T}} R_{\hat{j}}^t P) - 2tr(Q^{\mathsf{T}} R_{\hat{j}}^t P)\end{aligned}$$

Where $\|.\|_F$ denotes the Frobenius norm and $tr(.)$ denotes the matrix trace. Since P and Q are constant matrices and ${R_{\hat{j}}^t}^{\mathsf{T}} R_{\hat{j}}^t = I$, the problem then becomes maximizing

$$\zeta = tr(Q^{\mathsf{T}} R_{\hat{j}}^t P) = tr(P Q^{\mathsf{T}} R_{\hat{j}}^t) \rightarrow \max \tag{3.28}$$

Then, SVD is performed on PQ^{T} as follows:

$$PQ^{\mathsf{T}} = \sum_{i=1}^{N} w_{i\hat{j}} \overline{p}_i {\overline{q}_i}^{\mathsf{T}} = \mu \Sigma \vartheta^{\mathsf{T}} \tag{3.29}$$

We can rewrite the objective function ζ in equation (3.28) as follows:

$$\zeta = tr(\mu \Sigma \vartheta^{\mathsf{T}} R_{\hat{j}}^t) = tr(\Sigma \vartheta^{\mathsf{T}} R_{\hat{j}}^t \mu)$$

Since ϑ, $R_{\hat{j}}^t$, and μ are orthogonal matrices, $\vartheta^{\mathsf{T}} R_{\hat{j}}^t \mu$ is an orthogonal matrix as well. In addition, since Σ is a diagonal matrix, we have $tr(\Sigma \vartheta^{\mathsf{T}} R_{\hat{j}}^t \mu) \leq tr(\Sigma)$. Thus,

$$\zeta \rightarrow \max \Leftrightarrow \vartheta^{\mathsf{T}} R_{\hat{j}}^t \mu = I \Leftrightarrow R_{\hat{j}}^t = \vartheta \mu^{\mathsf{T}} \tag{3.30}$$

3.8.3 Adding Joint Constraints

To enforce the joint constraints while updating the rigid bone transformations, I employ the solution to the Absolute Orientation problem with blending to relate two sets of points [45] 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. 3.7.

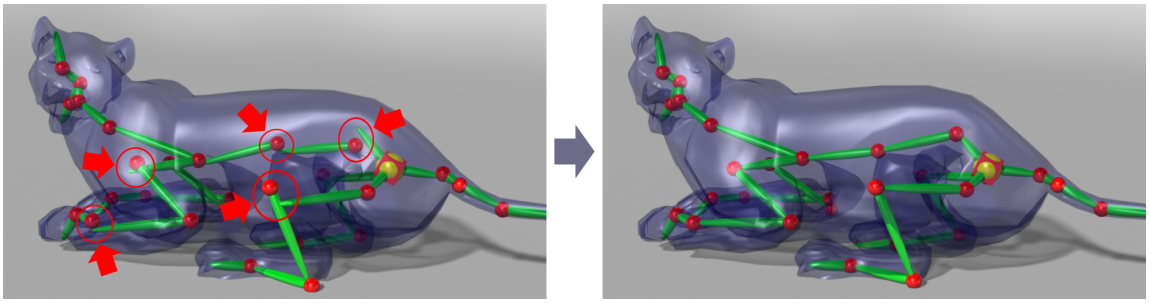

Figure 3.7: (Left) Without the joint constraints, the bone transformations (bones) generated by [45] do not always rotate around the joints. (Right) With the soft joint constraints, bones rotate more strictly around the joints. Since the bone transformations are alternatively updated with the joint locations, our rigging model can converge to a local optimum with a good approximation of the input.

At this bone transformations update step, we need to minimize the objective function (Eq. (3.1a)) with respect to the bone transformations $[R_j^f|T_j^f]$. To constrain the rotation matrix R_j^f to be orthogonal, we employ the optimization strategy for the rigid transformations with blending, where bones are updated one by one while keeping the rest of the bones fixed.

The transformation of bone $\hat{j}$ at frame f is updated by minimizing the following objective function, Eq. (3.31a), which contains two parts. The first part corresponds to the data fitting term (Eq. (3.1b)) whose optimal solution brings the rest pose u_i to the residual q_i^f (Eq. (3.31b)). The second part corresponds to the joint constraints in Eq. (3.1d), which enforces $[R_{\hat{j}}^f|T_{\hat{j}}^f]$ to bring every joint $C_{\hat{j}k}$ of bone $\hat{j}$ to its expected position $\Psi_k^f(C_{\hat{j}k})$ after bone k transformation. Note that the weight smoothness term E_S in Eq. (3.1c) can be dropped since $[R_j^f|T_j^f]$ is not involved in E_S.

$$\begin{aligned}\min E_{\hat{j}}^f =& \frac{1}{N}\sum_{i=1}^{N}\left\| w_{i\hat{j}}[R_{\hat{j}}^f|T_{\hat{j}}^f]\begin{bmatrix}u_i\\1\end{bmatrix} - q_i^f\right\|_2^2 + \\ & \lambda\sum_{(\hat{j},k)\in\mathbb{S}}\left\|[R_{\hat{j}}^f|T_{\hat{j}}^f]\begin{bmatrix}C_{\hat{j}k}\\1\end{bmatrix} - \Psi_k^f(C_{\hat{j}k})\right\|_2^2 \end{aligned} \tag{3.31a}$$

$$\text{Where: } q_i^f = v_i^f - \sum_{j=1,j\neq\hat{j}}^{B} w_{ij}[R_j^f|T_j^f]\begin{bmatrix}u_i\\1\end{bmatrix} \tag{3.31b}$$

$$\Psi_k^f(C_{\hat{j}k}) = [R_k^f|T_k^f]\begin{bmatrix}C_{\hat{j}k}\\1\end{bmatrix} \tag{3.31c}$$

The optimal solution to minimize the objective function Eq. (3.31a) is the combination of the solution of rigid transformations with blending and the solution of the Weighted Absolute Orientation problem [34]. We first need to compute the center of rotation p_* for the rest pose (Eq. (3.32a)) and the center of rotation q_*^f for frame f (Eq. (3.32b)). Note that, in Eqs. (3.32a) and (3.32b), $\left|(\hat{j},k)\in\mathbb{S}\right|$ denotes the number of the edges that are connected with $\hat{j}$ in $\mathbb{S}$.

$$p_* = \frac{\frac{1}{N}\sum_{i=1}^{N} w_{i\hat{j}}^2 u_i + \lambda \sum_{(\hat{j},k)\in\mathbb{S}} C_{\hat{j}k}}{\frac{1}{N}\sum_{i=1}^{N} w_{i\hat{j}}^2 + \lambda \left|(\hat{j},k)\in\mathbb{S}\right|} \tag{3.32a}$$

$$q_*^f = \frac{\frac{1}{N}\sum_{i=1}^{N} w_{i\hat{j}} q_i^f + \lambda \sum_{(\hat{j},k)\in\mathbb{S}} \Psi_k^f(C_{\hat{j}k})}{\frac{1}{N}\sum_{i=1}^{N} w_{i\hat{j}}^2 + \lambda \left|(\hat{j},k)\in\mathbb{S}\right|} \tag{3.32b}$$

Then, we subtract the center of rotation from each vertex and each joint as follows:

$$\overline{p}_i = u_i - p_*; \qquad \overline{C}_{\hat{j}k} = C_{\hat{j}k} - p_* \tag{3.33a}$$

$$\overline{q}_i^f = q_i^f - w_{i\hat{j}} q_*^f; \qquad \overline{\Psi}_k^f(C_{\hat{j}k}) = \Psi_k^f(C_{\hat{j}k}) - q_*^f \tag{3.33b}$$

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

$$P = \left[\frac{w_{1\hat{j}}}{N}\overline{p}_1 \ldots \frac{w_{N\hat{j}}}{N}\overline{p}_N \middle| \forall_{(\hat{j},k)\in\mathbb{S}} \lambda\overline{C}_{\hat{j}k}\right] \tag{3.34a}$$

$$Q = \left[\frac{1}{N}\overline{q}_1^f \ldots \frac{1}{N}\overline{q}_N^f \middle| \forall_{(\hat{j},k)\in\mathbb{S}} \lambda\overline{\Psi}_k^f(C_{\hat{j}k})\right] \tag{3.34b}$$

Finally, the optimal transformation of bone $\hat{j}$ at frame f, Eq. (3.35a), is computed by performing Singular Value Decomposition on PQ^{T}.

$$R_{\hat{j}}^{f} = \vartheta\mu^{\mathsf{T}}; \quad T_{\hat{j}}^{f} = q_{*}^{f} - R_{\hat{j}}^{f} p_{*} \tag{3.35a}$$

$$\text{Where: } \mu\varsigma\vartheta^{\mathsf{T}} = PQ^{\mathsf{T}} \tag{3.35b}$$

3.9 Skeleton Pruning

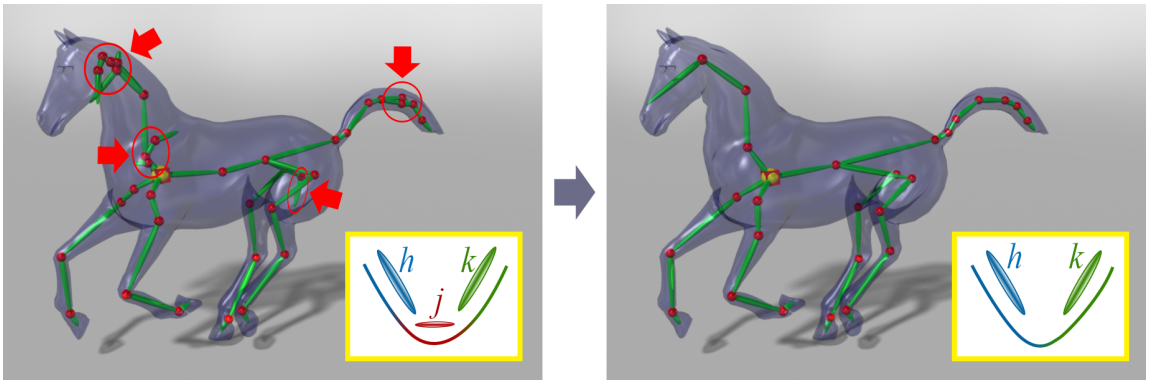

Figure 3.8: The redundant bones in the left panel are pruned to achieve the neat skeleton in the right panel. As illustrated in the two yellow boxes, the redundant bone j is identified by utilizing the weight regularization term to force its weights degenerate.

Since the initialization step (§3.4) 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 motion-driven clustering 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 [45] would still retain the redundant bones in order to minimize the LBS deformation error.

Specifically, instead of conducting a brute-force search for the redundant bones, we utilize the weight regularization term to force their weights degenerate. We illustrate an example of this strategy in two yellow boxes in Fig. 3.8. 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, our minimized objective function (Eq. (3.1a)) 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. In our experiments, we found that the weights of the redundant bones converge to zero quite slowly. For this reason, we remove the redundant bone j if the sum of its squared weights is smaller than 1% of the largest sum of the squared weights among all the bones. Mathematically, we remove the bone j if its weights satisfy the condition described in the following Eq. (3.36).

$$\text{Remove bone } j \text{ if } \sum_{i=1}^{N} {w_{ij}}^2 < 10^{-2} M \tag{3.36}$$

$$\text{where: } M = \max_k \left\{ \sum_{i=1}^{N} {w_{ik}}^2 \right\}$$

3.10 Results and Comparisons

3.10.1 With Skeleton

Datasets. I evaluated my approach on 9 test datasets (Table 3.1) obtained from various publicly available sources: the **cat-poses**, **horse-poses**, **lion-poses**, and **horse-gallop** were obtained from [71]; the **hand** was obtained from [73]; the **dance** and **cow** were obtained from [12]; the **scape** was obtained from [5]; and the **samba** was obtained from [74].

To test the robustness of my approach, I only used a *single* set of parameters for all the experiments on all the test datasets in this paper (see previous sections for parameter selection discussion).

Fig. 3.9 shows the skeletons extracted from 3 test datasets by my approach as well as their corresponding cluster initialization results. Despite generating over-estimated clusters at the initialization step, my approach successfully pruned redundant bones, especially for the **dance** and **hand** models. Fig. 3.10 demonstrates the robustness and effectiveness of my skeleton pruning for handling different numbers of clusters from the initialization step. My 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.

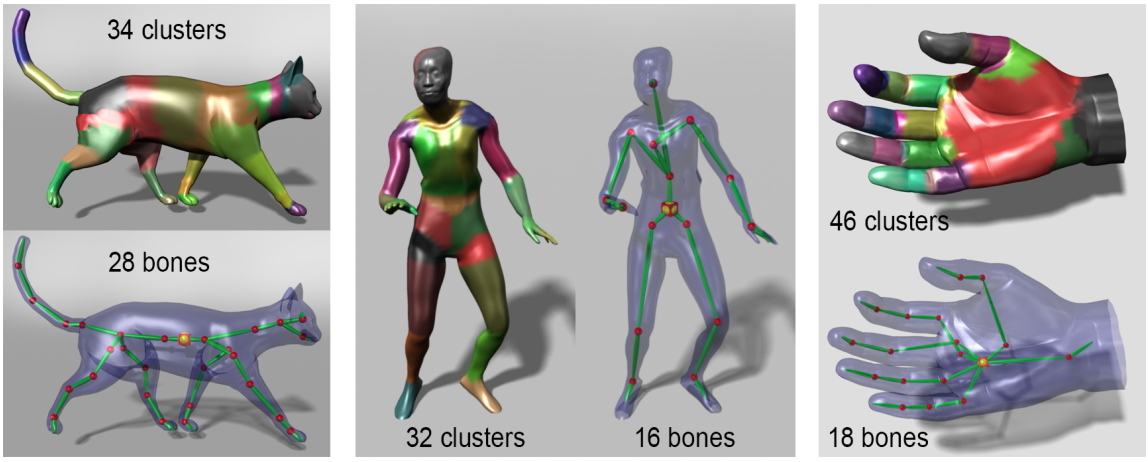

Figure 3.9: The skeletons extracted from 3 datasets by my approach (from left to right): **cat-poses**, **dance**, and **hand**. The clustering results obtained at the initialization step are also illustrated via a color-coded scheme. Using the same set of parameters, my approach can robustly determine the optimal number of bones in the skeletons thanks to the skeleton pruning.

