EXAMPLE BASED RETARGETING HUMAN MOTION TO ARBITRARY MESH MODELS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

İlker O. Yaz

January, 2013

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Tolga K. Çapın(Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Uğur Güdükbay

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Veysi İşler

Approved for the Graduate School of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Graduate School

ABSTRACT

EXAMPLE BASED RETARGETING HUMAN MOTION TO ARBITRARY MESH MODELS

İlker O. Yaz
M.S. in Computer Engineering
Supervisor: Assist. Prof. Dr. Tolga K. Çapın
January, 2013

Animation of mesh models can be accomplished in many ways, including character animation with skinned skeletons, deformable models, or physic-based simulation. Generating animations with all of these techniques is time consuming and laborious for novice users; however adapting already available wide-range human motion capture data might simplify the process significantly. This thesis presents a method for retargeting human motion to arbitrary 3D mesh models with as little user interaction as possible. Traditional motion retargeting systems try to preserve original motion as is, while satisfying several motion constraints. In our approach, we use a few pose-to-pose examples provided by the user to extract desired semantics behind retargeting process by not limiting the transfer to be only literal. Hence, mesh models, which have different structures and/or motion semantics from humanoid skeleton, become possible targets. Also considering mesh models which are widely available and without any additional structure (e.g. skeleton), our method does not require such a structure by providing a build-in surface-based deformation system. Since deformation for animation purpose can require more than rigid behaviour, we augment existing rigid deformation approaches to provide volume preserving and cartoon-like deformation. For demonstrating results of our approach, we retarget several motion capture data to three well-known models, and also investigate how automatic retargeting methods developed considering humanoid models work on our models.

Keywords: Mesh deformation, volume preservation, animation, motion retargeting.

ÖZET

İNSAN HAREKETİNİN KEYFİ ÖRGÜ MODELLERİNE ÖRNEK TABANLI AKTARIMI

İlker O. Yaz
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Yrd. Doç. Dr. Tolga K. Çapın
Ocak, 2013

Örgü modellerinin animasyonu sırasında, iskeletlendirilmiş model, deform olabilen model, veya fizik tabanlı simulasyon gibi bir çok teknik kullanılabilir. Tüm bu teknikler ile animasyon oluşturma işlemi zaman alıcı ve acemi kullanıcılar için zahmetli olmakla birlikte; halihazırda mevcut insan hareket yakalama verilerinin uyarlanması, süreci önemli ölçüde kolaylaştırabilir. Bu tezde, mümkün olduğunca az kullanıcı etkileşimi ile, insan hareketi verilerinin keyfi 3B örgü modellerine uyarlanması/aktarılması için bir yöntem sunulmaktadır. Daha önce geliştirilmiş hareket aktarma sistemleri, verilen çeşitli hareket kısıtlarını sağlayarak, orijinal hareketi olduğu gibi korumaya çalışır. Bizim yaklaşımımızda, kullanıcı tarafından sağlanan birkaç poza-poz örnek kullanılarak, aktarma sürecinin arkasındaki mantıksal yapının bulunması amaçlanmaktadır. Böylece insan iskeletinden farklı yapıya ve/veya hareket mantığına sahip olan modeller olası girdiler haline getirilmektedir. Ayrıca yaygın olarak bulunan ve herhangi bir ek yapı (örn. iskelet) içermeyen örgü modelleri düşünülerek, dahili yüzey tabanlı deformasyon sistemi sunulmaktadır. Bu deformasyon sisteminde, animasyon amaçlı deformasyonlar katı davranıştan daha fazlasını gerektirebileceği için, hacim korumalı ve çizgi-film benzeri sonuçların elde edilmesini sağlayabilecek bir teknik önerilmektedir. Geliştirilen yöntemin nasıl çalıştığını göstermek için, çeşitli hareket yakalama verileri üç adet tanınmış modele aktarılmıştır. Ek olarak, sadece insansı modelleri hedefleyen otomatik hareket aktarma tekniklerinin bizim modellerimiz üzerinde nasıl çalıştıkları da gösterilmiştir.

Anahtar sözcükler: Örgü deformasyonu, hacim korunumu, animasyon, hareket aktarımı.

Acknowledgement

I would like to thank to my advisor, Assist. Prof. Dr. Tolga K. Çapın for his guidance and help, also Assoc. Prof. Dr. Uğur Güdükbay and Assoc. Prof. Dr. Veysi İşler for reviewing this thesis.

My special thanks goes to Ali Fuat Ünal who graciously accepted to be guarantor for my scholarship, without knowing me or having any relations. Sincere thanks to my friends, and deepest gratitudes to my parents.

I thank to the Scientific and Technological Research Council of Turkey (TÜBİTAK) for providing financial assistance during my study, through the BİDEB program and the project TÜBİTAK 112E105.

The motion capture data used in this project was obtained from mocap.cs.cmu.edu. The database was created with funding from NSF EIA-0196217.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent advances in modeling and deformation, in parallel with motion capture technologies, have made the creation and animation of virtual humans a common task. Animation of virtual human meshes can be accomplished in many ways, including rigging, keyframing and inbetweening deformable models, or physics-based simulations. However, these processes require expert knowledge or prior training about these animation techniques, and they are tedious to use for non-expert users.

As demonstrated by the increasing popularity of player-generated art assets in video games, there is a growing need for animation tools designed for novice users, which allow easy creation of animations. For example, it is desirable to load an arbitrary 3D mesh model and animate it directly with a simple user interface. This thesis addresses this problem with a novel method for generating animations of arbitrary 3D mesh models from human motion capture data, without the need for prior user training.

While skeletal deformation techniques remain the most common way to animate mesh models, these require extra structural definitions (e.g. skeleton or cage) for the input mesh. Our goal is to relieve the user from constructing such a structure. Hence, our system follows a surface-based deformation approach. In particular, we rely on the principles of as-rigid-as possible mesh deformation

Figure 1.1: A traditional motion retargeting example. For a trained animator, preserving abstract qualities of the motion in retargeting process can be a trivial issue. However, achieving the same result with an automatic animation tool has challenges [1].

[4], with the purpose of preserving the shape of the mesh while satisfying given deformation constraints. We extend this deformation framework to also consider cartoon-like animations, by relieving the rigidity constraints to a certain extent. Thus, we aim for achieving deformation results that can be categorized somewhere between rigid and cartoon deformation.

Another key feature of our system is that it provides an example-based retargeting approach, which is capable of animating non-humanoid mesh models that are topologically different from human models. In our approach, the user only provides a few correspondent poses of the humanoid skeleton and the mesh model with a simple interface. These poses are then used to construct a mapping function between the source skeleton and the target mesh by our spacetime solver, which is also capable of handling kinematic and temporal constrains in the retargeting step.

In a nutshell, our work combines spacetime formulation of animation with shape preserving deformation. This hybrid approach extends existing shape preserving deformation methods in order to provide volume preserving and not totally rigid but more cartoon-like deformations. Our method also improves the traditional spacetime retargeting approach by making it compatible with example-based retargeting.

1.1 Contribution

In this work, we present a system for animating 3D mesh models with human motion data. Our main contributions can be listed as follows:

- **An example-based retargeting system.** Considering previous studies, there are many approaches where the main purpose is retargeting motions from one humanoid model to another. In our work, the main goal is to target arbitrary mesh models which might have different structures and/or motion semantics from humanoid models. To cope with differences between source and target structures, our system requires a few pose-to-pose examples to exploit relations between input structures. These relations are then used for implementing a mapping/retargeting function which takes a human motion as a sequence of poses and produces mesh animation by mapping each human-pose to a mesh pose. Also our mapping algorithm can employ user or system specified motion constraints.

- **Cartoon-like and volume preserving deformation.** Our system does not require a skeleton structure for manipulating the input mesh models, by directly working on the surface of the mesh. For surface-based deformation, previous studies focus on providing as-rigid-as possible deformation which mimics the physical deformation of an object in real world. In our system, we provide a deformation method which is not truly as-rigid-as possible but capable of generating squash and stretch effects which are among the key features of an expressive animation. Our deformation also simulates volume preservation while not increasing computational complexity significantly.

- **A simple interface for retargeting.** Since our method directly works on the surface of the mesh without requiring extra structure, the user is only obligated to select control points on the mesh and deform it by dragging these points. This interaction requires no or very brief prior training, which makes the system appropriate for novice users.

1.2 Outline

This thesis is structured as follows. In Chapter 2, a review of important concepts and previous studies for our work is presented. This review includes Laplacian representation of mesh structures, naive Laplacian deformation and as-rigid-as possible deformation approaches, skeleton-based and other types of motion retargeting methods. Chapter 3 describes our approach for retargeting human motion to arbitrary 3D mesh models in three parts. First, overview of our method is presented, where we also describe required user interactions. Second, the specialized deformation system for animation purpose is presented. Then, our spacetime example-based retargeting system is explained. Chapter 4 provides results of our system, discussion of encountered restrictions, and comparison with other approaches. Finally, Chapter 5 concludes the thesis by summarizing the work and possible future extensions.

Chapter 2

Background

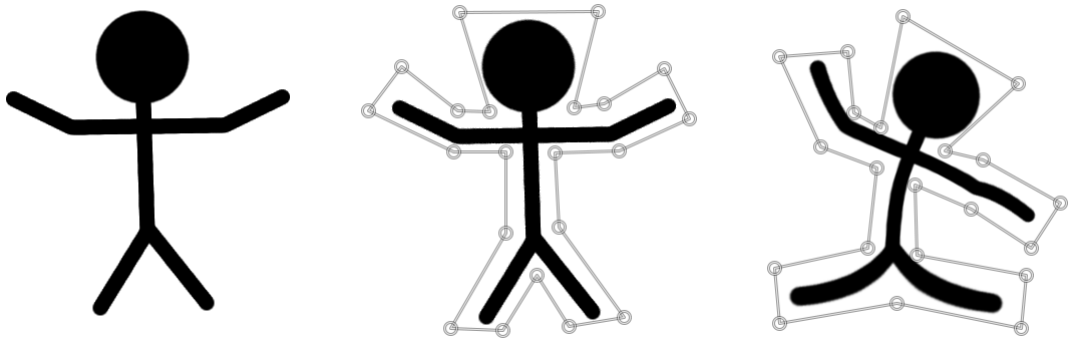
In this chapter, we first review mesh deformation approaches with focus on Laplacian representation and rigid deformation. We provide background related technical details at length for Section 2.1, since our deformation system relies on these foundations. In Section 2.2, motion retargeting methods are reviewed while trying to justify our choice on example-based retargeting approach.

2.1 Mesh Deformation

While skeletal deformation remains the most common way to animate mesh models, alternative surface-based and space-based methods exist for mesh deformation.

For surface-based methods, the main purpose is preserving the shape of the mesh while satisfying given deformation constraints. The shape of the mesh is represented by using variations of differential coordinate representation [7] [8], where the aim is making the representation invariant to editing operations (e.g. translation, rotation) as much as possible.

Surface-based deformations can also be used in animation. Zhou et al. [9] propose a volumetric representation together with differential coordinates of the



(a) (b) (c)

Figure 2.1: Manipulation example on a simple 2D model using implementation of Green Coordinates [2] in GIMP [3]. (a) is the original 2D image, (b) constructed cage around the image, (c) deformation result after dragging control vertices on the cage.

mesh while applying deformation energy minimization, in order to provide both surface detail and volume preserving deformation. In their work, motions from 2D cartoons are retargeted using curves as control structures, which provide geometric correspondences between a mesh and a 2D cartoon.

Among space-based methods, Joshi et al. [10] and Lipman et al. [2] use a cage-based representation for character animation and manipulation of mesh models. These approaches are applicable to generic mesh models, where deformation of the mesh is driven by a surrounding cage, with the deformed cage determining the resulting mesh regarding preservation of general shape and local details. The main disadvantage of these methods, for our problem, is having a carefully constructed cage is a prerequisite for high quality deformations.

Remaining of this section mainly focuses on surface-based deformation systems, since no additional structure is required in deformation process and satisfactory results can be achieved with acceptable computational performance.

2.1.1 Laplacian Representation

Mesh structure is a very common way to represent 3D models. Basically, it is a set of connected vertices where connections are defined by edges and/or facets. The main responsibility of a vertex is holding its position as a global coordinate with respect to a predefined origin. A survey on different types of data structures for representing meshes can be found in Blandford et al. [11].

Laplacian coordinates are a way of representing the mesh differentially, which use the local coordinate of a vertex with respect to its one-ring neighbors (i.e. cell), rather than its global position. The Laplacian coordinate of a vertex is:

$$L(\mathbf{v}_i) = \mathbf{v}_i - \sum_{\mathbf{v}_j \in N(\mathbf{v}_i)} w_{ij}\mathbf{v}_j, \tag{2.1}$$

where $\mathbf{v}_i$ is the current vertex, $N(\mathbf{v}_i)$ is the set of one-ring neighbor vertices of $\mathbf{v}_i$, and w_{ij} is a weight between $\mathbf{v}_i$ and neighbor $\mathbf{v}_j$. A simple choice for weighting schema can be uniform weights where each neighbour vertex is weighted by $1/|N(\mathbf{v}_i)|$, which actually corresponds to calculating the average of neighbour vertices and subtracting the result from the current vertex. Figure 2.2 illustrates the calculation of Laplacian coordinate on a simple mesh structure using uniform weights. Cotangent weights [12] can be used in order to eliminate the effect of non-uniform discretization occurred during the creation of the mesh model.

Considering the mesh as a whole, it is possible to generate a linear system which results in Laplacian coordinates of all vertices:

$$\mathbf{LV} = \delta, \tag{2.2}$$

where n is number of vertices in the mesh, $\mathbf{L}$ is an $n \times n$ square matrix, called Laplacian matrix, in which each row is filled according to Eq. 2.1; $\mathbf{V}$ is an $n \times 1$ column vector consisting of global position of vertices; and δ represents Laplacian coordinates of the mesh.

After constructing Laplacian matrix for a given mesh, we can directly use

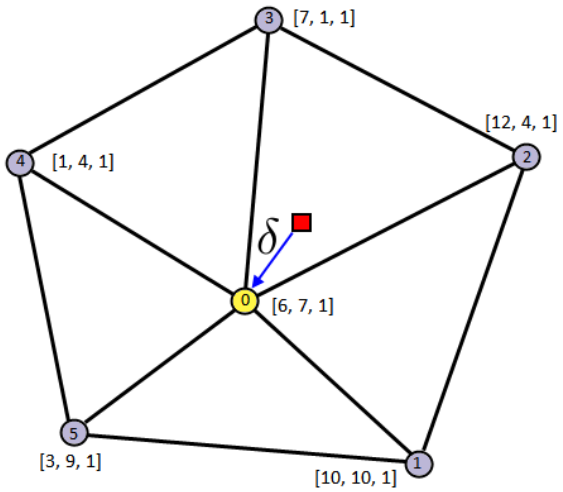


Figure 2.2: A simple mesh structure. Red square is the average of neighbours of the yellow vertex. Blue vector represents Laplacian coordinate of the yellow vertex.

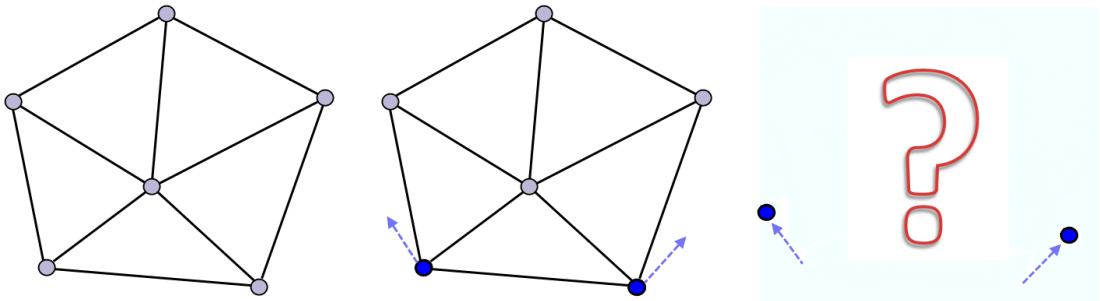
Equation 2.2 for each axis to calculate Laplacian coordinates of the mesh. As an example, using Laplacian matrix ($\mathbf{L}_s$) of the mesh in Figure 2.2 to calculate Laplacian coordinates (δ) in x-axis results in:

$$\begin{bmatrix} 1 & -0.2 & -0.2 & -0.2 & -0.2 & -0.2 \\ -0.33 & 1 & -0.33 & 0 & 0 & -0.33 \\ -0.33 & -0.33 & 1 & -0.33 & 0 & 0 \\ -0.33 & 0 & -0.33 & 1 & -0.33 & 0 \\ -0.33 & 0 & 0 & -0.33 & 1 & -0.33 \\ -0.33 & -0.33 & 0 & 0 & -0.33 & 1 \end{bmatrix} * \begin{bmatrix} 6 \\ 10 \\ 12 \\ 7 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} -0.6 \\ 3.0 \\ 4.3 \\ 0.6 \\ -4.3 \\ -2.6 \end{bmatrix}. \quad (2.3)$$

Similarly, the same process can be repeated for y-axis and z-axis to obtain Laplacian coordinates in all axes (i.e. dimensions).

2.1.2 Naive Laplacian Deformation

A simple deformation system can be provided by allowing the user to select a few vertices on the mesh and simply drag them to deform the mesh. Constrained



(a) (b) (c)

Figure 2.3: A simple deformation system. (a) Original mesh model, (b) selected control points on the mesh by the user, (c) deformed mesh is the question which should be answered by deformation system after dragging blue control points.

vertices are called *control points/vertices*, and after dragging session their positions determines *deformation constraints*. Figure 2.3 shows essential steps of the deformation process.

The main idea behind Laplacian deformation is preservation of Laplacian coordinates under deformation constraints. In other words, Laplacian coordinates are treated as a representative form for the shape of the mesh, and deformation process should satisfy given constraints while causing as little distortion as possible on Laplacian coordinates.

2.1.2.1 Deformation Constraints as Soft Constraints

The mentioned idea can be formulated by a energy function which both considers Laplacian coordinates of the mesh and provided deformation constraints:

$$E(\mathbf{V}') = \sum_{i=1}^{n} \|\delta_i - \delta_i'\|^2 + \lambda \sum_{\mathbf{v}_j \in \mathbf{C}} \left\|\mathbf{c}_j - \mathbf{v}_j'\right\|^2, \tag{2.4}$$

where δ_i is the Laplacian coordinate of the vertex $\mathbf{v}_i$; $\mathbf{v}_i'$ is the global position of the vertex after deformation; δ_i' is the Laplacian coordinate of the vertex after deformation, $\mathbf{C}$ is set of constrained vertices, and $\mathbf{c}_j$ is the position of the constraint for constrained vertex $\mathbf{v}_j$; λ is a weight for deformation constraints.

As clearly seen, the first part of the equation aims to minimize the differences between Laplacian coordinates of the original mesh and Laplacian coordinates of the deformed mesh. The second part of the equation is for minimizing the differences between positions of deformation constraints and positions of constrained vertices in the deformed mesh. The proposed energy function can be expressed as a linear system of:

$$\begin{bmatrix} \mathbf{L} \\ \lambda \mathbf{I}_c \end{bmatrix} \mathbf{V}' = \begin{bmatrix} \delta \\ \lambda \mathbf{C}_j \end{bmatrix}, \tag{2.5}$$

where $\mathbf{I}_j$ is an $m \times n$ matrix for m control points with each row containing 1 in related column with control point; and $\mathbf{C}_j$ contains the positions of deformation constraints. The over-determined system in Eq. 2.5 is solved in a least squares sense to find $\mathbf{V}'$, which preserves the Laplacian coordinates of the mesh (δ) and satisfies the positions of deformation constraints ($\mathbf{C}_j$) as much as possible.

As an example, we can use the mesh in Figure 2.2 by selecting vertices $\mathbf{v}_1$ and $\mathbf{v}_5$ as control points. Assuming deformation constraint positions are $\mathbf{v}_1 = [11, 11, 1]$, $\mathbf{v}_5 = [2, 8, 1]$, and $\lambda = 0.5$, Equation 2.5 results in a matrix system of (considering x-axis):

$$\begin{bmatrix} 1 & -0.2 & -0.2 & -0.2 & -0.2 & -0.2 \\ -0.33 & 1 & -0.33 & 0 & 0 & -0.33 \\ -0.33 & -0.33 & 1 & -0.33 & 0 & 0 \\ -0.33 & 0 & -0.33 & 1 & -0.33 & 0 \\ -0.33 & 0 & 0 & -0.33 & 1 & -0.33 \\ -0.33 & -0.33 & 0 & 0 & -0.33 & 1 \\ 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix} * \mathbf{V}' = \begin{bmatrix} -0.6 \\ 3.0 \\ 4.3 \\ 0.6 \\ -4.3 \\ -2.6 \\ 5.5 \\ 1.0 \end{bmatrix}, \tag{2.6}$$

where the last two rows of the left-hand side matrix are additions for soft constraints, and the last two scalars of right-hand side vector are weighted constraint positions in x-axis. The same left-hand side matrix can be used for y-axis and z-axis by updating the right-hand side vector with appropriate values (i.e. the

left-hand side matrix is independent of the dimension).

2.1.2.2 Deformation Constraints as Hard Constraints

Since the constructed linear system in Section 2.1.2.1 results in a overdetermined system, it can be only solved in a least squares manner. As a result, positions of the deformation constraints are not fully satisfied. However, the user might require a deformation system where positions of the control points can be exactly equal to positions of the deformation constraints after deformation.