In Fig. 3.11, I compare my approach with three state of the art approaches [66, 14, 25]. For a fair comparison, I use the number of bones generated by my approach as one of the input parameters to the three comparing methods. Other parameters in the three comparing approaches are set as suggested by their original authors. While my approach can generate sound results for all the test datasets, in general the other three approaches suffer from the following two issues:

1. *Redundant bones.* The **lion-poses**, **horse-gallop**, and **scape** generated by

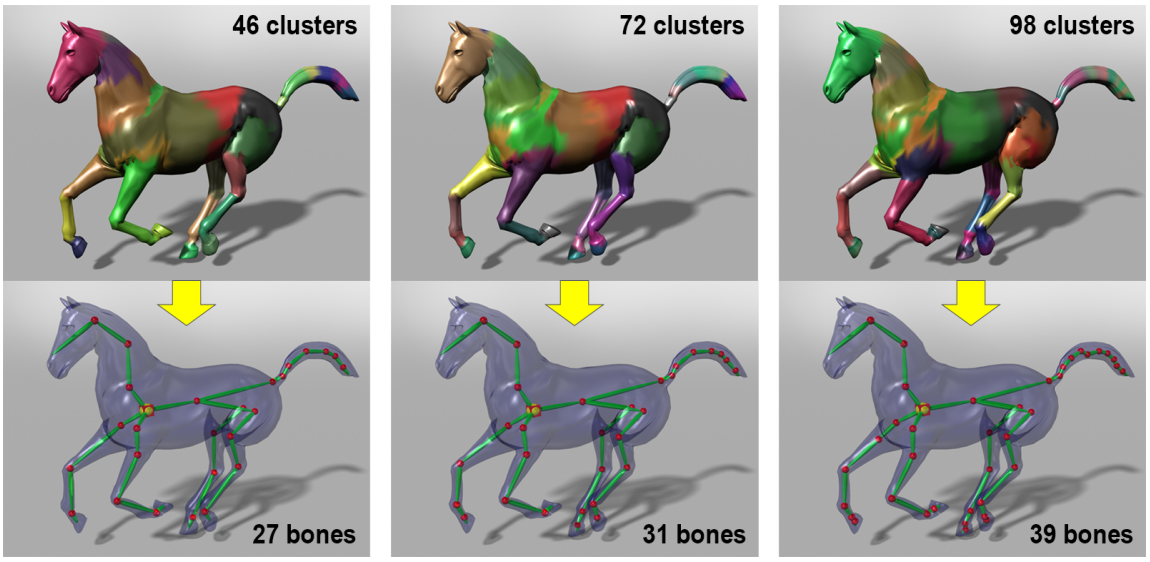

Figure 3.10: The extracted **horse-gallop** skeletons with different cluster initializations. Despite very different numbers of initialized clusters, my approach can output similar final skeletons with consistent structure on the trunk and legs; minor differences on the tail and feet are due to the high deformations on these regions.

both [66] and [25] have many redundant bones, since the clustering algorithms in their approaches result in an over-estimated number of bones in some highly-deformable regions.

2. *Inaccurate joint locations.* The joints at the back legs of the **lion-poses** are inaccurately estimated by both [14] and [25], and the joints at the legs of the **scape** are inaccurately estimated by [66]. I suspect this issue is due to the inaccurate estimation of bone transformations in some highly deformable parts.

In Table 3.1, I show quantitative comparisons among all the four methods (my

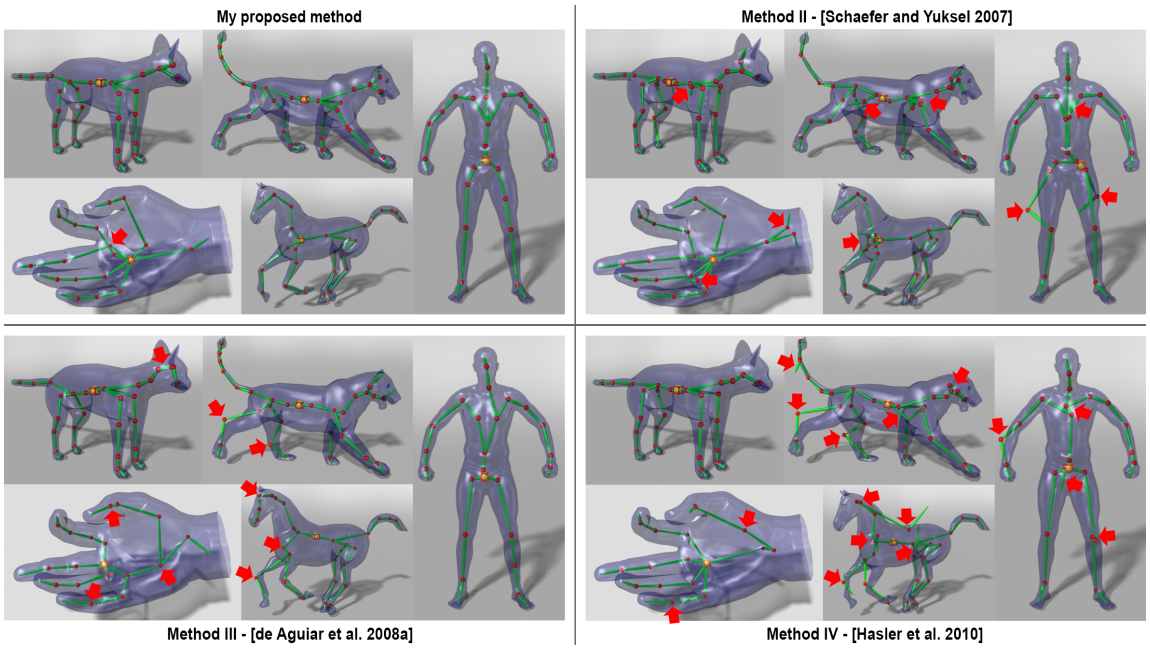

Figure 3.11: Comparisons between my proposed method and three state of the art approaches. The five test datasets shown in this figure are (in the clockwise direction starting from the top-left corner): **cat-poses**, **lion-poses**, **scape**, **horse-gallop**, and **hand**. For a fair comparison, I set the same number of bones for all the four methods. Only my proposed method can generate sound outputs for all the 5 datasets. The issues in the results are indicated by red arrows.

method, *method II* - [66], *method III* - [14], and *method IV* - [25]). In term of the approximation power (measured by lower RMSE), my method soundly outperforms all the other three methods thanks to its iterative rigging. However, my method is slower than both the method II and the method III, since it needs many iterations. Fig. 3.12 shows examples of visual distortion on the reconstructed **hand** poses with respect to different RMSE values.

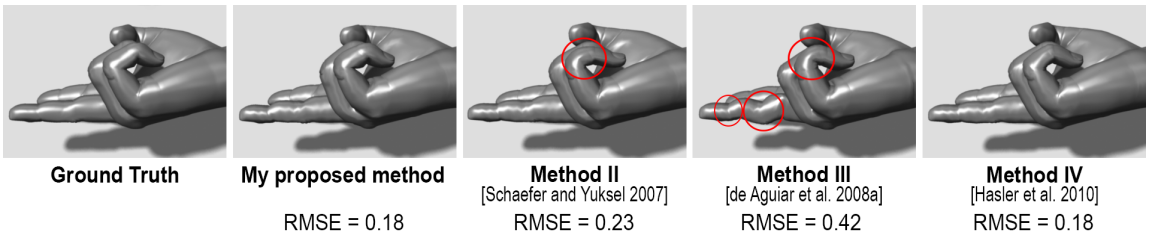

Figure 3.12: Examples of visual distortion on the reconstructed **hand** poses with respect to different RMSE values (pay attention to the red circled areas). Fig. 3.11 shows the corresponding skeletons.

3.10.2 Without Skeleton

The skeleton-based LBS is a good choice if the model is an articulated object. Otherwise, we can relax the skeletal constraint to get a more flexible LBS, which can approximate highly deformable input models. I generate the LBS model without skeleton by only keeping the skinning weights update (§3.7) and bone transformations update without skeleton (§3.8.2).

The test datasets. I used publicly available triangle mesh datasets, including 12 datasets from the "Deformation Transfer" [71], 4 datasets from the "Wavelet Compression" [21], and 1 dataset (i.e., the chicken character dataset) from the "Skinning Mesh Animations" [32]. The original name given by its authors is used to refer to each dataset in this writing. Based on properties of these datasets, I roughly divide the datasets into two categories: *articulated* (near-rigid) models and *elastic* (highly-deformable) models. Details of the used 17 datasets are described in Table 4.1.

Error metric. To quantitatively evaluate my model, I choose the error metric

proposed in [37] over the one in [32], because the former is less sensitive to global motions of the models and thus more robust than the latter, as reported in [37]. Specifically, I first resize all the datasets so that the rest poses are tightly enclosed by a unit sphere. Then, the error metric is calculated from the value E_D of the objective function in Eq. (4.1a) by:

$$E_{RMS} = 1000\sqrt{\frac{E_D}{3.N.F}}$$

Configurations. For each dataset obtained from the "Deformation Transfer" [71], the rest pose u_i is set to be the provided reference pose. For each of the other datasets, the rest pose u_i is chosen to be the first example pose. The number of bones, B, is manually determined for experiments. I denote this number B as a subscript of the dataset name. In my implementation, I stop the algorithm if within one iteration the the objective function E_D is not improved by 1%.

Figure 3.13 shows my results on four articulated models and two elastic models. I use the skin colors to illustrate the bone-vertex influence map. At the rest pose, I also draw a coordinate system for each bone to illustrate its bone transformation, and the origin of the coordinate system is put at the vertex that is most influenced by the bone. In each example pose, the coordinate system for each bone is deformed in accordance with its bone transformation.

Comparisons. I compared our LBS model with two state-of-the-art skinning from examples approaches: the *Skinning Mesh Animations* with rigid bones (SMA) [32] and the *Learning Skeletons for Shape and Pose* [25]. Both competitive methods impose, orthogonal constraints on the bone rotation matrices (rigid bones), convex

constraints and sparseness constraints on the bone-vertex weight map. To ensure a fair comparison, all the methods ran on a single thread, without GPU-based acceleration. I also used all the parameter settings published by the original authors in our implementations. It is noteworthy that the work of [37] was not included in this comparison because it can only handle the case of non-rigid bones, while the focus of this comparison experiment is skinning decomposition with rigid bones.

The SMA was implemented in C++. I used the open source library *Mean Shift Clustering* with *Locality Sensitive Hashing* (LSH) [19]. The two parameters for LSH, i.e., K and L, were set to 70 and 200, respectively. The bandwidth h was set to $h = 9|t|\epsilon$, where $|t|$ is the number of example poses and $\epsilon = 0.05$. To estimate the bone-vertex weights, I used the *Non-Negative Least Squares*.

The LSSP was implemented in MATLAB. I used the *Self-Tuning Spectral Clustering* [78] for the initialization step with automatic local scaling on distance matrix. On the weight matrix optimization step, I employed the *L1/L2+* minimizer [79], where the parameter ρ was set to 0.1 and the A matrix was normalized. At the factorization step, I performed 10 iterations.

Figure 3.14 shows the comparison results on three models generated by my proposed method, 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.

I also evaluated the accuracy and performance of three skinning methods in a quantitative way including the error metric, the number of required bones, execution time, etc. (shown in Table 3.3). In Table 3.3, the number of bones is denoted by the subscript of the dataset name. The error E_{RMS} as well as the error after rank-5 EigenSkin corrections [42] are reported where the numbers in the parentheses are the EigenSkin correction errors. 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 avaiable for many specific numbers of bones. The results of LSSP are also not avaiable for models with more than 10K vertices due to the large memory requirement in the Self-Tuning Spectral Clustering algorithm at the initialization step.

From Table 3.3, We can see that my proposed method clearly outperforms SMA and LSSP. Especially on the elastic models (i.e., camel-collapse$_{11}$, face-poses, horse-collapse$_3$, pcow$_{24}$, and pkanga$_{39}$), my proposed 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, and thus, it only works well if the model that can be divided into nearly rigid parts. Our results also outperform LSSP results for two limitations of the bone-vertex weight map optimization: (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 LSSP supports for the sparseness 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, LSSP performs slowest due to the MATLAB implementation and the comlexity of its clustering and optimization algorithm. In general, my proposed method performs slightly slower than SMA. This is one limitation of the current SSDR model. However, my proposed method can be efficiently accelerated via parallel implementations such as exploiting multi-core CPUs or GPUs. This can be done by parallelizing certain high computational cost operations such as calculating vertex transformations or matrix multiplications. Fortunatelly, most of those calculations are matrix operations, which are relatively easy to be parallelized. By contrast, parallelizing SMA and LSSP will be more challenging since it requires to parallelize spectral clusterings and non-linear optimizers.

3.11 Discussion

Performance. My method can only prune a small number of bones at one time, and thus its iterative rigging may need to be repeated many times if the number of initialized bones is significantly over-estimated. We found a positive correlation between the running time of my method and the number of bones pruned. Therefore, if the number of initialized bones is much larger than the number of bones in the final skeleton, my method takes a significant amount of time to prune redundant bones. Fig. 3.15 visualizes an example of its detailed computation breakdown, which shows that a more proper initialization of bones would significantly shorten the computational time of my method.

Due to the performance reason, my current method is only suitable for offline applications. A potential solution to speed up the performance is to incorporate skeleton templates or certain user interactions into my approach, which is a part of my future work.

Data dependency. Due to its data-driven (i.e., example-based) nature, rigging results by my approach largely depend on the quality of input example poses. I found that the limited motion range and noise of input example poses could affect the outcome of my approach at some cases. For example, the asymmetric skeletons of the **dance** model (in Fig. 3.1 and Fig. 3.9) and the **samba** model (in Fig. 3.1 and Fig. 3.16) are due to noise, asymmetry, and the limited motion ranges of some joints in the example poses. The incorrect joint of the **hand** model (in Fig. 3.11) is due to the very similar motions of the pinkie and index fingers in the example poses.

Approximation power. Results in §3.10.1 demonstrate that skeleton-based LBS is robust and accurate to handle articulated, nearly-rigid models. I also tested skeleton-based LBS with several highly deformable models although this is not one of the targeted applications for any skeleton extraction or rigging methods. The results in Fig. 3.16 and Fig. 3.17 show that, skeleton-based LBS can generate skeleton-based LBS models with better overall approximations than the three previous methods [66, 14, 25], since my iterative rigging algorithm can effectively optimize the objective function in Eq. (3.1a). Notice that skeleton-based LBS can even approximate the **cow** deformation reasonably well (Fig. 3.17), although a more suitable skinning model for this dataset is still the LBS model without skeletal structure §3.10.2. Moreover, due to the limited approximation power of the LBS model, my current

method cannot capture certain complicated non-linear deformations (one example is shown in Fig. 3.18).