Soft constraints can be converted into hard constraints by transferring the related vertex from the left side to the right side in Eq. 2.1, which is equal to multiplying its column with constrained value, adding it to δ, and removing its column from $\mathbf{L}$ in Eq. 2.2. As an example, we can convert soft constraints in Equation 2.6 to hard constraints as follows:

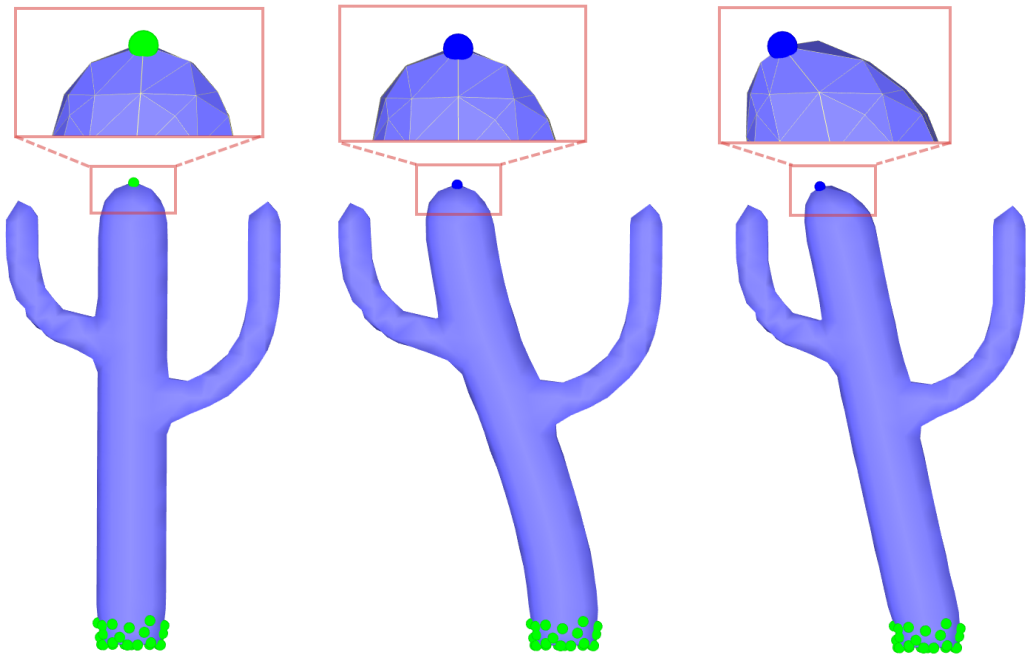


Figure 2.4: Naive Laplacian deformation. (a) Original mesh model with selected control points, (b) deformation result of integrating constraints using Equation 2.7, and (c) using Equation 2.8.

$$\begin{bmatrix} 1 & -0.2 & -0.2 & -0.2 \\ -0.33 & -0.33 & 0 & 0 \\ -0.33 & 1 & -0.33 & 0 \\ -0.33 & -0.33 & 1 & -0.33 \\ -0.33 & 0 & -0.33 & 1 \\ -0.33 & 0 & 0 & -0.33 \end{bmatrix} * \begin{bmatrix} \mathbf{v}_0' \\ \mathbf{v}_2' \\ \mathbf{v}_3' \\ \mathbf{v}_4' \end{bmatrix} = \begin{bmatrix} 2.0 \\ -7.3 \\ 8.0 \\ 0.66 \\ -3.6 \\ -1.0 \end{bmatrix}. \tag{2.7}$$

We should also note that another way to integrate constraints into Laplacian system as hard constraints is directly changing corresponding rows of control points with rows which only contain ones at diagonal positions, and Laplacian coordinates of the control points with constraint positions:

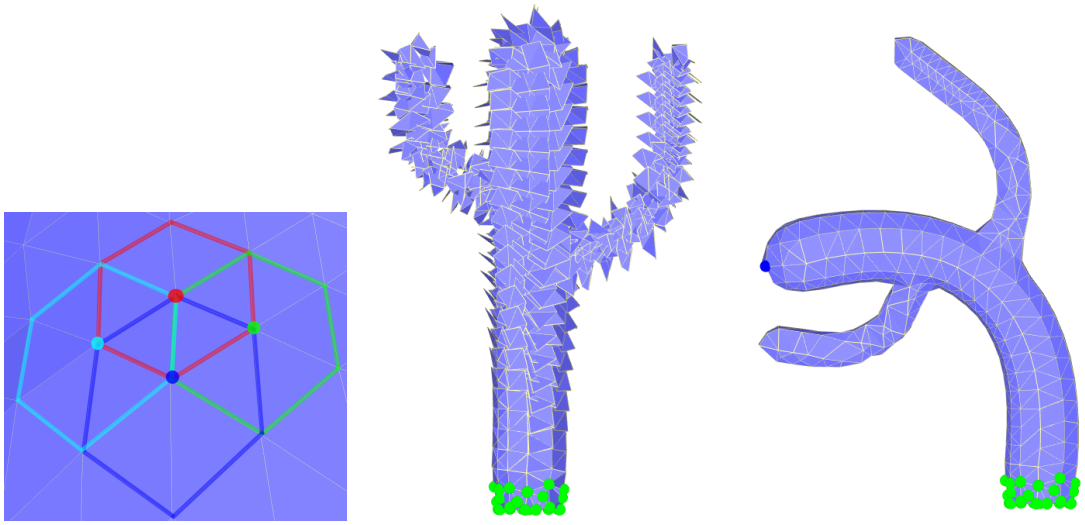
$$\begin{bmatrix} 1 & -0.2 & -0.2 & -0.2 & -0.2 & -0.2 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ -0.33 & -0.33 & 1 & -0.33 & 0 & 0 \\ -0.33 & 0 & -0.33 & 1 & -0.33 & 0 \\ -0.33 & 0 & 0 & -0.33 & 1 & -0.33 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} * \mathbf{V}' = \begin{bmatrix} -0.6 \\ 11.0 \\ 4.3 \\ 0.6 \\ -4.3 \\ 2.0 \end{bmatrix}. \tag{2.8}$$

The difference between constructed systems in Equation 2.7 and Equation 2.8 is that the latter one does not consider Laplacian coordinates of the control points, since we directly remove corresponding rows. As a result, extreme deformation constraints affect the cells of control points more than the mesh globally. In Figure 2.4, one can see that the cell around dragged control point is preserved with Equation 2.7 (Figure 2.4(b)), while Equation 2.8 causes a distortion around the cell (Figure 2.4(c)).

2.1.3 As-Rigid-As Possible Deformation

As previously stated in Section 2.1.2, using Laplacian coordinates is a way of representing the shape of the mesh, and the deformation system is responsible

for preserving the original Laplacian coordinates during the deformation process. However, a good representation of a shape should be invariant to operations which do not affect the shape. Laplacian coordinates fail to satisfy this condition, since rotating the mesh does not affect its shape but changes its Laplacian coordinates. To provide a better representation, Sorkine et al. [4] proposed a deformation energy defined as:



(a) (b) (c)

Figure 2.5: (a) Illustration of overlapping cells, (b) rotating each cell with its optimum rotation without considering overlaps between cells, (c) deformed mesh with considering both rotations and overlaps.

$$\sum_{\mathbf{v}_j \in N(\mathbf{v}_i)} w_{ij} \left\| (\mathbf{v}'_i - \mathbf{v}'_j) - \mathbf{R}_i(\mathbf{v}_i - \mathbf{v}_j) \right\|^2 . \tag{2.9}$$

The intuitive idea behind the proposed energy function is allowing cells to have individual rotations, and at the same time using overlapping nature of the cells to prevent sharing. In Figure 2.5, we provide a illustration of overlapping cells, rotated cells with their optimum rotations, and the final result of as-rigid-as possible deformation.

Minimization of the energy function in Equation 2.9 for every vertex under control point constraints provides as-rigid-as possible deformation. In this equation, there are two unknowns: the new positions of vertices ($\mathbf{v}'$) and rotation

matrices ($\mathbf{R}_i$). This minimization problem is solved by developing a two-step optimization approach. In the first step, the method finds the initial guess for the surface, by solving Eq. 2.5 or using the surface in the previous frame. The initial guess is then used to fill $(\mathbf{v}'_i - \mathbf{v}'_j)$, which leaves $\mathbf{R}_i$ as the only unknown.

Given covariance matrix S_i:

$$\mathbf{S}_i = \sum_{\mathbf{v}_j \in N(\mathbf{v}_i)} w_{ij}(\mathbf{v}_i - \mathbf{v}_j)(\mathbf{v}'_i - \mathbf{v}'_j)^T, \tag{2.10}$$

$\mathbf{R}_i$ which minimizes Eq. 2.9 can be found using SVD of $\mathbf{S}_i$ [1][13]:

$$\mathbf{R}_i = \mathbf{V}_i \mathbf{U}_i^T \; where \; \mathbf{S}_i = \mathbf{U}_i \Sigma_i \mathbf{V}_i^T. \tag{2.11}$$

In the second step, the computed rotation matrices are placed into the partial derivative of Eq. 2.9 with respect to $\mathbf{v}'$, by treating rotation matrices as constants. In order to find $\mathbf{v}'$ which minimizes deformation energy, setting the derivative to zero results in an equation of:

$$\sum_{\mathbf{v}_j \in N(\mathbf{v}_i)} w_{ij}(\mathbf{v}'_i - \mathbf{v}'_j) = \sum_{\mathbf{v}_j \in N(\mathbf{v}_i)} w_{ij} \frac{(\mathbf{R}_i + \mathbf{R}_j)}{2}(\mathbf{v}_i - \mathbf{v}_j). \tag{2.12}$$

One can see that the left-hand side of this equation corresponds to Eq. 2.1, so that we can fill δ using right-hand side of the equation which is all known in our system. After that, Eq. 2.5 or any version of integration of constraints (e.g. hard constrained version) can be solved in order to find $\mathbf{V}'$. Two-step optimization can be pursued for further refinement by altering the first and second steps.

2.1.4 Volume Preservation

Volume preservation is a key principle in animation, and also one of the most important factors of providing plausible deformation results. There are also several approaches in the literature which address this problem.

[1] $det(\mathbf{R}_i)$ should be guaranteed to be positive by changing the sign of eigenvector in $\mathbf{U}_i$ which corresponds to smallest eigenvalue in such case.

Zhou et al. [9] presents an augmented 3D mesh deformation system which relies on the principles of Sorkine et al. [8]. In their work, in order to take volumetric property of the mesh into account, a volume graph which spans the empty space inside the mesh is constructed. This volume graph is treated as a representative of the volume and differential coordinates of graph nodes are combined with differential coordinates of the surface vertices in deformation process. Since deformation process regards both features, surface detail preserving and volume preserving deformation is obtained.