Dataset	N	F	B	Proposed method		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 3.1: Quantitative comparisons among all the four methods. The reported RMSE is normalized by the bounding volume diagonal [66, 25]. Specifically, $RMSE = 100 \times \sqrt{E_D}/d$, where E_D is the data fitting error in Eq. (3.1b), 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	N	F	**Category**
camel-collapse	21887	53	Elastic
camel-gallop	21887	48	Articulated
camel-poses	21887	10	Articulated
cat-poses	7207	9	Articulated
chickenCrossing	3030	400	Articulated
elephant-gallop	42321	48	Articulated
elephant-poses	42321	10	Articulated
face-poses	29299	9	Elastic
flamingo-poses	26907	10	Articulated

Name	N	F	**Category**
horse-collapse	8431	53	Elastic
horse-gallop	8431	48	Articulated
horse-poses	8431	10	Articulated
lion-poses	5000	9	Articulated
pcow	2904	204	Elastic
pdance	7061	201	Articulated
pjump	15830	222	Articulated
pkanga	4002	65	Elastic

Table 3.2: The test/evaluation datasets used. N denotes the number of vertices and F denotes the number of example poses.

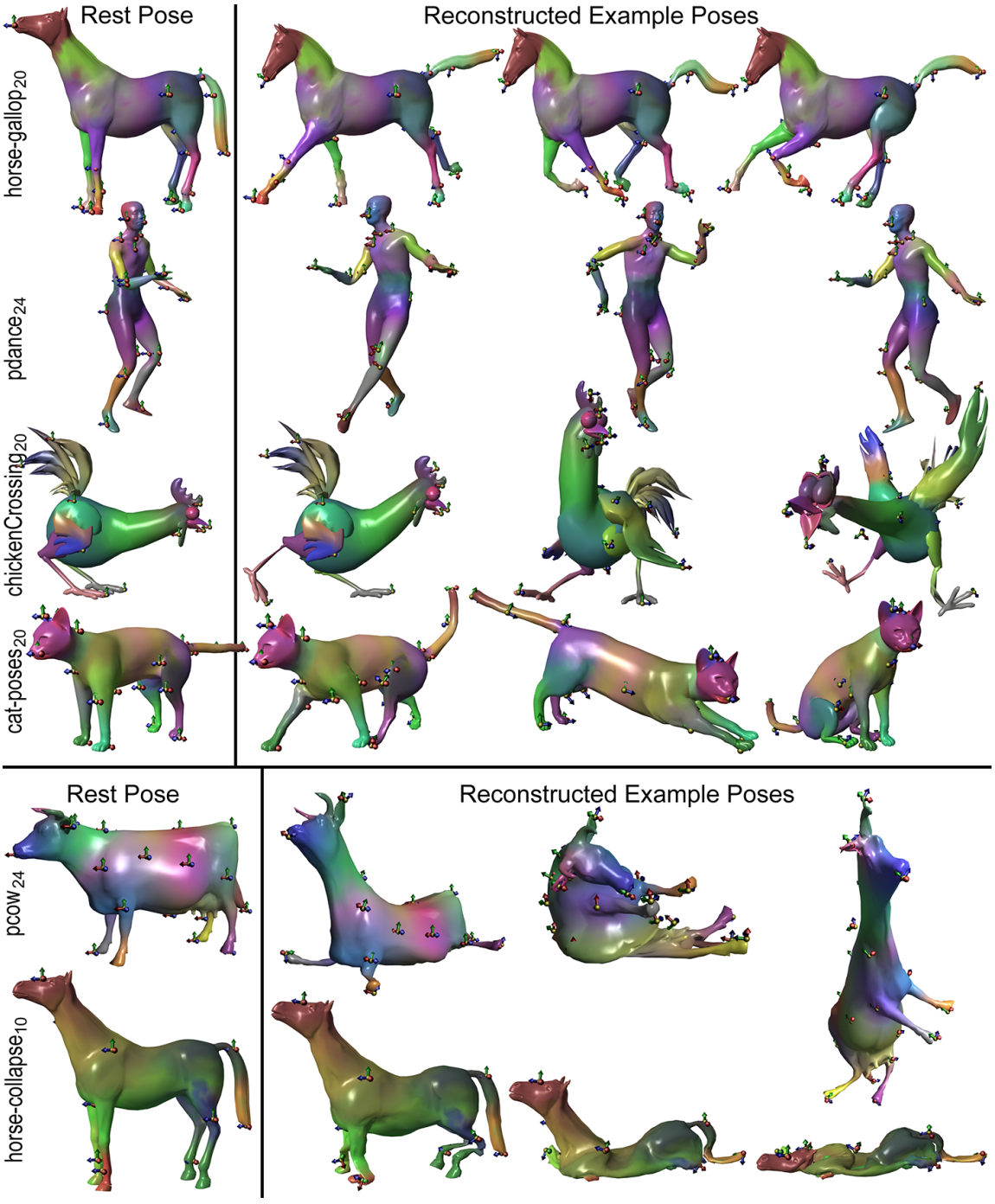

Figure 3.13: Results of my proposed LBS skinning without skeleton. My proposed model works well with both the articulated models (the top four models) and the elastic models (the bottom two models).

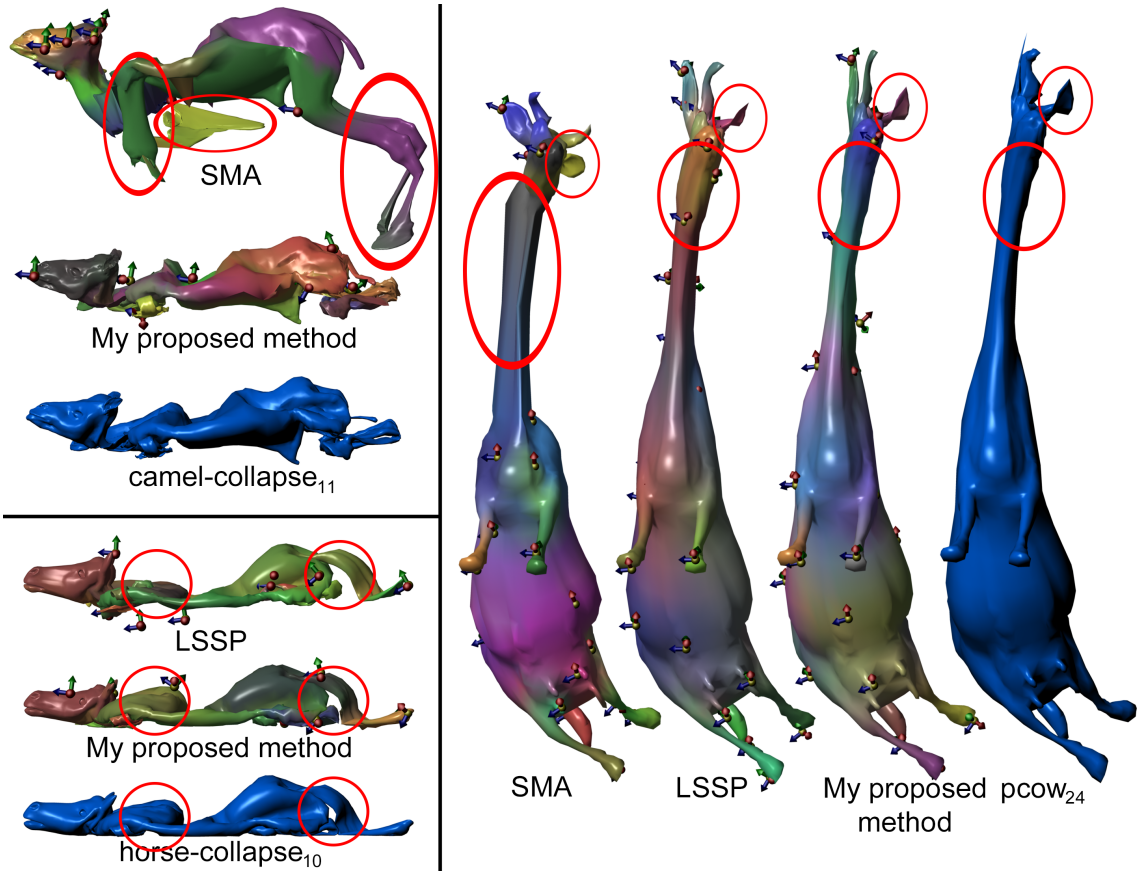

Figure 3.14: Comparisons of the skinning decomposition results among my proposed method, SMA [32], and LSSP [25]. LSSP is unable to run on the camel-collapse due to the large size of this dataset and SMA is unable to configure with 10 bones for the horse-collapse dataset. The example poses are rendered in blue. Significant distortion areas are indicated in red circles.

Dataset$_{[\text{No. of bones}]}$	Approximation error E_{RMS}			Execution time (minutes)		
	SMA	LSSP	Proposed	SMA	LSSP	Proposed
camel-collapse$_{11}$	125.3 (4)	-	5.4(1.7)	13.8	-	7.4
camel-collapse$_{20}$	-	-	4(1.4)	-	-	15.1
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$_{15}$	-	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$_{20}$	-	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$_{20}$	-	-	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$_{27}$	-	-	7(3.6)	-	-	7
face-poses$_{36}$	37.6 (8.5)	-	-	3.6	-	-
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$_{20}$	-	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$_{20}$	-	20.6(7.8)	3.8(1.8)	-	461.8	1.4
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$_{20}$	-	-	6.7(4.7)	-	-	42.7
pjump$_{40}$	15.3 (6.7)	-	4.5(3.4)	30.5	-	104.1
pkanga$_{20}$	-	32(8.5)	8.9(4.5)	-	308.5	3.2
pkanga$_{39}$	134.1 (15.4)	22.4(7)	6.7(4.2)	1.6	360.7	7.6

Table 3.3: Rigid bone skinning decomposition results of Skinning Mesh Animation with rigid bones (SMA) [32], Learning Skeletons for Shape and Pose (LSSP) [25], and my proposed skinning without skeleton.The result after rank-5 EigenSkin corrections [42] is also reported in the parentheses.

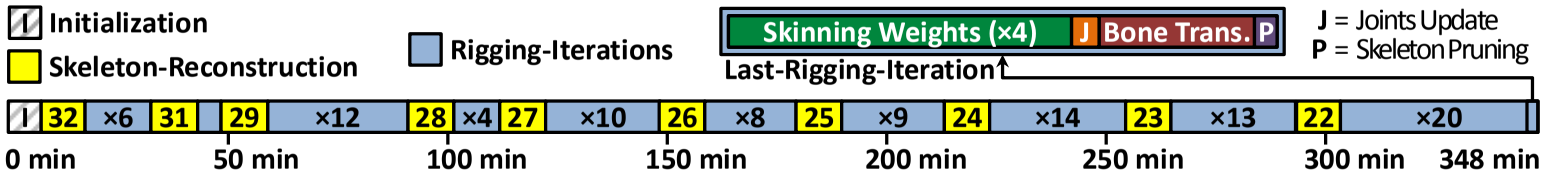

Figure 3.15: Detailed computation breakdown of my method on the **samba** model. The number inside a Skeleton-Reconstruction block denotes the number of bones at the current step. The number inside a Rigging-Iterations block denotes the number of iterations executed at the iterative rigging step. Each rigging iteration includes skinning weights update, joint positions update, bone transformations update, and skeleton pruning. The computational times of all the rigging iterations are approximately the same; one detailed breakdown is illustrated in the Last-Rigging-Iteration block.

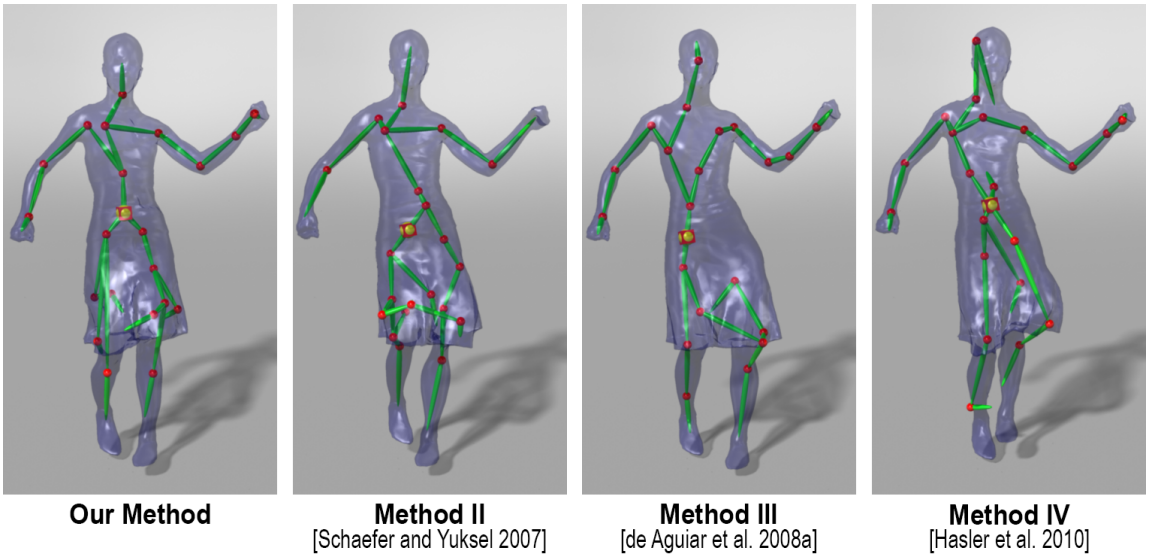

Figure 3.16: Despite the high deformation on the skirt part of the **samba** model, my approach is still able to generate a reasonable skeletal structure where the skirt is rigged by some bones originated from the hip. Meanwhile, the three previous approaches cannot extract similar skeleton patterns.