Another method for volume preservation is proposed by Zhang et al. [14] which is quite similar to work of Zhou et al. [9] in terms of reasoning. For representation of the volume, several patches are constructed using a user defined skeleton for the mesh. These patches are perpendicular to the skeleton and connect sampled points from the skeleton to the surface vertices by creating skeleton edges between them. ARAP deformation system [4] is extended where skeleton patches are treated as surface cells by integrating them to the optimization process.

Huang et al. [5] turn deformation problem into a non-linear optimization problem by also introducing several additional constraints such as skeleton constraint, projection constraint, and volume constraint. In this approach, the mesh volume is calculated by using the formulation stated in Zhang et al. [15] and integrated into the optimization process as a hard constraint. Hence, the volume is preserved exactly in contrast to previously mentioned approaches. However, increased computational complexity and required specific solver can be considered as main drawbacks of this technique.

The main purpose of first two mentioned methods [9, 14] can be summarized as trying to keep volume of the mesh during excessive bending and/or twisting. These methods are not suitable for generating desired volumetric effects during squash and stretch, since volumetric representation of the mesh does not change its form for such cases. However, the proposed approach by Huang et al. [5] can provide acceptable results considering squash and stretch effects.

2.2 Motion Retargeting

Although our retargeting method does not requires a skeleton structure, previous works on skeleton-based retargeting still contain valuable information. Hence, in this section, we first review skeleton-based methods and try to address their drawbacks for our problem, and then highlight other example-based methods for retargeting in the latter section.

2.2.1 Skeleton-based Retargeting Methods

One of the earliest approaches for motion retargeting, proposed by Gleicher [16], uses a similar skeleton in motion data with different bone proportions and joint degree of freedoms while rigging the mesh model. The motion from the source skeleton can then be retargeted to the target skeleton by minimizing the change in the motion while preserving motion constraints. This approach is only feasible if the structure of the mesh is similar to the skeleton (e.g. humanoid-like mesh with no topological differences). To overcome this restriction several solutions are proposed. Monzani et al. [17] use an intermediate skeleton while mapping source and target skeletons that have different hierarchies, using an integrated inverse kinematic engine for satisfying motion constraints.

More recent approaches, such as Ikemoto et al. [18] and Yamane et al.'s work [19], construct statistical mapping between the source and target skeletons, by using a set of training data (sequence of poses for the former, pose-to-pose correspondence for the latter) by using different statistical models. Yamane et al. [19] also address the issues of satisfying contact point constraints, improving physical realism, and achieving better global transformation.

Hecker et al. [20] propose an on-line retargeting system which relies on meta-data (e.g. semantic structure, constraints) defined on motions by animators in order to make them morphologically independent. These generic motions then can be retargeted to specific characters with the help of specialized inverse kinematic system at runtime. Compared to mentioned previous studies, their system

allows the animator to directly describe metadata for motions instead of using training data to extract these metadata.

Bharaj et al. [21] retarget skeletal motions to multi-components characters. After creating an appropriate skeleton for a given character, they construct a mapping between joints of the input and created skeleton. Their mapping algorithm is designed to handle different morphologies automatically where neither training data nor metadata are necessary. However, eliminating training leads to the lack of semantic correspondence between the source and target skeletons and their poses, thus limiting retargeting process to direct transfer only.

In any case, the mentioned methods are based on skeleton-to-skeleton retargeting and require a rigged mesh to produce final results. Although this requirement may not be fully achieved automatically, the required user interaction can be simplified by providing a preliminary rough skeleton and skinning results for the input mesh, using automatic skeletonization [22] and skinning[22, 23] techniques. These preliminary results then can be refined by the user manually. However, for a simple interface for retargeting human motion to arbitrary mesh models, it might be desirable to avoid rigging and skinning. Furthermore, the input mesh model might not even have an obvious skeletal structure.

2.2.2 Mesh-based and Shape-based Retargeting Methods

Since skinned skeletons are considered to be the main mechanism for creating animations, most of the retargeting methods try to adapt a motion from one skeleton to another. However, there are other methods which do not focus on skeleton-based retargeting and also try to exploit retargeting semantics by using a training data set (i.e. set of example poses).

Baran et al. [24] propose a method to retarget motion from one mesh to another by learning semantic relationship between the source mesh and the target mesh from pose-to-pose correspondent training data. The key point in their work is developing a shape representation that is suitable for performing two

main operations: measuring mesh similarities, and interpolating weighted meshes. Mesh similarities are used in order to extract weights from training source-poses for a given test source-pose. These weights are then used while interpolating corresponding target-poses to obtain the final result for the test source-pose.

A relatively similar approach is proposed by Bregler et al. [25] while retargeting motion between 2D images. Their system also requires a set of pose-to-pose correspondent training data for the source and the target 2D models. For constructing a given test source-pose, interpolation of training source-poses with proper weights are followed by a proper affine deformation. The same weights are then applied to training target-poses while interpolating and the resulting model is deformed by the same affine deformation to obtain final model for the test source-pose.

Pose-to-pose examples are also used in the area of facial animation retargeting [26, 27]. Pyun et al. [27] propose an approach where examples between source and target face models are used in parametrization and blending (i.e. interpolation) stages. The parametrization stage corresponds to weighting stage in previously mentioned approaches, where for a given test source-face, several parameters are extracted from training source-faces. These parameters work on displacement maps for a number of selected feature points from the source-face. Same parameters are then used on corresponding feature-points from a target-face in blending operation.

Chapter 3

Approach

3.1 Overview

Our method accepts as input a motion sequence and a 3D mesh. The input mesh is to be of a single shell, which is customary with 3D models. The motion sequence can be either keyframed or captured; furthermore the sequence can be either customized for the character of the given mesh, or any generic motion.

Our method consists of two phases (Figure 3.1). The first one is the training phase, where initially the user selects a desired number of joints from the source skeleton and vertices of target mesh with a simple interface. This step allows the user to specify significant joints, which are semantically important for the motion and/or appropriate for the target mesh structure. While there are methods for automatically rigging a mesh with a given skeleton [6], [20], in which the corresponding parts between the source skeleton and the target mesh are identified by the system itself, these methods necessitate a skeleton structure, and further require the structure of the skeleton to be compatible with the mesh.

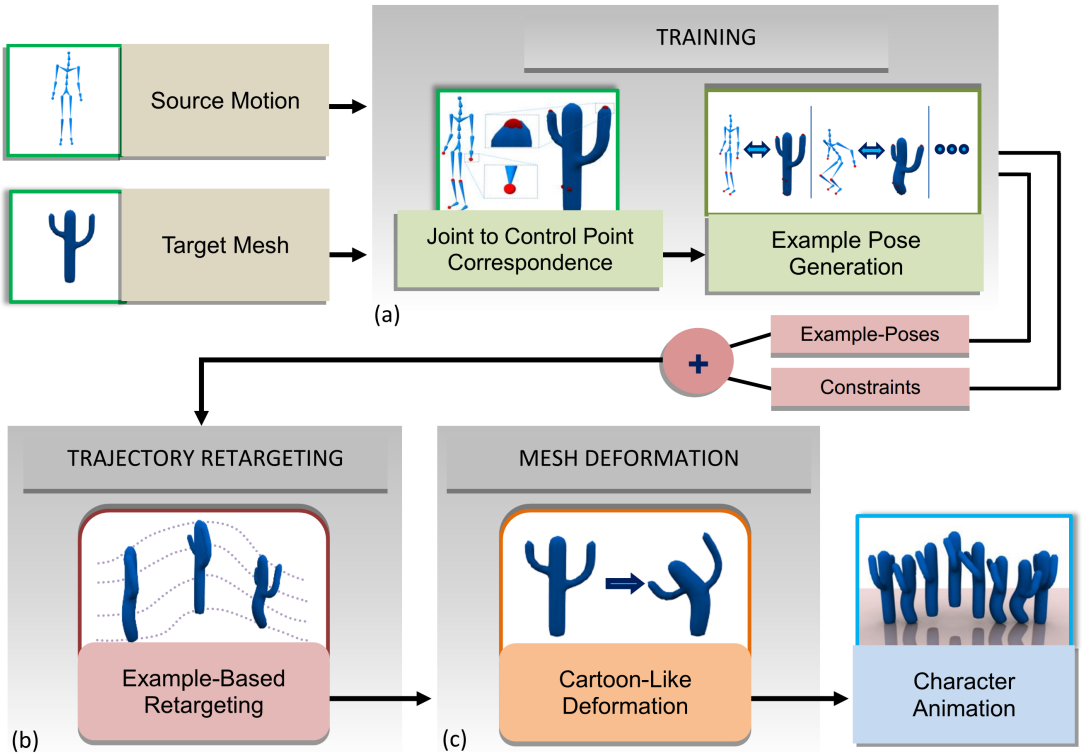


Figure 3.1: Overview of our system. (a) Initially, the user designates significant joints and pairs these with appropriate vertices of the target mesh. The user then provides example pose-to-pose matches selecting key poses from the source motion sequence and using our built-in shape deformation system to create corresponding mesh poses. (b) For each frame in the input motion sequence, corresponding positions of the control points are found by our example-based trajectory retargeting subsystem. This phase may further employ user or system specified constraints. (c) Using the calculated control point positions, our mesh deformation subsystem determines the deformation of the target mesh, and produces final retargeted mesh animation

Consecutively, the user provides pose-to-pose training data, which is used for constructing the mapping function between joints of the skeleton and control points of the mesh. For this purpose, the user selects only a few key poses from the source motion sequence and uses our built-in shape deformation system to create corresponding mesh poses by simply dragging control points. Selection of these example poses is essential to our approach as we pursue a semantical

correspondence between the source skeleton and the target mesh rather than a direct geometrical mapping. Thus, our method conceptually falls in the same category with recent works [19],[24], which also utilize example poses so that the user can specify desired retargeting semantics.

After these simple manual steps, the automatic retargeting phase follows an example-based retargeting approach to reflect the relationship between example poses. In this phase, for each frame in the input motion sequence, we find the corresponding positions of the mesh control points using our example-based trajectory retargeting system. This phase may further contain user or system specified constraints (Section 3.3).

Using the calculated control point positions, our mesh deformation subsystem determines the deformation of the target mesh, and produces final retargeted mesh animation (Section 3.2). For this purpose, our proposed solution extends the as-rigid-as possible surface deformation technique [4] for cartoon-like animation.

Our system uses Cartesian space for representing both humanoid skeleton joint positions and mesh model poses. For transforming the skeleton into the body local coordinate frame, we only cancel yaw angle together with global translation. We use the root joint as the origin of coordinate frame, although any other point can be used for this purpose since our retargeting system is translation invariant. For the mesh model, its center of mass is used as origin for transforming control points, once again the choice of origin is trivial as in the skeleton case.