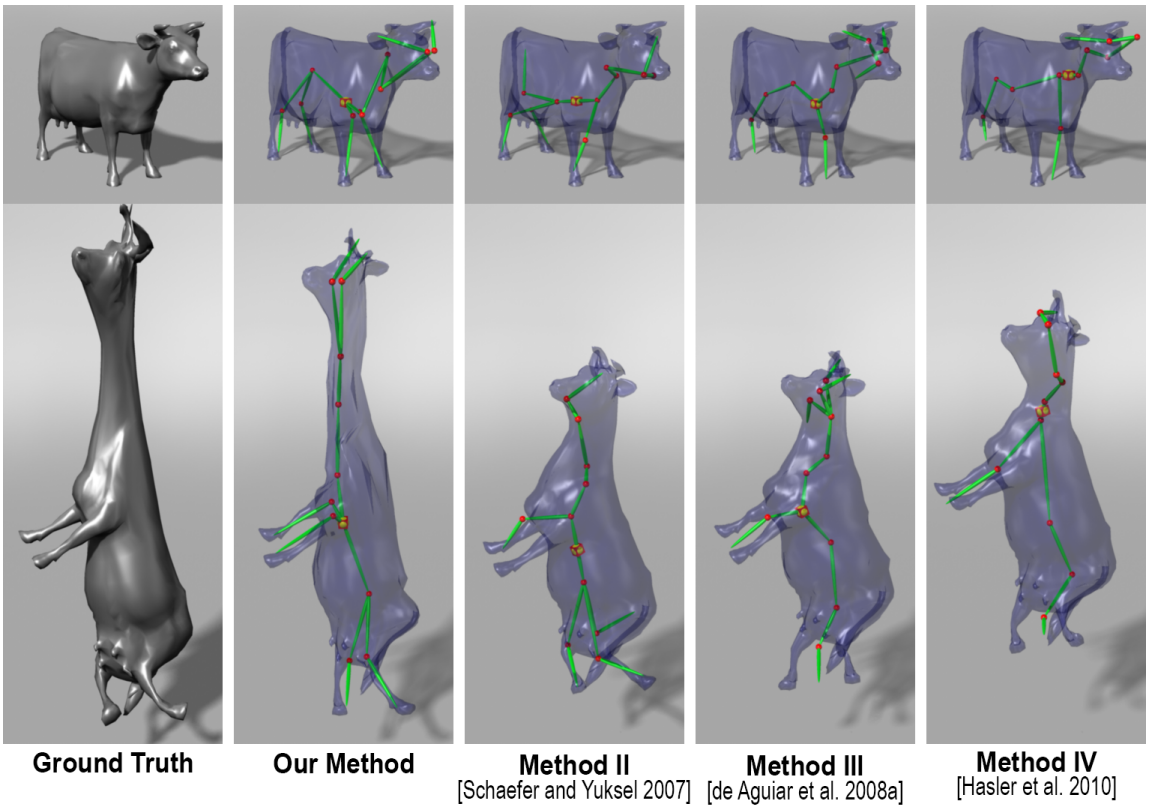

Figure 3.17: My proposed method can even rig an elastic model such as this stretched **cow**. The top row shows the resulting skeletons in the rest pose. Compared with the reconstructed result by my method, the reconstructed poses by the three previous methods are not visually close to the ground-truth.

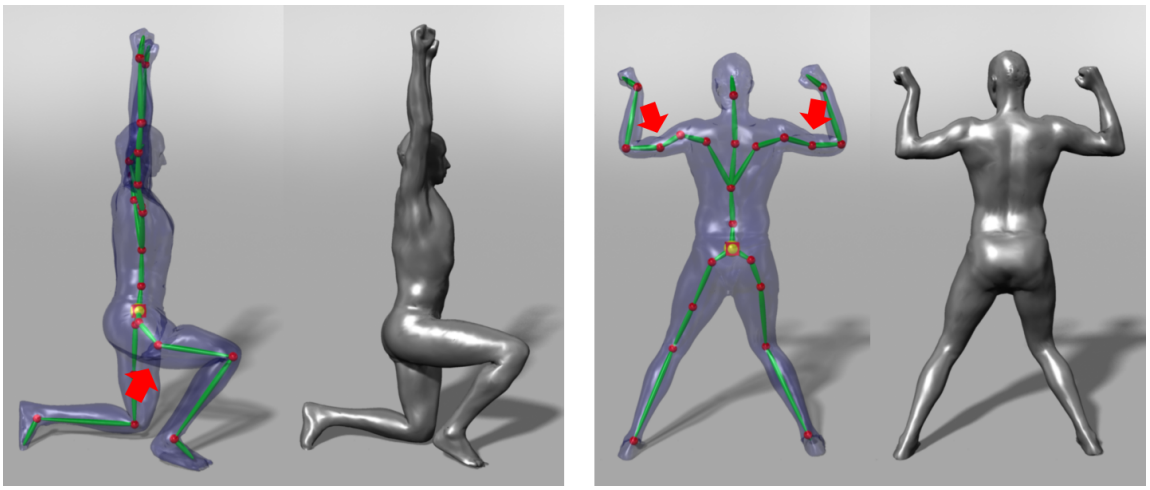

Figure 3.18: My approach with using skeleton LBS fails to remove 4 redundant bones in the upper arms and upper legs of the **scape** model, since the LBS model need them for a better approximation of muscle bulging on these parts (indicated by red arrows). To the end, my approach keeps the 4 bones to capture the non-linear deformation effect. The gray models are the ground-truth.

Chapter 4

Two-layer Blend Skinning Compression

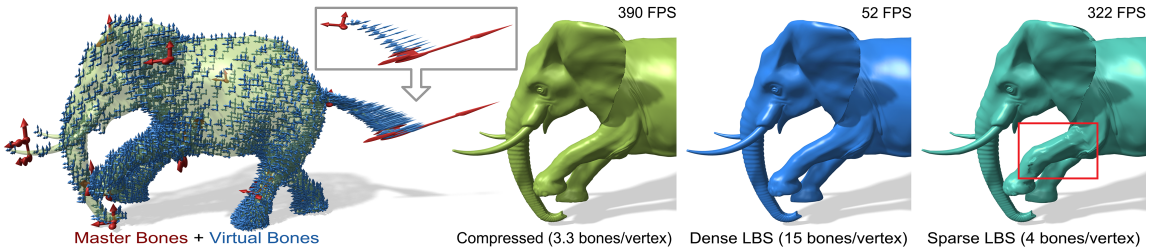

Figure 4.1: My compression model blends master bone transformations and caches them as virtual bone transformations (left most). It can compress a Linear Blend Skinning (LBS) model with dense weights and generate a fast and compact model without sacrificing the quality of skinning, compared with dense-weight LBS model.

Blend skinning is typically linear; it is controlled by a weight matrix, where each element defines the contribution of a bone (in skeleton-based skinning) or a

control point (in cage-based skinning) to interpolation of a mesh vertex. To speed up skinning performance, a sparseness constraint is often imposed on the weight matrix, that is, the weight matrix contains only a small proportion of non-zero elements. In practice, for the sake of effective parallel implementation on GPUs or multi-core CPUs, a more strict sparseness constraint is typically imposed on the weight matrix, which requires every vertex to be associated with no more than k bones or control points. On the one hand, the sparseness constraint has the advantages of saving computation and balancing workload between different processing cores. On the other hand, this setup has the following intrinsic limitations.

Limitation #1: It is difficult to handle *exceptional vertices* that are naturally associated with more than k bones or control points. The exceptional vertices typically appear on smooth and highly deformable regions of 3D models. As a specific example shown in Fig. 4.2(b), more than 23 percent of the vertices in a cheb model (illustrated in red), rigged by [7] with 17 bones, are influenced by all the bones. Also, exceptional vertices might be required by design; as an example, vertices on the palm region of a hand model (Fig. 4.2(c)) are influenced by several proximal phalanges and all five metacarpal bones. Indeed, due to existence of such exceptional vertices, a difficult trade-off often needs to be delicately handled in sparse-weight skinning models.

Limitation #2: Conventional weighted blending is inefficient from a computational perspective, despite the fact that performance has always been one of most

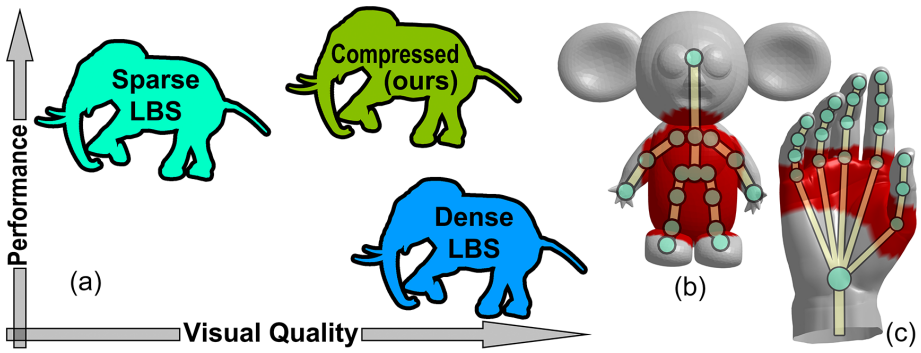

Figure 4.2: (a): Our skinning compression model can achieve a high performance with insignificant loss of visual quality. (b) and (c): Examples of exceptional vertices (illustrated in red color): the Cheb model (b) is rigged by [7], and a hand model (c) is rigged manually.

important concerns for skinning models [37]. Specifically, the weights of two neighboring vertices are typically similar if skinning is smooth. These similar linear combinations are calculated multiple times; if they can be properly cached (or compressed), overall computational cost can be measurably reduced.

Limitation #3: Imposing a sparseness constraint makes a skinning problem a selection of discrete variables that does not have any yet known optimal polynomial solutions [80]. Putting it together with convex (non-negativity and affinity) constraints on weights would make the skinning problem even more challenging. Any non-optimal solution might lead to a non-smooth skinning model or even a bad approximation [44]. Solutions to many skinning applications can be significantly simplified if a sparseness constraint does not need to be handled.

In this chapter, I present a lossy weight matrix compression approach 4.1 to

free skinning models from the sparseness constraint and thus overcome the above limitations (Fig. 4.2(a)). Based on the weight compression, I construct an effective two-layer blend skinning model to reduce computation of linear blend skinning (LBS) with dense weights (Fig. 4.3). 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. In the virtual bone blending layer, each vertex transformation is blended by no more than two virtual bones.

Compared with existing weight reduction techniques [32, 44], my model allows more flexible control on the trade-off between accuracy (skinning error) and performance (a desired number of blending operations). Through experiments and direct comparisons, I show that the proposed model achieves substantially smaller approximation errors than state of the art weight reduction techniques, given the same total number of bones (§4.4). I also analyze the performance and memory overhead of my approach on graphics hardware and its potential applications for other skinning models (§4.5).

4.1 Problem Formulation

Let $W \in \mathbb{R}^{k \times n}$ be the weight matrix of an input skinning model with n vertices and k bones, as illustrated at the top left of Fig. 4.3. We denote the i-th column of W (or the original weights of the i-th vertex) as w_i. We compress this original

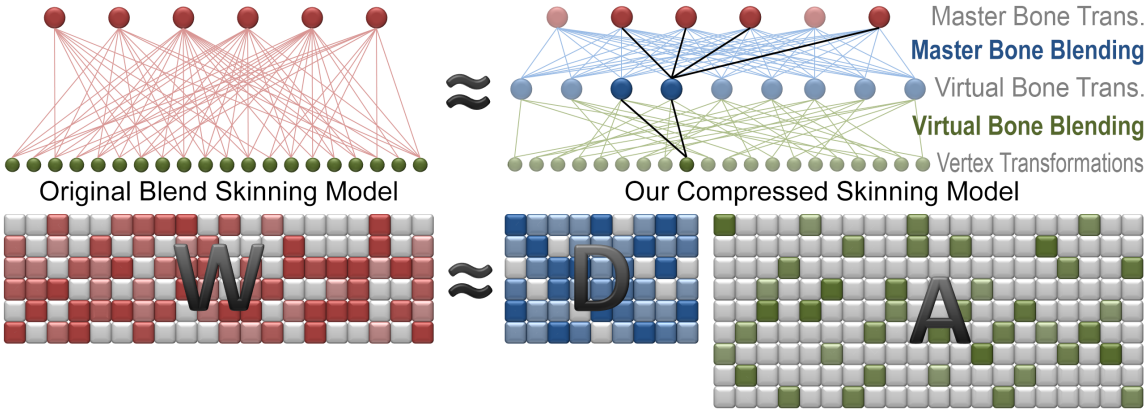

Figure 4.3: A conventional blend skinning model (top left) with a dense weight matrix W (bottom left) is approximated as a two-layer blending with virtual bones (top right). This is equivalent to factorizing W into a sparse dictionary D and a matrix of sparse coefficients A (bottom right). D has at most c non-zero elements, while A has at most 2 non-zero elements.

skinning model using a two-layer blending scheme with virtual bones (the top right of Fig. 4.3). At the first layer, a.k.a. *master bone blending*, we calculate and cache the transformations of m virtual bones by blending the transformations of k original bones (called *master bones*). At the second layer, a.k.a. *virtual bone blending*, we calculate the position of each vertex by blending the transformations of the virtual bones and applying the resultant transformation to the vertex. We also impose 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, we allow at most c blending operations for each virtual bone; at the virtual bone blending layer, we allow at most 2 blending operations for each vertex.

Letting $d_j \in \mathbb{R}^k$ be the blending weights of the j-th virtual bone, we can represent

all the master bone blending weights as a sparse matrix $D = [d_1, \ldots, d_m] \in \mathbb{R}^{k \times m}$. Similarly, letting $\alpha_i \in \mathbb{R}^m$ be the blending weights of the i-th vertex, we can represent all the virtual bone blending weights as another sparse matrix $A = [\alpha_1, \ldots, \alpha_n] \in \mathbb{R}^{m \times n}$, where each column α_i has at most two non-zero elements. With the above matrix representations, the blend skinning compression problem can be viewed as a sparse coding problem in which the original matrix W needs to be factorized into D (i.e., dictionary) and A (i.e., coefficients), as illustrated at the bottom of Fig. 4.3. Each column d_j is called an *atom* of the dictionary. This sparse coding problem is then formulated as minimizing the following quadratic error function:

$$\min_{D,A} {\Delta_W}^2 = \min_{D,A} \frac{1}{kn} \|DA - W\|_F^2 \tag{4.1a}$$

$$\text{Subject to: } \mathbf{card}(\alpha_i) \leq 2, \forall i \tag{4.1b}$$

$$\mathbf{card}(d_i) \leq c, \forall i \tag{4.1c}$$

Parameter selection. We denote $\mathbf{card}(x)$ as the cardinality (number) of non-zero elements in vector x. The constraint in Eq. (4.1b) enforces that at most two virtual bones can influence any particular vertex. This number (two) is empirically chosen for best performance. Since the number of vertices is typically large, minimizing the number of blending operations at the virtual bone blending layer would be the most effective way to reduce computational cost. In Eq. (4.1c), we also constrain the sparseness of D to reduce computational cost and thus improve performance. In order to maintain the approximation power of this model, we need to keep the sparseness of dictionary atoms to be no less than that of the original weights or vertices. However,

we empirically found that slightly increasing the sparseness of atoms could expand its approximation power. For this reason, we choose $c = \max_{i=1...n} \mathbf{card}(w_i) + 1$ in our experiments. Note that users can perform fine tuning on c to further optimize performance with different input data.