3.2 Mesh Deformation

Our surface-based mesh deformation system builds on the general-purpose deformation approach, in that the main purpose is preserving the shape of the mesh while satisfying given deformation constraints. However, differently from traditional general-purpose deformation systems, our mesh deformation is aimed for mesh animations. We consider the following to be desirable characteristics of our surface deformation algorithm:

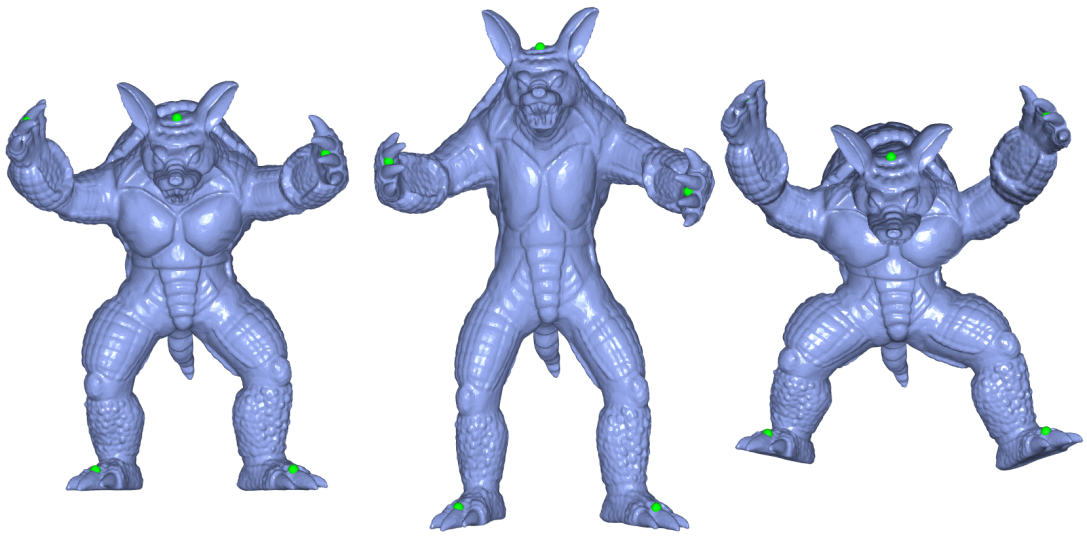


Figure 3.2: Demonstrating our deformation system on Armadillo model using a few control points to interactively deform meshes. Notice the volumetric changes around legs and chest.

1. Mesh deformation should be interactive and allow direct manipulation of the desired deformation, since the user deforms the mesh with the same system while providing example poses.

2. The system should be sparse: selecting only a few control points on the mesh should give acceptable deformation results.

3. The deformation should be as shape preserving as possible, and at the same time allow deviations from rigidity to achieve cartoon-like effects.

4. The deformation should be volume preserving, as a key principle of animation.

Considering the first characteristic, among existing methods in the literature, it might appear more plausible to employ linear deformation methods such as [8, 28, 29] over nonlinear methods [30, 4, 31]. However, linear methods require a large number of control points and/or explicit rotational constraints in order to result in acceptable deformation, which contradicts with the second target characteristic. Assuming that the target mesh is reasonably complex, nonlinear methods are more appropriate to satisfy both the first and second desirable

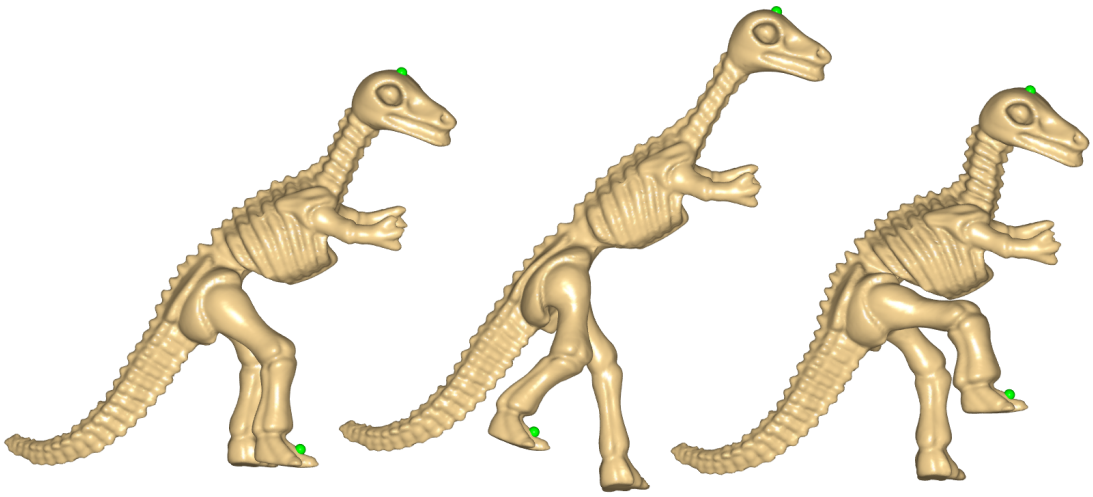


Figure 3.3: Demonstrating our deformation system on Dinosaur model. Notice the volumetric changes around neck and chest.

characteristics. In our approach, we adopt As-Rigid-As Possible (ARAP) surface modeling [4] (Section 2.1.3), which is an iterative non-linear deformation method that is suitable for interactive applications. The ARAP method satisfies our first and second requirements, however it has the drawback that it fails to support the third and fourth desirable characteristics. In the rest of this section, we explain our extensions to this approach.

3.2.1 Cartoon-Like Deformation

In traditional ARAP surface deformation [4], the deformation energy for vertex v_i is defined as:

$$\sum_{\mathbf{v}_j \in N(\mathbf{v}_i)} w_{ij} \left\| (\mathbf{v}'_i - \mathbf{v}'_j) - \mathbf{R}_i(\mathbf{v}_i - \mathbf{v}_j) \right\|^2 . \tag{3.1}$$

Minimization of this equation for every vertex under control point constraints provides as-rigid-as possible deformation. The main characteristic of the traditional ARAP deformation is finding optimum rotations that preserve edge lengths

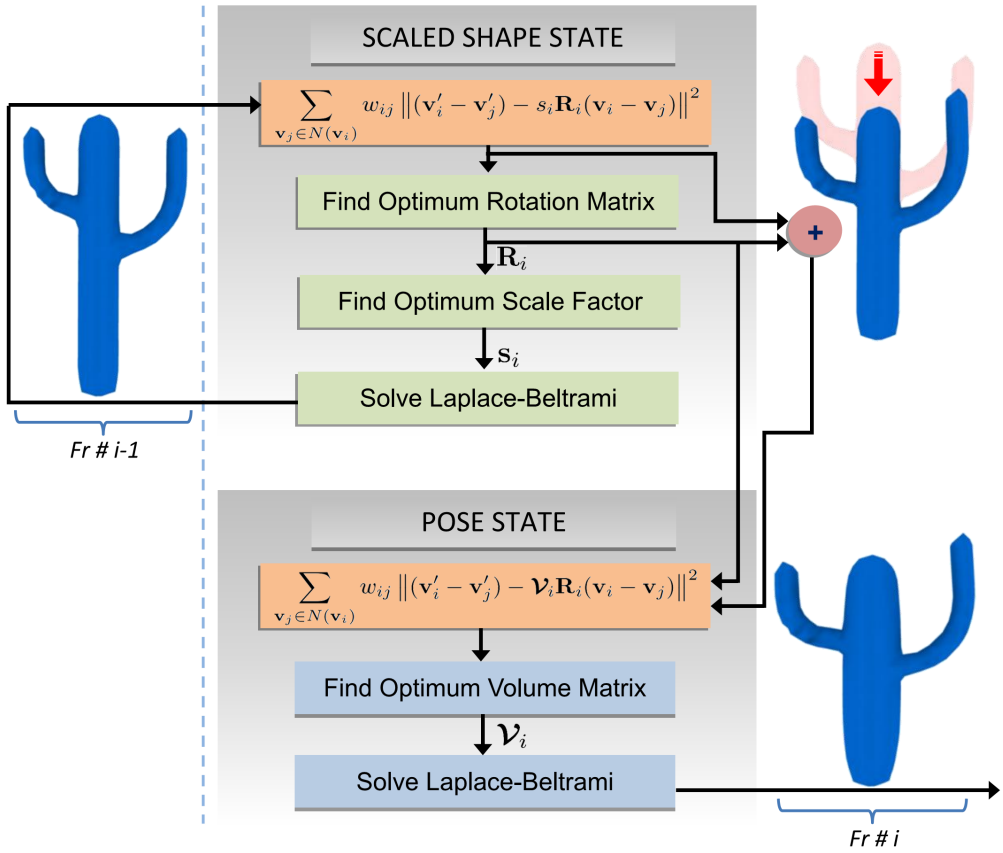


Figure 3.4: Our surface deformation approach consists of two states. In the preliminary scaled shape state, overlapping cells covering the mesh surface are rotated and scaled uniformly in order to satisfy given constraints. The intermediary surface with its cells rotated but not scaled is then used in the pose state in order to achieve the volume preserved end result. While the shape state can be run continuously for finer minimization of the deformation energy function, the pose state is visited only once per frame

under given constraints. However, while animating mesh models, it is more desirable to have a pose which conveys the emotion rather than a pose where local details of the mesh are preserved. In other words, acceptability and cartoon-like features of the global shape of the mesh suppress the preservation of edge lengths. Considering this assumption, we developed an alternative deformation approach which is not truly rigid but cartoon-like.

The principle behind cartoon-like deformation is simple: change edge lengths when it is desired and preserve them otherwise. In our deformation system, we

use a *scaled shape state* where the surface is covered with overlapping cells such that uniform scale coefficients are used for cells along with rotations. Therefore cells can be rotated and scaled uniformly in order to satisfy given constraints. The main contribution of the scale coefficient is making squash and stretch effects possible together with volume preservation, by canceling rotation where only scale can satisfy the given constraints.

Thus, incorporating the scale component into deformation energy (Equation 3.1) yields:

$$\sum_{\mathbf{v}_j \in N(\mathbf{v}_i)} w_{ij} \left\| (\mathbf{v}'_i - \mathbf{v}'_j) - s_i \mathbf{R}_i (\mathbf{v}_i - \mathbf{v}_j) \right\|^2 . \tag{3.2}$$

In this equation, there are three unknowns: the new positions of vertices ($\mathbf{v}'$), the rotation matrix ($\mathbf{R}_i$), and the scale component ($\mathbf{s}_i$). This minimization problem is solved by developing a three-step optimization approach. In the first step, the method finds the initial guess for the surface, by solving Equation 2.5 or using the surface in the previous frame. The initial guess is then used to fill $(\mathbf{v}'_i - \mathbf{v}'_j)$, which leaves $\mathbf{R}_i$ as the only unknown while $\mathbf{s}_i$ is treated as constant. Given covariance matrix $\mathbf{C}_i$:

$$\mathbf{C}_i = \sum_{\mathbf{v}_j \in N(\mathbf{v}_i)} w_{ij} (\mathbf{v}_i - \mathbf{v}_j)(\mathbf{v}'_i - \mathbf{v}'_j)^T, \tag{3.3}$$

$\mathbf{R}_i$ which minimizes Equation 3.1 can be found by applying singular value decomposition (SVD) on $\mathbf{C}_i$ [13]:

$$\mathbf{R}_i = \mathbb{V}_i \mathbb{U}_i^T \; where \; \mathbf{C}_i = \mathbb{U}_i \Sigma_i \mathbb{V}_i^T, \tag{3.4}$$

where $det(\mathbf{R}_i)$ should be guaranteed to be positive by changing sign of eigenvector in $\mathbf{U}_i$ which corresponds to the smallest eigenvalue in such case.