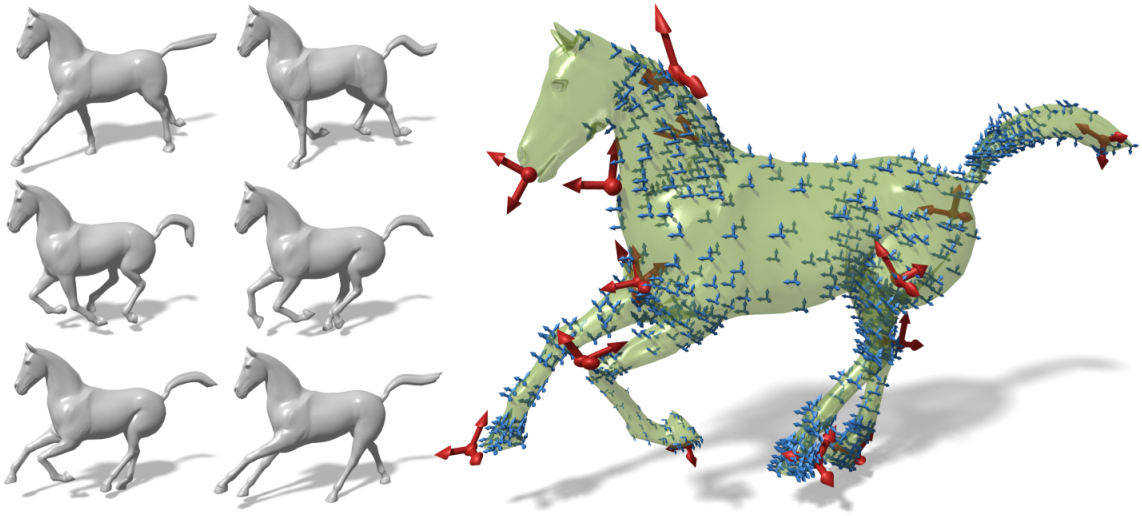

Figure 4.4: Some example poses of an animated mesh sequence (left) and its corresponding compressed blend skinning model (right). Master bone transformations are illustrated in red, and virtual bone transformations are illustrated in blue. We place each virtual bone at a vertex with the largest sparse coefficient. Our model distributes virtual bones adaptively so that more virtual bones are employed for highly deformed regions (e.g., the legs or the tail).

Approximation power and effectiveness of our model. Compared with directly imposing a sparseness constraint on skinning weights, our 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 our 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. 4.4.

In terms of parallel implementation, our compression model is friendly to stream processing architectures, such as GPUs, since blending operations for each node (virtual bone or vertex) are independent of each other, and they have the same workload (i.e., the same number of blending operations per node). Thus, the two blending layers in our model can be effectively implemented on GPUs as a process of two passes.

4.2 Sparse Compression Algorithm

The overview of our sparse weight matrix factorization for compressing blend skinning models is presented in **Algorithm** 3. Users can terminate this algorithm two different ways: specifying a maximum number of virtual bones (or dictionary size) Σ, or specifying an error threshold ϵ. In the former case, we can control the computational cost of the compressed model (i.e., number of blending operations). In the latter case, we can control the accuracy of the compressed model (i.e., the approximation error, Δ_W).

Algorithm 3 can be briefly summarized as follows: First, we initialize a minimum dictionary with 2 atoms and its corresponding coefficients (line 1 and line 2). Then

we sequentially add atoms 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 $m = \Sigma$.

Initialization. We initialize the first atom of the dictionary using the weights of a vertex with the largest ℓ^2-norm (i.e. $d_1 = \arg\max_{w_i} \|w_i\|_2$). The second atom is initialized as the weights of another vertex with the smallest dot product to d_1 (i.e., $d_2 = \arg\min_{w_i} \{w_i \cdot d_1\}$). The coefficients A can be solved per-vertex (column by column) by linear least squares with two unknowns (corresponding to two atoms) as follows: $\alpha_i = \arg\min_x \|Dx - w_i\|_2^2$.

Adding atoms. In order to minimize Δ_W, we always add the weights of a vertex p that has the largest error to the dictionary as follows (line 5):

$$D \leftarrow [D, w_p] \text{ s.t. } p = \arg\max_i \|D\alpha_i - w_i\|_2^2 \tag{4.2}$$

Instead of adding atoms to the dictionary one by one, followed by a joint dictionary-coefficients optimization, we can improve performance by adding $\kappa(\Delta_W, m)$ atoms (line 4) before each joint optimization. $\kappa(\Delta_W, m)$ is computed based on the current approximation error $\Delta_W(D, A)$ (refer to Eq. (4.1a)) and the current dictionary size m (Eq. (4.3)). Note that when adding each atom without dictionary optimization, we still need to update coefficients according to the added atom (line 7).

$$\kappa(\Delta_W, m) = \min\{\left(\frac{\Delta_W}{\varepsilon} - 1\right) m + 1, \Sigma - m\} \tag{4.3}$$

Dictionary-coefficients optimization. We solve the joint dictionary-coefficients optimization problem by a block coordinate descent approach with warm restarts [60], that is, alternatively updating the dictionary (line 10) and coefficients (line 11 to line 13). With the warm restarts, we found that its alternative update process can converge within a small number of iterations. In our experiments, we found that it typically converged within 3 iterations.

In the following sections, we describe details of two major steps in the algorithm 3, i.e., dictionary update (§4.2.1) and coefficients update (§4.2.2). Note that two coefficients update steps (line 7 and line 12) use the same procedure. To speed up performance, we only perform updates on a subset of vertices with potential coefficient changes. This is implemented as an ordered update process that starts from one vertex, i.e., vertex p in line 7 and vertex i at line 12.

4.2.1 Dictionary Update

At the dictionary update step, we employ an online dictionary update algorithm with warm restarts [53], since it is simpler and faster than other methods (e.g., K-SVD [1] or Newton's method [47]). Since our dataset is relatively small, we can use it at each update rather than sampling it. Specifically, we precompute Φ and Γ as follows:

$$\Phi = \sum_{i=1}^{n} \alpha_i \alpha_i^{\mathsf{T}} = [\phi_1, \ldots, \phi_m] \in \mathbb{R}^{m \times m} \tag{4.4a}$$

$$\Gamma = \sum_{i=1}^{n} w_i \alpha_i^{\mathsf{T}} = [\gamma_1, \ldots, \gamma_m] \in \mathbb{R}^{k \times m} \tag{4.4b}$$

Since α_i is sparse, the complexity of computing Φ and Γ is $\mathcal{O}(n)$. Then we update each atom (column) d_j of the dictionary as follows:

$$d_j \leftarrow \frac{1}{A_{j,j}} (\gamma_j - D\phi_j) + d_j \tag{4.5}$$

After each atom update, we enforce the sparseness constraint in Eq. (4.1c) by keeping the c largest elements of vector d_j, while setting the others to be 0. Finally, we normalize the result so that d_j satisfies the affinity constraint (i.e., the sum of all the elements equals to 1) by keeping it within the effective range of a floating point number on the machine. Although other normalization methods such as dividing d_j by the maximum element would work in theory, we found that dividing d_j by the sum of all the elements (normalization with the affinity constraint) can improve the rate of convergence since the weights $\{w_i\}$ are also constrained to be affinity.

4.2.2 Coefficients Update

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. Although the two optimum atoms can be determined by a greedy algorithm such as matching pursuit [54], it requires $\mathcal{O}(m)$ operations per vertex, or $\mathcal{O}(mn)$ operations for updating all the coefficients. To this end, we propose a fast coefficients update method that requires approximately $\mathcal{O}(n)$ operations

by assuming skinning weights are typically smooth across neighboring vertices (the fairness assumption on the manifold).

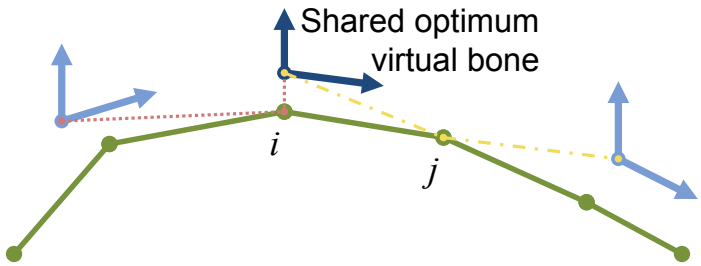

Figure 4.5: On a manifold, two neighboring vertices typically have similar skinning weights and coefficients; thus, they most likely share the same optimum virtual bones (dictionary atoms). This assumption can be used to accelerate the update of coefficients.

With this assumption, two neighboring vertices on a mesh would have similar skinning weights and coefficients. Furthermore, the two vertices might share the same optimum virtual bones, as illustrated in Fig. 4.5. If the two optimum atoms for vertex i are updated, we expect the two atoms might be the optimal ones for its neighboring vertices. Thus, we can perform the update as a preorder graph traversal, that is, we start coefficient updating at a vertex with two known optimum dictionary atoms and use these atoms as candidates to update the coefficients of its neighboring vertices; then we repeat the same process for these neighboring vertices. Specifically, we implement the coefficients update recursively as a depth-first search on a mesh edge-based graph (Algorithm 4).

In Algorithm 4, assuming vertex i has two candidate atoms with indices r and s, we update coefficients α_i and corresponding approximation error E_i (line 2) if

and only if the linear combination of the two candidate atoms d_r and d_s can improve E_i (line 1). Here we define the approximation error E_i of weight w_i as ${E_i}^2 = \|D\alpha_i - w_i\|_2^2$. Since the two candidate atoms are known, we solve the optimum α_i by least squares with two unknowns, i.e., $(\alpha_i)_r$ and $(\alpha_i)_s$. If w_i is constrained to be affinity, we also impose the affinity constraint to α_i to improve the rate of convergence. Specifically, we solve:

$$\begin{aligned} &\min \quad \|d_r(\alpha_i)_r + d_s(\alpha_i)_s - w_i\|_2^2 \text{ s.t. } (\alpha_i)_r + (\alpha_i)_s = 1 \\ &(\alpha_i)_r \\ &(\alpha_i)_s \end{aligned} \tag{4.6}$$

Once candidate atoms d_r and d_s are used to update coefficients α_i of vertex i, we then recursively update coefficients of its neighboring vertices $\mathcal{N}(i)$ (line 3 to line 6). Let j be a vertex in $\mathcal{N}(i)$, we first find candidate atoms r' and s' of j so that the linear combination of $d_{r'}$ and $d_{s'}$ best approximates w_j (line 4). We only select r' and s' within the set of current optimum atoms of vertex j and the candidate atoms of vertex i. Specifically, let ρ_j and σ_j be indices of current optimum atoms of vertex j, i.e. $(\alpha_j)_{\rho_j}$ and $(\alpha_j)_{\sigma_j}$ are non-zero coefficients, we only select r' and s' in the set of $\{r, s, \rho_j, \sigma_j\}$. In the end, we consider six combinations of two atom indices and solve linear least squares for each of them, similar to Eq. (4.6), to find the best combination.

Employing coefficients update in the main algorithm. We employ the above recursive algorithm 4 in the main algorithm 3 for two purposes. First, after weights w_p are added to the dictionary (line 6 in Algorithm 3), the new dictionary

atom w_p is the optimum atom for vertex p. We start a recursive coefficients update from vertex p (line 7) with the latest added dictionary atom w_p. The other candidate atom can be selected arbitrarily. Compared to a conventional solution of updating coefficients for all the vertices, our algorithm can save a significant amount of unnecessary update cost for vertices far away from p on the mesh that are not potentially affected by the latest added atom (virtual bone).

Second, in full coefficients update step, we start the recursive coefficients update one by one from every vertex (line 12 in Algorithm 3). For each vertex i, we keep its current optimum atoms and use them as candidate atoms. In other words, candidate atoms d_r and d_s for vertex i will be the atoms such that $(\alpha_i)_r \neq 0$ and $(\alpha_i)_s \neq 0$. If the least squares solution using these candidate atoms improves the approximation error, the update will be propagated through its neighboring vertices on the mesh. In general, this recursive update process might visit each vertex several times, which makes the amortized complexity of the full coefficient update step $\mathcal{O}(n)$ in practice. We will further analyze performance of our algorithm through experiments in §4.4.

4.3 Factorization from Example Poses

Our model can also utilize example poses for a better approximation. Although we can potentially extend our weight factorization algorithm for various skinning models such as cage based deformation [33] or dual-quaternion blending [35], without loss of generality, in this work we only demonstrate our model for the most popular LBS approach.

In order to utilize the example poses, we need to find a compression model that best approximates given example poses instead of a compression model that best approximates skinning weights. Similar to previous works [32, 37, 45], we also minimize a quadratic error function on all the example poses given corresponding bone transformations. Fortunately, we can still adapt our sparse weight matrix factorization (Algorithm 3) by replacing the approximation error on skinning weights with the approximation error on example poses.

Assume we have f example poses. Let $v_i^t \in \mathbb{R}^3$ be the position of vertex i in example pose t, and $u_i \in \mathbb{R}^3$ be the rest pose position of vertex i. For each example pose t, we represent its bone transformations as matrices, and use $T_j^t \in \mathbb{R}^{3\times 4}$ to denote the transformation of bone j relative to its position at the rest pose. Given the example poses and bone transformations, we can solve the optimized LBS weights $W^* = [w_1^*, \ldots, w_n^*] \in \mathbb{R}^{k\times n}$ using linear least squares with equality constraint ($\sum_{j=1}^{k} (w_i^*)_j = 1$) and inequality constraint ($w_i^* \geq 0$) [45]. Note that the optimized weights W^* are not constrained to be sparse. We utilize example poses by modifying certain steps in Algorithm 3 as follows.

Approximation error. We replace the approximation error on weight matrix ${\Delta_W}^2$ (Eq. (4.1)) with the approximation error on example poses ${\Delta_E}^2$ (Eq. (4.7a)). The approximation error for each vertex ${E_i}^2$ (Eq. (4.7b)) is used to find new atoms for the dictionary (line 5 in Algorithm 3). Here, we normalize the approximation error by subtracting the lower bound E_i^{*2} (i.e., the approximation error with the optimized LBS weights w_i^*, refer to Eq. (4.7c)) from it.

$$\Delta_E{}^2 = \frac{1}{3fn}\sum_{i=1}^{n} E_i{}^2 \tag{4.7a}$$

$$\text{Where: } E_i{}^2 = \sum_{t=1}^{f}\left\|\sum_{j=1}^{k}(D\alpha_i)_j T_j^t \begin{bmatrix} u_i^t \\ 1 \end{bmatrix} - v_i\right\|_2^2 - E_i^{*2} \tag{4.7b}$$

$$E_i^{*2} = \sum_{t=1}^{f}\left\|\sum_{j=1}^{k}(w_i^*)_j T_j^t \begin{bmatrix} u_i^t \\ 1 \end{bmatrix} - v_i\right\|_2^2 \tag{4.7c}$$

Coefficients update. We also use the approximation error $E_i{}^2$ on example poses (Eq. (4.7b)) to solve coefficients for the linear combination of atoms (line 1 and line 4 in Algorithm 4).

Dictionary update. We use the optimized weights W^* to pre-compute matrix Γ instead of W (refer to Eq. (4.4b)). Then we perform the same column update using Eq. (4.5). It should be noted that this dictionary update neither gives the best error reduction nor guarantees convergence in theory. However, this simple update is faster and more effective than other complicated methods, e.g., employing divergence protections such as Shift-cutting [60] or Marquardt parameter [55].

4.4 Results and Comparisons

To evaluate our approach, we first performed skinning decomposition on a number of animated mesh sequences provided by [71, 74] (reported in Table 4.1) to extract their optimized linear blend skinning models. For fair comparisons between different

sequences, we first rescaled all the datasets so that their rest poses are tightly fit in a unit sphere. Then, for each sequence, we implemented the smooth skinning decomposition algorithm [45] to extract its flexible bone transformations and convex weight map.

Name	n	f
samba	9971	175
camel-gallop	21887	48
elephant-gallop	42321	48

Name	n	f
horse-gallop	8431	48
camel-collapse	21887	53
horse-collapse	8431	53

Table 4.1: The test datasets used in this work. n denotes the number of vertices, and f denotes the number of example poses.

We also considered different levels of the sparseness constraint on the weight map by setting different maximum numbers of bones per vertex (namely, 4 bones per vertex, 8 bones per vertex, and dense weights without the sparseness constraint). In our compressed skinning model, we calculate *the number of bones per vertex* as the average number of blending operations per vertex (including the blending operations on both layers). Specifically, the number of bones per vertex in our model is $\frac{mc}{n} + 2$. To compare the approximation powers of different approaches on the example poses, we use a fitting error measure E proposed by Kavan et al. [37]. In this section, we report all errors multiplied by 1000 for the sake of convenience.

In Fig. 4.6, we show several example results of our skinning compression model with different thresholds of the objective function. We can hardly observe visual distortions if the error threshold drops below 0.001. If example poses are not utilized

in our approach, the value of the objective function Δ_W represents the average error in original bone-vertex weights, since the original bone-vertex weights satisfy the affinity constraint. Δ_W can also represent the *relative* distortion of the compressed model, compared to the original blend skinning model. Otherwise, if example poses are used, the value of the objective function Δ_E represents the *absolute* distortion, compared to the original skinning model.

Since all the datasets are rescaled to tightly fit in a unit sphere, the amounts of distortion caused by the both compression methods (with/without utilizing example poses) are highly similar if the same error threshold is used. This relation is illustrated in Fig. 4.7. In this figure, at each step we incrementally add 10 virtual bones to our skinning compression approach and then perform joint dictionary-coefficients optimization. For each step, the fitting errors on the example poses are plotted along with the objective function.

Figs. 4.6 and 4.7 also illustrate that, to achieve the same approximation accuracy, utilizing the example poses can produce a compressed skinning model with a smaller number of virtual bones than without utilizing the example poses. In addition, when the example poses are provided, we have a more accurate way to control the fitting error. Specifically, we can calculate the fitting error using the following equation: $E^2 = {\Delta_E}^2 + E^{*2}$, where E^{*2} is the lower bound of the fitting error (i.e., the fitting error with the optimized LBS weights W^*).

Comparison without utilizing example poses. In Table 4.2, we compare the approximation errors on weight matrix (i.e., Δ_W) among our method, k-largest weight reduction, and smooth weight reduction [44]. We can see that, for all the cases,

our approach achieves substantially smaller approximation errors than the other two approaches. In this comparison, all the three methods only perform reduction on the weight matrix without utilizing example poses. We use two sets of input skinning models generated by skinning decomposition [45]: 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. Given a reduced number of bones per vertex, k_r, we calculate the number of virtual bones (or the size of the dictionary) for our method as $\Sigma = \frac{(k_r-2)n}{c}$. In the k-largest weight reduction method, we keep only the k_r largest weights per vertex and then normalize the sum of these weights to be 1. In the smooth weight reduction method [44], at the normalization step, it also takes mesh fairness into account by matching the Laplacian of reduced weights with that of the original weight map. Incorporating smoothness to the reduction process would increase the approximation error on the weight matrix.

Comparison with utilizing example poses. If example poses are used, we also compare our method with geometry weight reduction [32], poisson weight reduction [44], and smooth skinning decomposition [45]. From an original skinning model with k bones per vertex, the geometry weight reduction method selects k_r bones per vertex with best approximation, normalizes the weights of the k_r bones, and then minimizes the fitting error on the example poses using linear least squares. Similar to the smooth weight reduction [44], the Poisson weight reduction method takes mesh fairness into account by matching Laplacians on the example poses.

Qualitative comparisons among the weight reduction methods are presented in Fig. 4.8. All the methods reduce the skinning models with dense weights to 4 bones per vertex. From this figure, we can 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 smooth and Poisson weight reduction methods are smoother and more pleasing than those by the k-largest and geometry weight reduction methods. Meanwhile, without noticeable visual distortions, our skinning compression method approximates the original skinning models significantly better than the four weight reduction methods (i.e., smooth, Poisson, k-largest, and geometry). In particular, when the example poses are utilized, our 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 [45] with 4 bones per vertex, e.g. in the case of the elephant-gallop model.

In Table 4.3, we also quantitatively compare the fitting errors on example poses among all the methods. The results show that our model can substantially outperform the four chosen weight reduction methods. In most cases, our results are even better than the employed original skinning decomposition approach [45] with the same number of bones. Note that this comparison gives certain bias to the employed skinning decomposition approach [45], since it is allowed to optimize bone transformations and the rest pose to fit with the reduced weights.

Performance. We also illustrate the running time of our approach for some test

cases in Fig. 4.9. We implemented our approach in C++ and ran all the experiments on an off-the-shelf computer with a single 2.0 GHz CPU core. As illustrated in this figure, when the example poses are utilized, our approach will need more than an order of magnitude running time than the case without utilizing example poses. For the same number of master bones, we observe the running time of our compression increases linearly with respect to the size of the input data. Approximately, the running time without example poses increases linearly with respect to the number of vertices n, and the running time with the example poses increases linearly with respect to the size of all the example poses (i.e., $n \times f$). This observation shows that our algorithm, especially the coefficients update (§4.2.2), has approximately a linear complexity in practice.

4.5 Discussion

Besides the LBS model (as demonstrated above), our two-layer blend skinning compression approach can also be potentially extended to handle and compress other skinning models. For instance, we could reduce non-linear skinning models with dense weights such as dual-quaternion blend skinning [35] by replacing the linear coefficients update in §4.2.2 with a non-linear least squares solver (e.g., the Levenberg-Marquardt algorithm [55]). Our compression method could also work for cage-based deformation models by using an objective function similar to the one used in [44]. Note that additional non-trivial analysis would be required to polish the above thoughts.

Our current approach has certain limitations.

- First, our two-layer blending approach imposes some computational overhead. In Fig. 4.10, we show GPU performance comparison between our approach and the LBS model. Our two-layer model performs much faster than the LBS model with dense weights. Our model and the LBS with sparse weights have similar performance in some scenarios that require intensive computation, i.e., models with 8 bones per vertex or models with a high resolution (e.g., the camel model (21887 vertices) and the elephant model (42321 vertices)). In other scenarios that require light computation, our model performs slower due to extra I/O operations on the buffers and cost for synchronization between two passes on GPUs. Fortunately, since the overhead is insignificant for complex models, our approach is suitable well for skinning applications that require both high performance and visual quality.

- Second, our model requires additional storage space for caching virtual bone transformations. Fig. 4.11 visualizes the memory overhead required by our approach, compared with linear blend skinning with the same number of bones per vertex. Note that the total memory required by our approach to store blending weights is equal to the memory for storing the linear blending skinning weights. The latter is proportional to the number of vertices of a 3D model.

- Finally, transformation blending in our approach cannot go beyond certain intrinsic limitations of the LBS model, among which sophisticated deformation effects such as muscle bulges or skin wrinkles cannot be captured well. Several

sound solutions have been proposed to alleviate this problem including multi-weight blending [77] and skinning correction [50, 42].

Within specific contexts, we can modify or extend our compression model for better performance. For example, the number of virtual bones per vertex in the virtual bone blending layer can be increased for smoother skinning. However, this will measurably affect performance, since the number of blending operations will increase by n (the number of vertices) when the number of virtual bones per vertex increases by one. In addition, we can also increase the number of blending layers, which will increase the number of combinations of master bone blending operations at an exponential rate. Nevertheless, increasing the number of layers would put additional overhead on skinning models, especially with common multi-pass implementation on GPUs. We believe that analyzing and extending our compression model in these directions will be useful for certain skinning applications.

Algorithm 3 Sparse Weight Matrix Factorization

Input: a weight matrix $W = [w_1, \ldots, w_n] \in \mathbb{R}^{k \times n}$, an error threshold ε OR a maximum number of virtual bones Σ.

Output: $D = [d_1, \ldots, d_m] \in \mathbb{R}^{k \times m}$ and $A = [\alpha_1, \ldots, \alpha_n] \in \mathbb{R}^{m \times n}$ s.t. Eq. (4.1)

Initialize a dictionary with $m = 2$ atoms: $D = \{d_1, d_2\}$
Initialize coefficients A according to D
repeat
  for $t = 1 \rightarrow \kappa(\Delta_W, m)$ **do**
    Find vertex p with the largest approximation error
    Add w_p to the dictionary
    Update coefficients A from vertex p
  end for
  repeat
    Update dictionary D
    for $i = 1 \rightarrow n$ **do**
      Update coefficients A from vertex i
    end for
  until Convergence
until $\Delta_W < \varepsilon$ OR $m = \Sigma$
return D and A

Algorithm 4 CoefficientsUpdate(vertex i, candidate atoms d_r, d_s)

Input: vertex i, candidate atoms d_r, d_s

1: **if** the linear combination of d_r and d_s improves error E_i **then**
2: Update α_i and E_i 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 $d_{r'}$ and $d_{s'}$ best approximates w_j
5: CoefficientsUpdate(j, $d_{r'}$, $d_{s'}$)
6: **end for**
7: **end if**

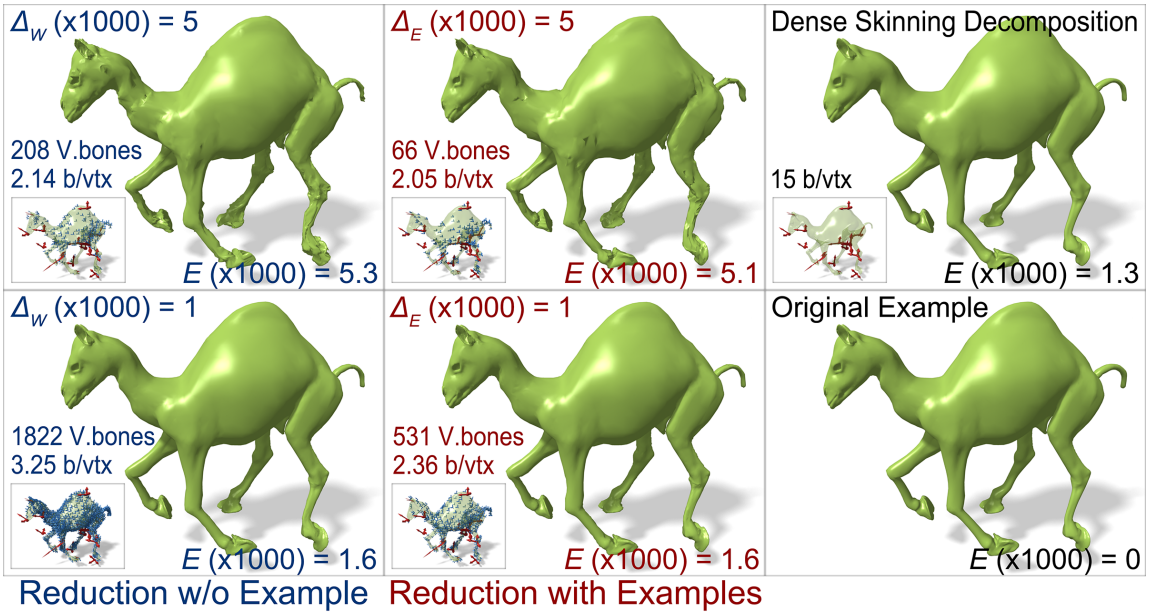

Figure 4.6: Example results of our skinning compression model with different thresholds of the objective function (Δ_W and Δ_E). We can hardly observe visual distortions if the error threshold drops below 0.001. The input skinning model (camel gallop) has 15 bones with a dense weight matrix. For each compression example, its bone distribution is illustrated at the bottom-left corner. Master bone transformations are illustrated in red; virtual bone transformations are illustrated in blue. The fitting error on the example poses, E, is also reported. "V. Bones" denotes virtual bones; "b/vtx" denotes bones per vertex.

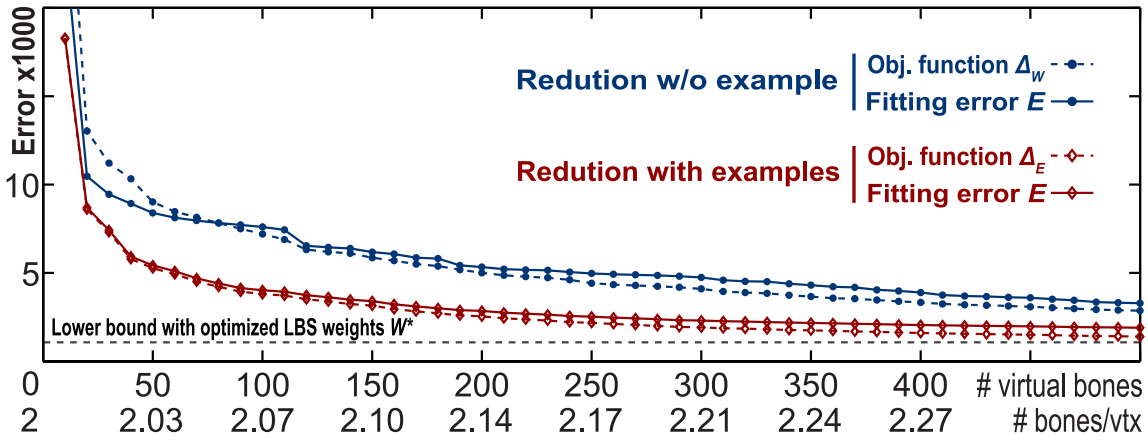

Figure 4.7: Relation of the objective function and the fitting error on the example poses. The input skinning model has 15 bones with a dense weight matrix extracted from a camel gallop sequence. The relation between value of the objective function and the fitting error is quite similar for the both cases (i.e., with/without utilizing example poses). Compression utilizing the example poses produces smaller fitting errors than that without utilizing the example poses.