In the second step, by concatenating edge vectors and rewriting deformation energy of a vertex $\mathbf{v}_i$ where $\mathbf{e}'_{ij} = \sqrt{w_{ij}}(\mathbf{v}'_i - \mathbf{v}'_j)$, $\mathbf{e}_{ij} = \sqrt{w_{ij}}(\mathbf{v}_i - \mathbf{v}_j)$ results in:

$$\left\| \begin{bmatrix} \mathbf{e}'_{ij} \\ \mathbf{e}'_{ik} \\ \vdots \end{bmatrix} - s_i \begin{bmatrix} \mathbf{R}_i\mathbf{e}_{ij} \\ \mathbf{R}_i\mathbf{e}_{ik} \\ \vdots \end{bmatrix} \right\|^2 =$$

$$\|\mathbf{e}'_i - s_i{\mathbf{e}^r}_i\|^2 =$$

$$\mathbf{e}_i'^T\mathbf{e}'_i - 2s_i\mathbf{e}_i'^T{\mathbf{e}^r}_i + s_i^2{\mathbf{e}^r}_i^T{\mathbf{e}^r}_i. \tag{3.5}$$

Then, the scale factor for cell i in scaled shape state can be found by taking the partial derivative of Equation 3.5 with respect to s_i and setting it to zero:

$$\frac{\partial}{\partial s_i}\left(\mathbf{e}_i'^T\mathbf{e}'_i - 2s_i\mathbf{e}_i'^T{\mathbf{e}^r}_i + s_i^2{\mathbf{e}^r}_i^T{\mathbf{e}^r}_i\right) = 0$$

$$-2\mathbf{e}_i'^T{\mathbf{e}^r}_i + 2s_i{\mathbf{e}^r}_i^T{\mathbf{e}^r}_i = 0$$

$$s_i = \frac{\mathbf{e}_i'^T{\mathbf{e}^r}_i}{{\mathbf{e}^r}_i^T{\mathbf{e}^r}_i}. \tag{3.6}$$

In the third step, the computed rotation matrices and scale coefficients are placed into the partial derivative of Equation 3.2 with respect to $\mathbf{v}'$, by treating rotation matrices and scale coefficients as constants. In order to find $\mathbf{v}'$ which minimizes deformation energy, setting the derivative to zero results in an equation of:

$$\sum_{\mathbf{v}_j \in N(\mathbf{v}_i)} w_{ij}(\mathbf{v}'_i - \mathbf{v}'_j) = \sum_{\mathbf{v}_j \in N(\mathbf{v}_i)} w_{ij}\frac{(s_i\mathbf{R}_i + s_j\mathbf{R}_j)}{2}(\mathbf{v}_i - \mathbf{v}_j). \tag{3.7}$$

One can see that the left-hand side of this equation corresponds to Equation 2.1, so that we can fill δ using right-hand side of the equation which is all known in our system. After that, Equation 2.5 can be solved in order to find $\mathbf{V}'$. This optimization can be pursued for further refinement by reiterations of the three-step cycle.

After finding rotations and scales, Equation 3.7 is solved by using rotations as well as scales while calculating right-hand side. The resulting surface in the scaled shape state is then used in the *pose state* for achieving volume preservation.

3.2.2 Simulating Volume Preservation

For preserving volume while deforming the mesh, there are several approaches in the literature. Zhou et al. [9] suggested an approach where a volumetric graph of the mesh is constructed and deformation energy function is augmented in order to preserve both volume and surface details approximately. In another approach developed by Huang et al. [5], the volume of the mesh is integrated as a nonlinear hard constraint into the energy function, so that volume can be preserved exactly.

We take another path by trying to provide a solution which is computationally effective and requires no extra volumetric representation. Our solution is inspired by [32], where the volume of the mesh is altered by applying a scale matrix:

$$\mathbf{S}_s = \begin{bmatrix} s & 0 & 0 \\ 0 & \sqrt{1/s} & 0 \\ 0 & 0 & \sqrt{1/s} \end{bmatrix} \tag{3.8}$$

in deformation coordinate system where x-axis as the primary axis that squash-stretch effects take place. Also, in our configuration, we do not apply scale in primary axis, since cells are already scaled because of the deformation (which means scale matrix in Equation 3.8 contains 1 as scale coefficient for the primary axis).

We extend this concept into vertex-level, where each vertex has its own scale matrix. The deformation coordinate system of each vertex is dynamically adjusted by considering the deformation of one-ring neighbours of the vertex. For this adjustment, we find principle components by applying eigenvalue decomposition on covariance matrix of edge differences:

$$\sum_{j \in N(i)} (\mathbf{e}'_{ij} - \mathbf{R}_i \mathbf{e}_{ij})(\mathbf{e}'_{ij} - \mathbf{R}_i \mathbf{e}_{ij})^T = \mathbf{P}_i \Sigma_i {\mathbf{P}_i}^T, \tag{3.9}$$

where $\mathbf{P}_i$ contains eigenvectors that are sorted in decreasing order according to their eigenvalues. The largest eigenvector is used as the primary axis for scale, since it represents the greatest deformation axis for one-ring neighbors. Having $\mathbf{P}_i$ which represents the deformation coordinate system for vertex $\mathbf{v}_i$, we can

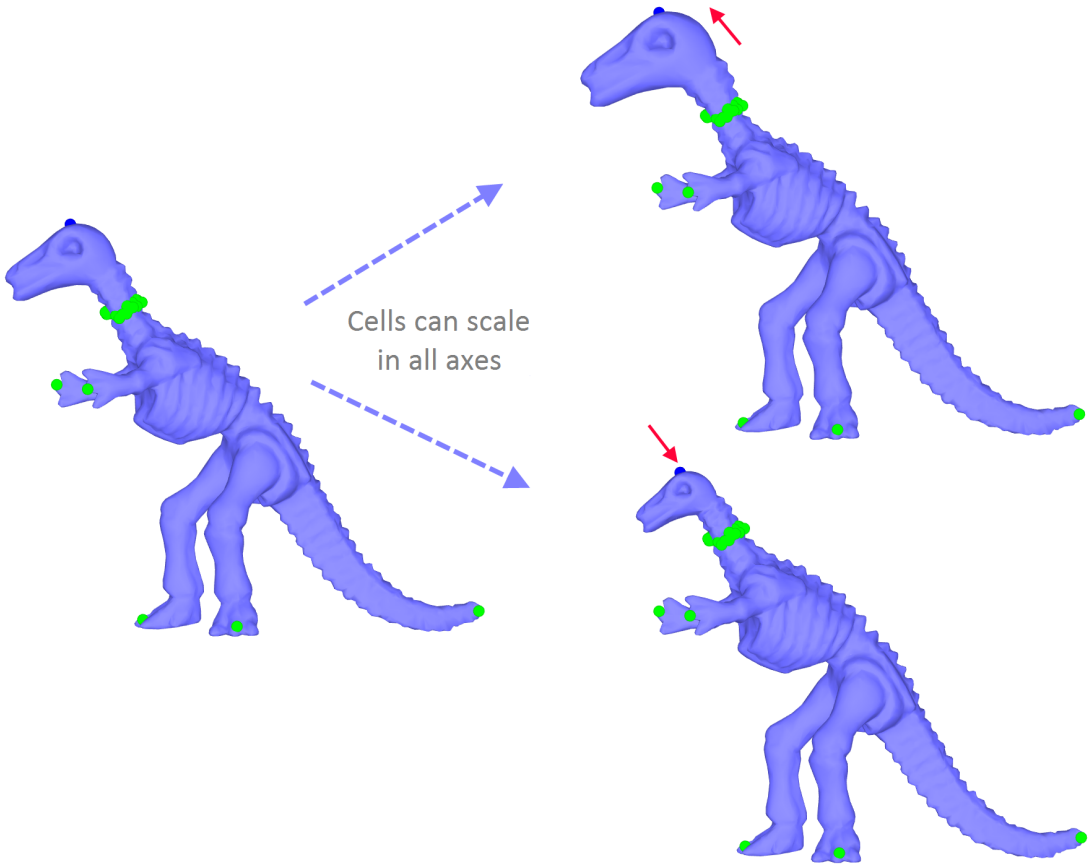


Figure 3.5: Deformation results with both rotation and uniform scale (scale in all axes). Our main idea is restricting scales in meaningful axes with meaningful values, so that simulation of volume preservation can be achieved.

obtain the volume preserving scale matrix $\boldsymbol{\mathcal{V}}_i$ by first projecting on $\mathbf{P}_i$, applying scale matrix $\mathbf{S}_{s_i}$, and re-projecting to original coordinate system:

$$\boldsymbol{\mathcal{V}}_i = \mathbf{P}_i \mathbf{S}_{s_i} \mathbf{P}_i^T, \tag{3.10}$$

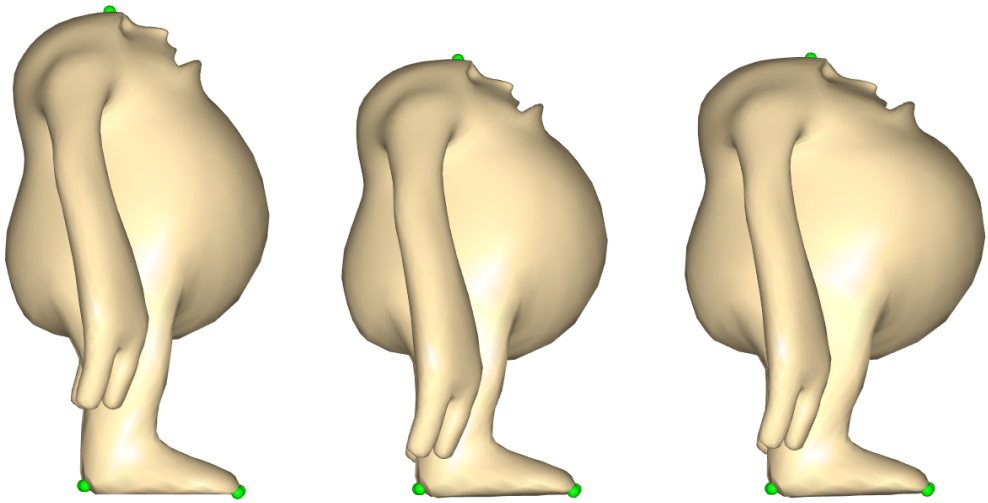
where s_i is the scale coefficient calculated by using Equation3.2.1 with projected edges on the primary axis. Assuming the primary axis is represented by $\mathbf{p_i}$, only change in Equation3.5 is:

$$\left\| \begin{bmatrix} \mathbf{p_i}^T \mathbf{e}'_{ij} \\ \mathbf{p_i}^T \mathbf{e}'_{ik} \\ \vdots \end{bmatrix} - s_i \begin{bmatrix} \mathbf{p_i}^T \mathbf{R}_i \mathbf{e}_{ij} \\ \mathbf{p_i}^T \mathbf{R}_i \mathbf{e}_{ik} \\ \vdots \end{bmatrix} \right\|^2 . \tag{3.11}$$

Our method is also applicable where it is desired to have volumetric changes exaggerated (Figure 3.6). This volumetric effect is easily achieved by increasing or decreasing s_i with a factor λ_v using a simple equation:

$$s_i' = (s_i - 1)\lambda_v + 1. \tag{3.12}$$

Volume preserving scale matrices found in this step are used in the second part of the system (Figure 3.4) which is periodically executed to obtain the pose. In this state, while calculating right-hand side for Equation 3.7, rotations are pre-multiplied by corresponding volume matrices instead of scale coefficients.