Name$_{[k]}$	8 to 4 bones/vertex			Dense to 4 bones/vertex			Dense to 8 bones/vertex		
	k-Largest	Smooth	Ours	k-Largest	Smooth	Ours	k-Largest	Smooth	Ours
samba$_{10}$	19.6	24.5	**1.2**	21.9	26.6	**1.1**	1.8	2.4	**0.2**
samba$_{20}$	14	18	**1.7**	18.1	22.2	**2.4**	5	6.5	**0.9**
camel-gallop$_{15}$	12.3	18.8	**0.5**	19.5	27.6	**0.8**	5	8.9	**0.2**
camel-gallop$_{30}$	8.8	14	**0.6**	19.7	26.2	**1.3**	7.9	12.6	**0.5**
elephant-gallop$_{15}$	16.4	24.5	**0.5**	30.3	42	**0.8**	8.1	12.1	**0.3**
elephant-gallop$_{30}$	12.8	20.1	**0.6**	25.9	34.4	**1.3**	10	14.6	**0.5**
horse-gallop$_{15}$	13	17.3	**0.9**	157.7	161.5	**1.3**	4.9	6.6	**0.3**
horse-gallop$_{30}$	7.3	9.6	**0.8**	23.1	29.6	**2.4**	9.1	12.4	**0.9**
camel-collapse$_{20}$	14.7	23.2	**1.3**	22	30.1	**1.8**	7.5	12.4	**0.6**
camel-collapse$_{40}$	9.9	15.9	**1.6**	18.8	24.5	**2.5**	8.3	12.1	**1.1**
horse-collapse$_{20}$	15.9	22.4	**2.8**	28.8	37	**4**	10.1	14.2	**1.5**
horse-collapse$_{40}$	10.8	14.7	**3.3**	23.6	29.4	**5.1**	11	14.6	**2.7**

Table 4.2: Comparison of the approximation errors on weight matrix (Δ_W): k-largest weight reduction, smooth weight reduction [44], and our method. In this comparison, example poses are not used. The number of bones k is shown as a subscript of input model name. The errors are multiplied by 1000 for convenience.

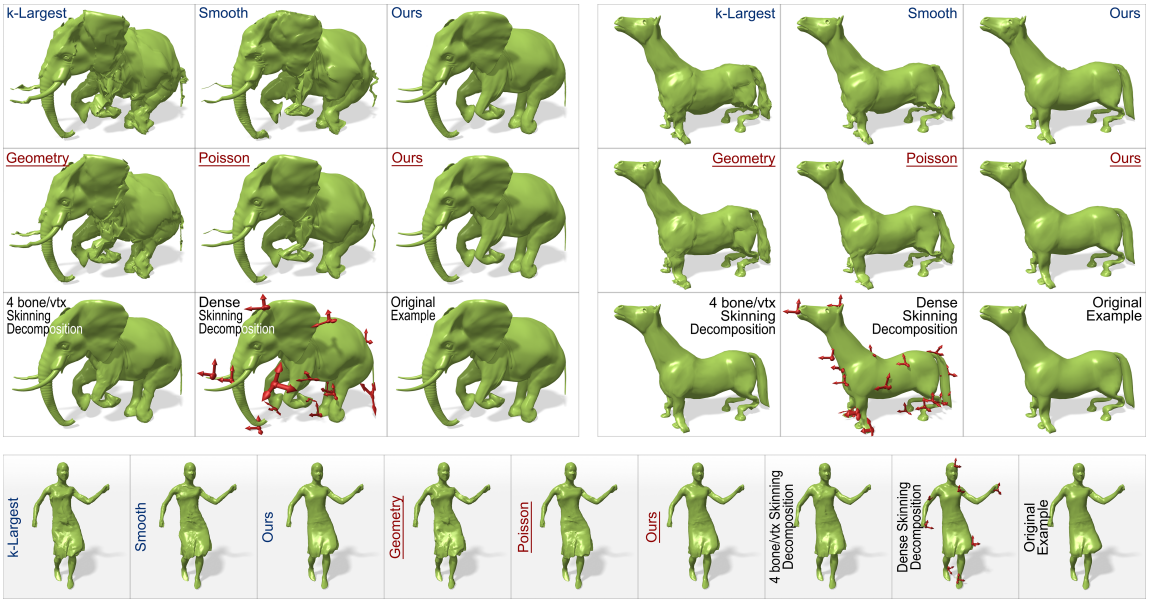

Figure 4.8: Comparisons between our method and several selected skinning weight reduction methods (i.e., k-largest weight reduction, smooth weight reduction [44], geometry weight reduction [32], and Poisson weight reduction [44]). The three used models are elephant-gallop with 15 bones (top-left), horse-collapse with 20 bones (top-right), and samba with 10 bones (bottom). All the methods reduce the skinning models with dense weights (obtained via the smooth skinning decomposition method [45]) to 4 bones per vertex. The methods noted with blue do not utilize example poses, while the methods noted with red utilize example poses. 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 [45].

Name$_{[k]}$	8 to 4 bones/vertex						Dense to 4 bones/vertex						4 b/v SD	Dense to 8 bones/vertex						8 b/v SD	Dense SD
	k-Largest	Smooth	Ours w/o Ex	Geometry	Poisson	Ours w/ Ex	k-Largest	Smooth	Ours w/o Ex	Geometry	Poisson	Ours w/ Ex		k-Largest	Smooth	Ours w/o Ex	Geometry	Poisson	Ours w/ Ex		
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 4.3: Comparison of the fitting errors on example poses (E) among our skinning compression method and several selected weight reduction methods: k-largest weight reduction, geometry weight reduction method [32], the smooth weight reduction method [44], and the Poisson weight reduction method [44]. "SD" denotes the employed original skinning decomposition method [45]. "b/v" denotes bones per vertex. "w/ Ex" (or "w/o Ex") denotes with (or without) utilizing example poses in our approach. 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 k 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.

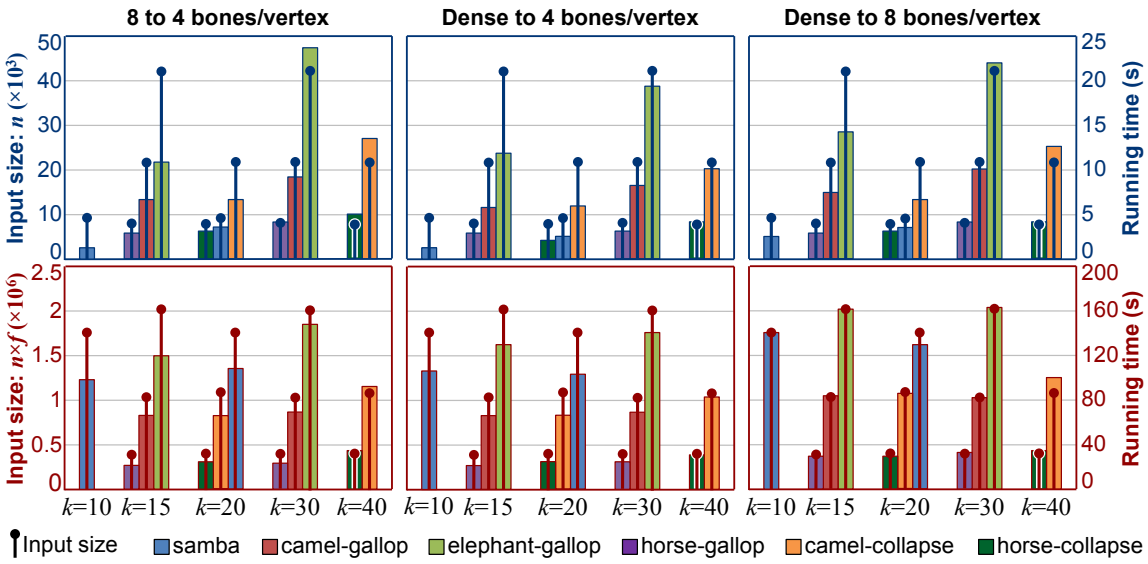

Figure 4.9: Running time (seconds) of our skinning compression approach on CPU. The first row shows running time of our compression without using example poses and the bottom row shows the running time with using example poses. k denotes the number of master bones, n denotes the number of vertices of the input 3D model, and f is the frame number of example poses.

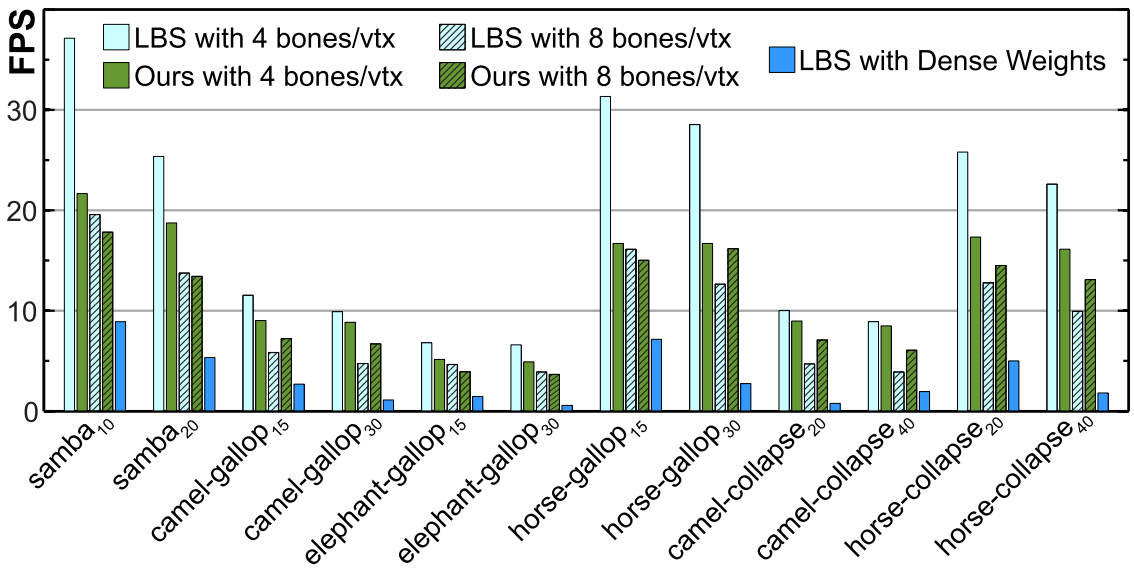

Figure 4.10: GPU Performance comparison between LBS and our approach. Both methods were implemented with Microsoft Direct3D 11 running on NVIDIA Geforce GT 540M. Normal vectors were transformed and then normalized to unit length. The performance was measured by animating 150 instances and reporting FPS for the entire pipeline including rendering.

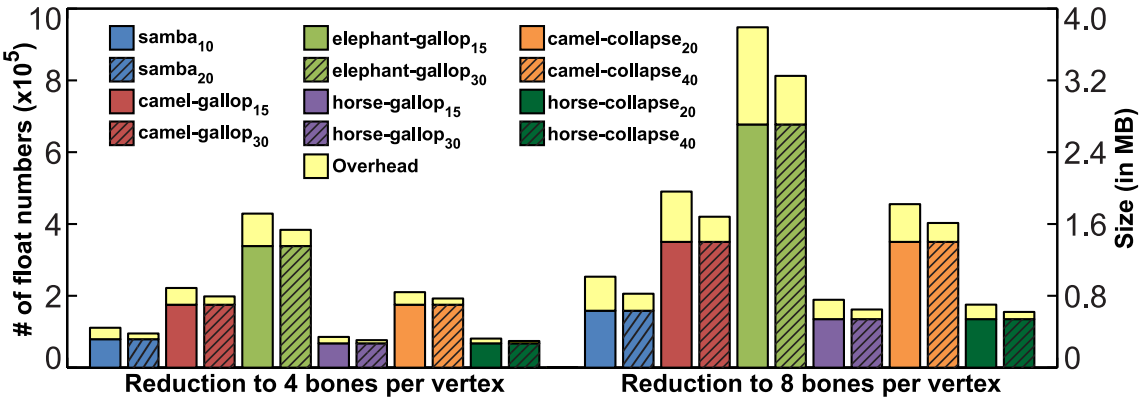

Figure 4.11: The memory overhead required by our approach to cache virtual bone transformations, compared with the linear blend skinning model with the same number of bones per vertex. Here we assume each vertex is stored as four 32-bit floating point numbers, a transformation matrix is stored as sixteen floating point numbers, and a blending weight with index is stored as two floating point numbers.

Chapter 5

Conclusion

This dissertation focuses on two fundamental problems of character skinning. (1) Model setup: My skinning from examples method extracts the skeleton-based LBS model. Such an extracted skinning model makes the editing task become more intuitive, and it is also compatible with mainstream 3D modeling and animation tools and 3D game engines. With the iterative linear solver I developed, my method offers more robust and more accurate results than the previous work. (2) Hardware accelerated rendering of skinning animation: I introduced an efficient two-layer LBS model to substantially reduce the computational cost of a dense-weight skinning model, with insignificant loss of its visual quality. This model is constructed by sparse coding technique using dense skinning weights or using animated mesh sequence to further improve its accuracy. This method significantly outperforms state-of-the-art weight reduction algorithms, as well as skinning decomposition algorithms with the sparseness constraint.

I believe these skinning methods might have numerous potential applications in both graphics and other disciplines. For example, skinning can be used as a data compression or a data reduction model, serving for hardware accelerated rendering or real-time simulation. In addition, skinning decomposition with skeleton extraction can be used in bio-mechanical and medical applications to analyze the motion and structure of articulated subjects. Skinning techniques could also benefit performance capture techniques since a deep understanding about the relation between skin deformations and object structure, e.g. skeleton or motion actuators, would help to limit the search space for 3D reconstruction and surface tracking. Skinning models can be used as a regularizer for building performance capture systems. With some extensions, skinning from example methods can also be applied on raw 3D scan data such as noisy, incomplete, and unorganized point cloud datasets.

However, the proposed models still has certain limitations. The first limitation is the limited approximation power of the linear blending models, which makes them impossible to capture complicated deformations such as muscle bulges or skin wrinkles. Thus, high-quality visual results call for effective and accurate skinning models. In the future, I plan to develop high quality skinning models with intuitive user controls, e.g. via semantic or dragging handles. This would significantly benefit artists and practitioners who use the current animation pipeline. Other relevant research directions I would like to pursue include: incorporating physics for skinning, interactive and seamless correction by artists, and hybrid simulation driven skinning methods. The second limitation is the example data dependency, which is also a common limitation of any data driven method. For example, if the input sequence only consists

of very limited or biased motions, then my proposed approach might not be able to extract its skinning model with a high accuracy. I believe some potential solutions to tackle these issues would be to utilize templates or introduce certain user interventions. The third limitation is its relatively high computational cost, although it can be alleviated, to a certain extent, via GPUs or multi-core CPUs based parallel implementation.

Bibliography

[1] M. Aharon, M. Elad, and A. Bruckstein. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, Nov. 2006.

[2] M. Alexa, D. Cohen-Or, and D. Levin. As-rigid-as-possible shape interpolation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 157–164, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[3] D. Ali-Hamadi, T. Liu, B. Gilles, L. Kavan, F. Faure, O. Palombi, and M.-P. Cani. Anatomy transfer. *ACM Transaction on Graphics*, 32(6):188:1–188:8, Nov. 2013.

[4] D. Anguelov, D. Koller, H.-C. Pang, P. Srinivasan, and S. Thrun. Recovering articulated object models from 3D range data. In *UAI'04: Proceedings of Conference on Uncertainty in Artificial Intelligence*, pages 18–26, 2004.

[5] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis. Scape: Shape completion and animation of people. *ACM Transaction on Graphics*, 24(3):24:408–24:416, Jul. 2005.

[6] O. K.-C. Au, C.-L. Tai, H.-K. Chu, D. Cohen-Or, and T.-Y. Lee. Skeleton extraction by mesh contraction. *ACM Transaction on Graphics*, 27(3):44:1–44:10, Aug. 2008.

[7] I. Baran and J. Popović. Automatic rigging and animation of 3D characters. *ACM Transaction on Graphics*, 26(3):72:1–72:8, Jul. 2007.

[8] J. Barbič, A. Safonova, J.-Y. Pan, C. Faloutsos, J. K. Hodgins, and N. S. Pollard. Segmenting motion capture data into distinct behaviors. In *Proceedings of Graphics Interface 2004*, GI '04, pages 185–194, 2004.

[9] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, Sept. 1999.

[10] B. Bickel, M. Botsch, R. Angst, W. Matusik, M. Otaduy, H. Pfister, and M. Gross. Multi-scale capture of facial geometry and motion. *ACM Transaction on Graphics*, 26(3):33:1–33:10, Jul. 2007.

[11] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM: Society for Industrial and Applied Mathematics, 1st edition, 1996.

[12] H. M. Briceño, P. V. Sander, L. McMillan, S. Gortler, and H. Hoppe. Geometry videos: a new representation for 3D animations. In *SCA'03: Proceedings of 2003 Symposium on Computer Animation*, pages 136–146, 2003.

[13] R. Bro and S. De Jong. A fast non-negativity-constrained least squares algorithm. *Journal of Chemometrics*, 11(5):393–401, 1997.

[14] E. de Aguiar, C. Stoll, C. Theobalt, N. Ahmed, H.-P. Seidel, and S. Thrun. Performance capture from sparse multi-view video. *ACM Transaction on Graphics*, 27:98:1–98:10, Aug. 2008.

[15] E. de Aguiar, C. Stoll, C. Theobalt, N. Ahmed, H.-P. Seidel, and S. Thrun. Performance capture from sparse multi-view video. *ACM Transaction on Graphics*, 27(3):98:1–98:10, Aug. 2008.

[16] W.-W. Feng, B.-U. Kim, and Y. Yu. Real-time data driven deformation using kernel canonical correlation analysis. *ACM Transaction on Graphics*, 27:91:1–91:9, Aug. 2008.

[17] K. Forbes and F. E. An efficient search algorithm for motion data using weighted pca. In *SCA'05: Proceedings of 2005 Symposium on Computer Animation*, SCA '05, pages 67–76, 2005.

[18] J. Gain and D. Bechmann. A survey of spatial deformation from a user-centered perspective. *ACM Transaction on Graphics*, 27:107:1–107:21, Nov. 2008.

[19] B. Georgescu, I. Shimshoni, and P. Meer. Mean shift based clustering in high dimensions: a texture classification example. In *ICCV'03*, pages 456–463, 2003.

[20] P. Gill, W. Murray, and M. Wright. *Practical Optimization*. Academic Press, London, UK, 1981.

[21] I. Guskov and A. Khodakovsky. Wavelet compression of parametrically coherent mesh sequences. In *SCA'04: Proceedings of 2004 Symposium on Computer Animation*, pages 183–192, 2004.

[22] F. Hahn, S. Martin, B. Thomaszewski, R. Sumner, S. Coros, and M. Gross. Rig-space physics. *ACM Transaction on Graphics*, 31(4):72:1–72:8, Jul. 2012.

[23] F. Hahn, B. Thomaszewski, S. Coros, R. W. Sumner, and M. Gross. Efficient simulation of secondary motion in rig-space. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '13, pages 165–171, New York, NY, USA, 2013. ACM.

[24] O. Härkegård. Efficient active set algorithms for solving constrained least squares problems in aircraft control allocation. In *Proceedings of the 41st IEEE Conference on Decision and Control*, volume 2, pages 1295–1300. IEEE, 2002.

[25] N. Hasler, T. Thormählen, B. Rosenhahn, and H.-P. Seidel. Learning skeletons for shape and pose. In *I3D'10: Proceedings SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 23–30, 2010.

[26] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, 1987.

[27] H. Huang, S. Wu, D. Cohen-Or, M. Gong, H. Zhang, G. Li, and B. Chen. L1-medial skeleton of point cloud. *ACM Transaction on Graphics*, 32(4):65:1–65:8, Jul. 2013.

[28] T. Igarashi, T. Moscovich, and J. F. Hughes. As-rigid-as-possible shape manipulation. In *ACM SIGGRAPH 2005 Papers*, pages 1134–1141, 2005.

[29] A. Jacobson, I. Baran, L. Kavan, J. Popović, and O. Sorkine. Fast automatic skinning transformations. *ACM Transaction on Graphics*, 31(4):77:1–77:10, Jul. 2012.

[30] A. Jacobson, I. Baran, J. Popović, and O. Sorkine. Bounded biharmonic weights for real-time deformation. *ACM Transaction on Graphics*, 30(4):78:1–78:8, Jul. 2011.

[31] A. Jacobson and O. Sorkine. Stretchable and twistable bones for skeletal shape deformation. *ACM Transaction on Graphics*, 30:165:1–165:8, Dec. 2011.

[32] D. L. James and C. D. Twigg. Skinning mesh animations. *ACM Transaction on Graphics*, 24:399–407, July 2005.

[33] T. Ju, S. Schaefer, and J. Warren. Mean value coordinates for closed triangular meshes. *ACM Transaction on Graphics*, 24(3):561–566, Jul. 2005.

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

[35] L. Kavan, S. Collins, J. Žára, and C. O'Sullivan. Geometric skinning with approximate dual quaternion blending. *ACM Transaction on Graphics*, 27:105:1–105:23, Nov. 2008.

[36] L. Kavan, R. McDonnell, S. Dobbyn, J. Žára, and C. O'Sullivan. Skinning arbitrary deformations. In *I3D'07: Proceedings SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 53–60, 2007.

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

[38] L. Kavan and J. Žára. Spherical blend skinning: a real-time deformation of articulated models. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, I3D '05, pages 9–16, 2005.

[39] B.-U. Kim, W.-W. Feng, and Y. Yu. Real-time data driven deformation with affine bones. *Vis. Comput.*, 26(6-8):487–495, June 2010.

[40] A. G. Kirk, J. F. O'Brien, and D. A. Forsyth. Skeletal parameter estimation from optical motion capture data. In *IEEE CVPR*, pages 782–788, 2005.

[41] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, Feb. 1956.

[42] P. G. Kry, D. L. James, and D. K. Pai. Eigenskin: real time large deformation character skinning in hardware. In *SCA'02: Proceedings of Symposium on Computer Animation*, pages 153–159, 2002.

[43] H. W. Kuhn and A. W. Tucker. *Nonlinear programming*. University of California Press, 1951.

[44] E. Landreneau and S. Schaefer. Poisson-based weight reduction of animated meshes. *Comput. on Graphics Forum*, 29(6):1945–1954, 2010.

[45] B. H. Le and Z. Deng. Smooth skinning decomposition with rigid bones. *ACM Transaction on Graphics*, 31(6):199:1–199:10, Nov. 2012.

[46] B. H. Le and Z. Deng. Two-layer sparse compression of dense-weight blend skinning. *ACM Transaction on Graphics*, 32(4):124:1–124:10, Jul. 2013.

[47] H. Lee, A. Battle, R. Raina, and A. Y. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems 19*, pages 801–808. MIT Press, 2007.

[48] S.-H. Lee, E. Sifakis, and D. Terzopoulos. Comprehensive biomechanical modeling and simulation of the upper body. *ACM Transaction on Graphics*, 28(4):99:1–99:17, Sep. 2009.

[49] J. Lewis and K.-i. Anjyo. Direct manipulation blendshapes. *Computer Graphics and Applications, IEEE*, 30(4):42 –50, july-aug. 2010.

[50] J. P. Lewis, M. Cordner, and N. Fong. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of SIGGRAPH'00*, pages 165–172, 2000.

[51] Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *IEEE Transaction on Communication*, 28(1):84–95, 1980.

[52] Y. Livny, F. Yan, M. Olson, B. Chen, H. Zhang, and J. El-Sana. Automatic reconstruction of tree skeletal structures from point clouds. *ACM Transaction on Graphics*, 29(6):151:1–151:8, Dec. 2010.

[53] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *J. Mach. Learn. Res.*, 11:19–60, Mar. 2010.

[54] S. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *Transaction Sig. Proceedings*, 41(12):3397–3415, Dec. 1993.

[55] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, 11(2):431–441, 1963.

[56] A. McAdams, Y. Zhu, A. Selle, M. Empey, R. Tamstorf, J. Teran, and E. Sifakis. Efficient elasticity for character skinning with contact and collisions. *ACM Transaction on Graphics*, 30(4):37:1–37:12, Jul. 2011.

[57] B. Merry, P. Marais, and J. Gain. Animation space: A truly linear framework for character animation. *ACM Transaction on Graphics*, 25:1400–1423, Oct. 2006.

[58] A. Mohr and M. Gleicher. Building efficient, accurate character skins from examples. *ACM Transaction on Graphics*, 22:562–568, Jul. 2003.

[59] M. Müller, B. Heidelberger, M. Teschner, and M. Gross. Meshless deformations based on shape matching. *ACM Transaction on Graphics*, 24(3):471–478, Jul. 2005.

[60] J. Nocedal and S. Wright. *Numerical Optimization*. Springer New York, 2000.

[61] S. I. Park and J. K. Hodgins. Capturing and animating skin deformation in human motion. *ACM Transaction on Graphics*, 25:881–889, Jul. 2006.

[62] T. Patterson, N. Mitchell, and E. Sifakis. Simulation of complex nonlinear elastic bodies using lattice deformers. *ACM Transaction on Graphics*, 31(6):197:1–197:10, Nov. 2012.

[63] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[64] M. Sattler, R. Sarlette, and R. Klein. Simple and efficient compression of animation sequences. In *SCA'05: Proceedings of 2005 Symposium on Computer Animation*, pages 209–217, 2005.

[65] S. Schaefer, T. McPhail, and J. Warren. Image deformation using moving least squares. *ACM Transaction on Graphics*, 25:533–540, Jul. 2006.

[66] S. Schaefer and C. Yuksel. Example-based skeleton extraction. In *SGP'07*, pages 153–162, 2007.

[67] T. W. Sederberg and S. R. Parry. Free-form deformation of solid geometric models. *SIGGRAPH Computer on Graphics*, 20(4):151–160, Aug. 1986.

[68] K. Shoemake. Animating rotation with quaternion curves. *SIGGRAPH Computer on Graphics*, 19(3):245–254, Jul. 1985.

[69] E. Sifakis, I. Neverov, and R. Fedkiw. Automatic determination of facial muscle activations from sparse motion capture marker data. *ACM Transaction on Graphics*, 24(3):417–425, Jul. 2005.

[70] C. Stoll, J. Gall, E. de Aguiar, S. Thrun, and C. Theobalt. Video-based reconstruction of animatable human characters. *ACM Transaction on Graphics*, 29(6):139:1–139:10, Dec. 2010.

[71] R. W. Sumner and J. Popović. Deformation transfer for triangle meshes. *ACM Transaction on Graphics*, 23:399–405, Aug. 2004.

[72] A. Tagliasacchi, H. Zhang, and D. Cohen-Or. Curve skeleton extraction from incomplete point cloud. *ACM Transaction on Graphics*, 28(3):71:1–71:9, Jul. 2009.

[73] Utah. The Utah 3D animation repository, 2013.

[74] D. Vlasic, I. Baran, W. Matusik, and J. Popović. Articulated mesh animation from multi-view silhouettes. *ACM Transaction on Graphics*, 27:97:1–97:9, Aug. 2008.

[75] D. Vlasic, P. Peers, I. Baran, P. Debevec, J. Popović, S. Rusinkiewicz, and W. Matusik. Dynamic shape capture using multi-view photometric stereo. *ACM Transaction on Graphics*, 28(5):174:1–174:11, Dec. 2009.

[76] R. Y. Wang, K. Pulli, and J. Popović. Real-time enveloping with rotational regression. *ACM Transaction on Graphics*, 26:73:1–73:9, Jul. 2007.

[77] X. C. Wang and C. Phillips. Multi-weight enveloping: least-squares approximation techniques for skin animation. In *SCA'02: Proceedings of 2002 Symposium on Computer Animation*, pages 129–138, 2002.

[78] L. Zelnik-manor and P. Perona. Self-tuning spectral clustering. In *Advances in Neural Information Processing Systems 17*, pages 1601–1608. MIT Press, 2004.

[79] Y. Zhang. User's guide for yall1: Your algorithms for l1 optimization. Technical report, Rice University, May 2009.

[80] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, 67:301–320, 2005.