(a) (b) (c)

Figure 3.6: (a) is the original Humpty model; (b) deformation result with no volume preservation ($\lambda_v = 0$), and (c) with volume preservation ($\lambda_v = 3$).

3.2.3 Numerical Solution

There are several numerical techniques in order to solve the sparse linear system that Equation 2.5 results in, and a survey can be found in [33] (focuses on surface deformation), and also in [34] (provides introductory level foundations). We follow the proposed approach by [7] where Equation 2.5 is considered as a least squares problem $\min \|\mathbf{Ax} - \mathbf{b}\|^2$, and Cholesky factorization is applied on normal

matrix $\mathbf{A}^T\mathbf{A}$ in normal equation $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$. This factorization can then be used multiple times to solve this equation for different right-hand sides with only the cost of back-substitution. After factorization, we can write $\mathbf{M} = \mathbf{A}^T\mathbf{A} = \mathbf{R}^T\mathbf{R}$ where $\mathbf{R}$ is a upper triangular matrix. The solution for the mentioned system, by using upper triangular matrix and its transpose, can be achieved in following steps:

$$\begin{aligned} \mathbf{A}^T\mathbf{A}\mathbf{x} &= \mathbf{A}^T\mathbf{b} \\ \mathbf{M}\mathbf{x} &= \mathbf{A}^T\mathbf{b} \\ \mathbf{R}^T\mathbf{R}\mathbf{x} &= \mathbf{A}^T\mathbf{b} \\ \mathbf{R}^T\mathbf{x}' &= \mathbf{A}^T\mathbf{b} \\ \mathbf{R}\mathbf{x} &= \mathbf{x}' \end{aligned} \tag{3.13}$$

Since $\mathbf{A}$ is a sparse matrix (hence $\mathbf{A}^T\mathbf{A}$ is), during factorization process, it is desired to keep constructed upper triangular matrix $\mathbf{R}$ as sparse as possible, considering benefits of sparsity in terms of both memory and computational complexity. For this purpose, a factorization process begins with computing a permutation matrix, called $\mathbf{P}$, which is responsible for arranging rows of the parameter matrix (in our case $\mathbf{A}^T\mathbf{A}$) in an order that produced $\mathbf{R}$ using the permuted matrix is sparse. Also another permutation matrix can be used for ordering columns of the parameter matrix. It might be worth to note that, permutation matrix is calculated considering positions non-zero elements since actual values of non-zero elements do not affect the sparsity of the factorization result ($\mathbf{R}$). Demonstrating the effect of permutation matrix, we provide an example in which two versions of a matrix $\mathbf{M}$ are factorized using LU factorization:

$$\mathbf{M} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \mathbf{LU},\ \mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0.5 & 1 & 0 \\ 1 & 1 & 1 & 0.5 & 0.3 & 1 \end{bmatrix}, \tag{3.14}$$

$$\mathbf{M}' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \mathbf{L}'\mathbf{U}',\ \mathbf{L}' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \tag{3.15}$$

where first row and first column of $\mathbf{M}$ are moved to the end while constructing $\mathbf{M}'$. One can see that with reordering $\mathbf{M}$, sparsity of the lower triangular matrix $\mathbf{L}$ can be increased (although we do not use LU factorization, the same principle applies to Cholesky factorization as well).

Here, one should note that adding (or removing) soft constraints into the system only affects the diagonal of the normal matrix $\mathbf{A}^T\mathbf{A}$ which is always non-zero. Assuming $\mathbf{N}$ represents the additional rows for soft constraints where each row contains only one non-zero term (Section 2.1.2.1), the resulting system can be written as:

$$\begin{bmatrix} \mathbf{A} \\ \mathbf{N} \end{bmatrix}^T \begin{bmatrix} \mathbf{A} \\ \mathbf{N} \end{bmatrix} = \begin{bmatrix} \mathbf{A}^T\mathbf{N}^T \end{bmatrix} \begin{bmatrix} \mathbf{A} \\ \mathbf{N} \end{bmatrix} = \mathbf{A}^T\mathbf{A} + \mathbf{N}^T\mathbf{N}, \tag{3.16}$$

where $\mathbf{N}^T\mathbf{N}$ is a diagonal matrix.

So that, while re-factorization, we can skip computation of the permutation matrix, which is used for reducing fill-ins, since non-zero pattern of the matrix is not changed. This improvement becomes important especially in interactive

editing systems where the user frequently adds or removes control points and expects immediate response. Also, we can update the existing factorization after low rank modifications (in case of addition or removal of very a few constrains) [35, 36].

3.3 Trajectory Retargeting

Having aforementioned pose-to-pose examples between the input skeleton and the mesh model, our system is capable of retargeting trajectories in a semantic manner. As a post-processing step, our system also handles generation of suitable global translation.

Using provided examples, our algorithm tries to learn a mapping between a joint and corresponding control point for trajectory retargeting. We consider the following to be desirable characteristics of our mapping function:

1. The mapping function should be translation and rotation invariant (more precisely, applying translation and rotation to example poses and test case should have minimal effect on the final retargeting result).
2. Retargeted trajectories should be smooth (i.e. no jitter and jerkiness).
3. Given a joint example as a test case, the result should be corresponding control point example.
4. The mapping function should be suitable for integrating user or system specified constraints (e.g. positional constraints).

In lights of these requirements, we first consider the alternative approaches for the mapping function, and then propose our approach.

3.3.1 Linear Combination

A simple mapping solution is to employ a linear combination solution. Assume that for a joint $\mathbf{j}$, we have n joint-control point examples $\mathbf{j}^{(1)} \leftrightarrow \mathbf{c}^{(1)}, ..., \mathbf{j}^{(n)} \leftrightarrow \mathbf{c}^{(n)}$. Given an animation frame k, with the position of joint $\mathbf{j}$ as $\mathbf{j}_k$, the mapping function is responsible for finding the corresponding control point location $\mathbf{c}_k$. In this approach $\mathbf{j}_k$ is represented in terms of $\mathbf{j}^{(1)}, ..., \mathbf{j}^{(n)}$ using appropriate weights, which can be found by solving the linear system of:

$$\begin{bmatrix} \begin{bmatrix} \\ \mathbf{j}^{(1)} \\ \\ \end{bmatrix} & \cdots & \begin{bmatrix} \\ \mathbf{j}^{(n)} \\ \\ \end{bmatrix} \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} \\ \mathbf{j}_k \\ \\ \end{bmatrix}, \tag{3.17}$$

so that the computed weights are applied to the corresponding control points $\mathbf{c}^{(1)}, ..., \mathbf{c}^{(n)}$ in order to find $\mathbf{c}_k$:

$$\sum_{i=1}^{n} w_i \begin{bmatrix} \\ \mathbf{c}^{(i)} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \mathbf{c}_k \\ \\ \end{bmatrix}. \tag{3.18}$$

However, this simple projection-interpolation based method is too general to yield acceptable results, and can be restricted by considering requirements as below.

3.3.2 Affine Combination with Bounded Weights

Using an affine combination of poses provides translation invariance, satisfying the first requirement. This technique is employed by Baran et al. [24] by a weighting schema for their example poses that are represented by a vector in shape space.

Note that since weights are not bounded in a range in these approaches, there is the advantage of having no limit for extrapolation. On the other hand, too large weights are not meaningful and may easily lead to unnatural results. Bregler et

al. [25] also put a margin for weights in order to limit extrapolation while finding affine combinations of example shapes in their work.

Bounded weights with affine combination can be found solving a constrained linear system of:

$$\begin{bmatrix} \begin{bmatrix} \mathbf{j}^{(1)} \\ \\ 1 \end{bmatrix} & \cdots & \begin{bmatrix} \mathbf{j}^{(n)} \\ \\ 1 \end{bmatrix} \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} \mathbf{j}_k \\ \\ 1 \end{bmatrix}, \tag{3.19}$$

where $l \leq w_i \leq u$, and weights can be used to find $\mathbf{c}_k$ as in Equation 3.18.

Besides the fact that this approach can only handle the first and second requirements (i.e. translation invariance and smoothness), there is a further shortcoming. If more than four linearly independent examples are provided for a control point, the system becomes under-determined and there is no exact way to select the proper solution among many solutions. To cope with this problem, one can choose four examples for a given frame k which are closest to $\mathbf{j}_k$. Note that although this approach can satisfy the third requirement in this way, it can easily lead to jerkiness on the resulting trajectory since different examples might be used for adjacent frames.

3.3.3 Spacetime Solution

Our spacetime solution addresses the shortcomings of linear and affine combination approaches, and also allows direct mapping and addition of constraints. We also handle possible jitter and jerkiness by introducing an acceleration constraint as a soft constraint.

Considering all frames *1* to F together, the function to be minimized is:

$$\min_{\mathbf{w}} \sum_{i \in F} \left\| \mathbf{J}_i \mathbf{w}_i - \mathbf{j}_i \right\|^2, \tag{3.20}$$

where $\mathbf{J}_i$ represents a 4×4 matrix which is filled with selected example joint positions for frame number i with soft affine constraints (Equation 3.19). The

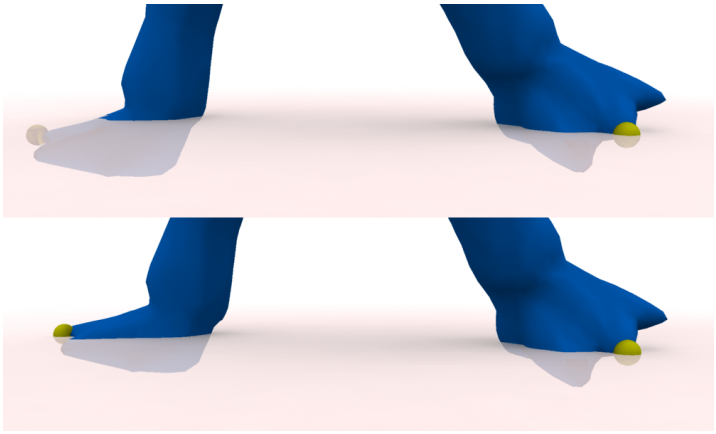


Figure 3.7: Since our retargeting process only relies on training data, there is no way to guarantee that two control points are in suitable positions when contact anchor is transferred one to another (top). We can solve this problem by using equality constraints between control points in y-axis for smooth transition (bottom).

corresponding linear system for this equation can be constructed as:

$$\min_{w} \left\| \mathcal{J}w - j \right\|^2, \tag{3.21}$$

with,

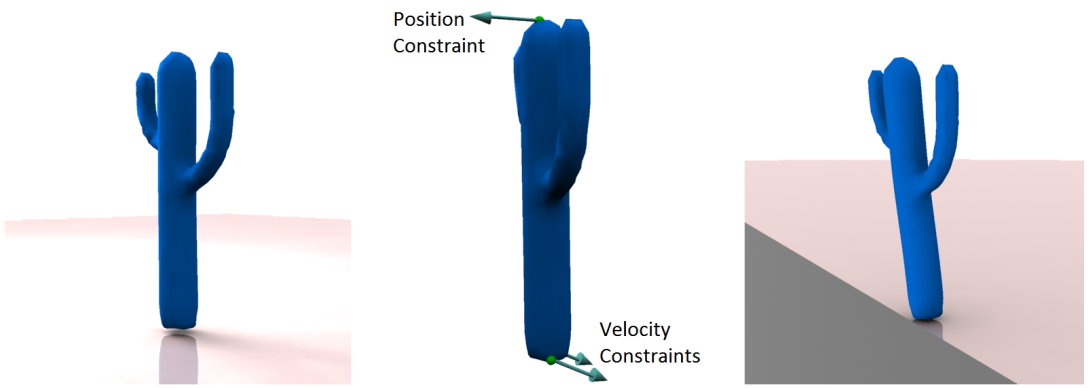
$$\mathcal{J} = \begin{bmatrix} \mathbf{J}_1 & \cdots & \mathbf{0} \\ \vdots & \ddots & \\ \mathbf{0} & & \mathbf{J}_F \end{bmatrix}, w = \begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_F \end{bmatrix}, j = \begin{bmatrix} \mathbf{j}_1 \\ \vdots \\ \mathbf{j}_F \end{bmatrix}. \tag{3.22}$$

In the same manner, we use the example control point positions for finding the mapped trajectory, once the weights are available.

$$\mathcal{C}w = c,\ \mathcal{C} = \begin{bmatrix} \mathbf{C}_1 & \cdots & \mathbf{0} \\ \vdots & \ddots & \\ \mathbf{0} & & \mathbf{C}_F \end{bmatrix}, c = \begin{bmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_F \end{bmatrix}, \tag{3.23}$$

where $\mathbf{C}_i$ represents a matrix which is filled with corresponding example control point positions to selected example joint positions for frame number i, and also c is the result of the trajectory mapping.

Constraints. Our approach also differs from the previous approaches in its solution to integrating constraints into the spacetime system. Since our weights



(a) (b) (c)

Figure 3.8: Position and velocity constraints can be used for editing original retargeting results (jumping moment). (a) Original retargeting result, (b) employed constraints by the user (or the system), (c) retargeting results after constraints.

might belong to different examples for adjacent frames, we can not directly apply temporal or equality constraints on weights. To overcome this problem, we first obtain the mapped trajectory from the calculated weights, and then apply constraints directly on the trajectory:

$$\min_{\mathbf{w}} \sum_{i=2}^{F-1} \left\| \mathbf{C}_{i-1}\mathbf{w}_{i-1} - 2\mathbf{C}_i\mathbf{w}_i + \mathbf{C}_{i+1}\mathbf{w}_{i+1} \right\|^2 . \tag{3.24}$$

We can construct a corresponding linear system for Equation 3.24 to integrate into our mentioned system Equation 3.21:

$$\min_{\boldsymbol{w}} \left\| \mathcal{A}\mathcal{C}\boldsymbol{w} \right\|^2 , \tag{3.25}$$

with acceleration matrix constructed according to Equation 3.24:

$$\mathcal{A} = \lambda_A \begin{bmatrix} 0 & & & & & \\ \mathbf{I} & -2\mathbf{I} & \mathbf{I} & & & \\ & & \ddots & & & \\ & & \mathbf{I} & -2\mathbf{I} & \mathbf{I} & \\ & & & & 0 \end{bmatrix} , \tag{3.26}$$

where λ_A is the weight of acceleration constraint and the only parameter for our trajectory retargeting system.

Regarding linear constraints, we provide support for position and velocity constraints as hard constraints:

$$\mathbf{C}_i\mathbf{w}_i = \mathbf{p}_i, \tag{3.27}$$

$$\mathbf{C}_i\mathbf{w}_i - \mathbf{C}_{i-1}\mathbf{w}_{i-1} = \mathbf{v}_i, \tag{3.28}$$

where the position and velocity of the control point at frame i are constrained to $\mathbf{p}_i$ and $\mathbf{v}_i$ respectively. We can use these constraints to edit original retargeting results as demonstrated in Figure 3.8.

Spacetime system. Integrating positional and velocity constraints as hard constraints and acceleration constraint as soft constraint, we can construct a linear system with equality constraints:

$$\min_{w} \left\| \begin{bmatrix} \mathcal{J} \\ \mathcal{AC} \end{bmatrix} \mathcal{w} - \begin{bmatrix} j \\ \mathbf{0} \end{bmatrix} \right\|^2, \ \mathcal{HC}\mathcal{w} = \mathcal{h}, \tag{3.29}$$

where each position and velocity constraint represents a row in matrix $\mathcal{H}$, and each associated constraint value is placed under $\mathcal{h}$. Since our formulation only requires equality constraints, we can use Lagrange multipliers in order to obtain a single linear system of:

$$\begin{bmatrix} \begin{bmatrix} \mathcal{J} \\ \mathcal{AC} \end{bmatrix}^T \begin{bmatrix} \mathcal{J} \\ \mathcal{AC} \end{bmatrix} & \mathcal{C}^T\mathcal{H}^T \\ \mathcal{HC} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathcal{w} \\ \lambda \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} \mathcal{J} \\ \mathcal{AC} \end{bmatrix}^T \begin{bmatrix} j \\ \mathbf{0} \end{bmatrix} \\ \mathcal{h} \end{bmatrix}. \tag{3.30}$$

Although we do not bound weights in a range in our current system, this can easily be achieved by solving Equation 3.29 with extra inequality constraints along with equality constraints, using quadratic programming for solution [37].

We can also combine systems generated for different control points into one system, and introduce constraints between control points in a very similar manner to what we did for previous constraints. As demonstrated in Figure 3.7, a useful application for this feature is generating an equality constraint in y-axis for two control points at a switch frame when contact anchor is transferred from one to another.

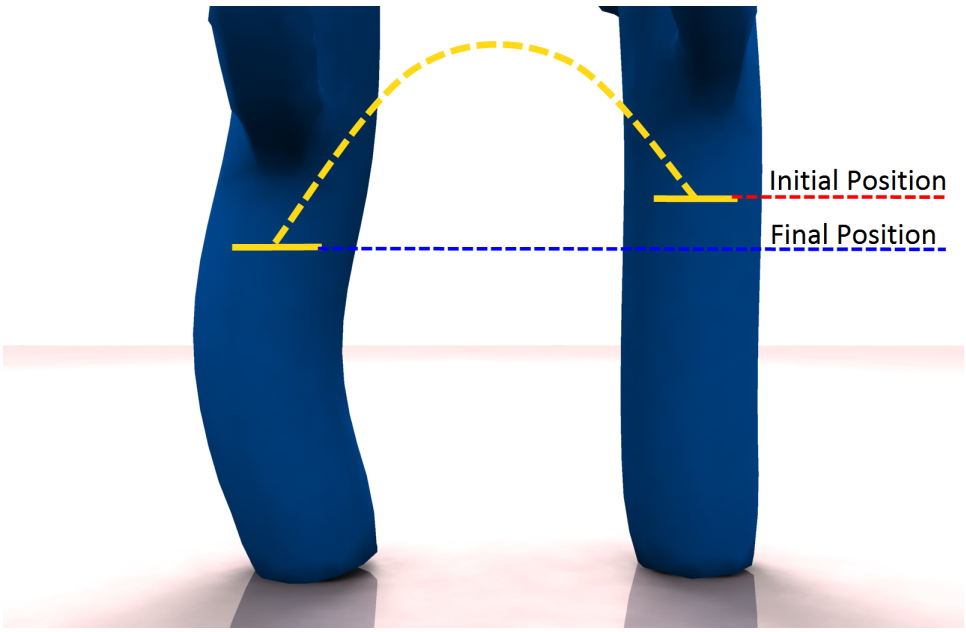


Figure 3.9: Flight phase example on retargeting result of jumping motion to Cactus model. Initial position and final position in y-axis can be found from retargeting results in local coordinate frame. Together with given duration time of the flight, an appropriate initial velocity can be found using simple kinematic equations.

3.3.4 Adding Global Position

Since the result of trajectory retargeting only includes the motion in local coordinate frame of the mesh, we use a simple technique to create appropriate global position, particularly for motions that include locomotion, or that accommodate flight (such as jumping, hopping, leaping).

Assuming that *switch frames* at which the source motion switches from contact phase to flight phase are given, our system uses two basic techniques to determine global position. During the mesh animation, at each switch frame, linear velocity of the mesh is calculated. Then, physical simulation of the mesh is activated and pursued until the mesh reaches to the ground. At this point, contact phase is activated where the closest control point to the ground is used as an anchor, and local movement of this point is reflected to the global position of the mesh.

Another way to improve results, when calculated velocity from retargeted motion is not good enough, is updating velocity in y-axis. Assuming that predefined phase information (i.e. every flight beginning and flight end frames), and displacement of the body in y-axis during flight are given, we can calculate an appropriate initial velocity for each flight using a simple equation:

$$v_i = \frac{\Delta_X - \frac{1}{2}(\Delta_F)(\Delta_F + 1)g}{\Delta_F}, \tag{3.31}$$

where displacement is calculated as $\Delta_X = x_f - x_i$ with initial position x_i and final position x_f, also Δ_F is the duration of the flight phase which is given by the user or extracted from the original human motion (Figure 3.9).

We manually extract switch frames or flight durations from the source motion, however automatic techniques [38] can be used if there is a correct correspondence between switch frames of the source and retargeted motion.

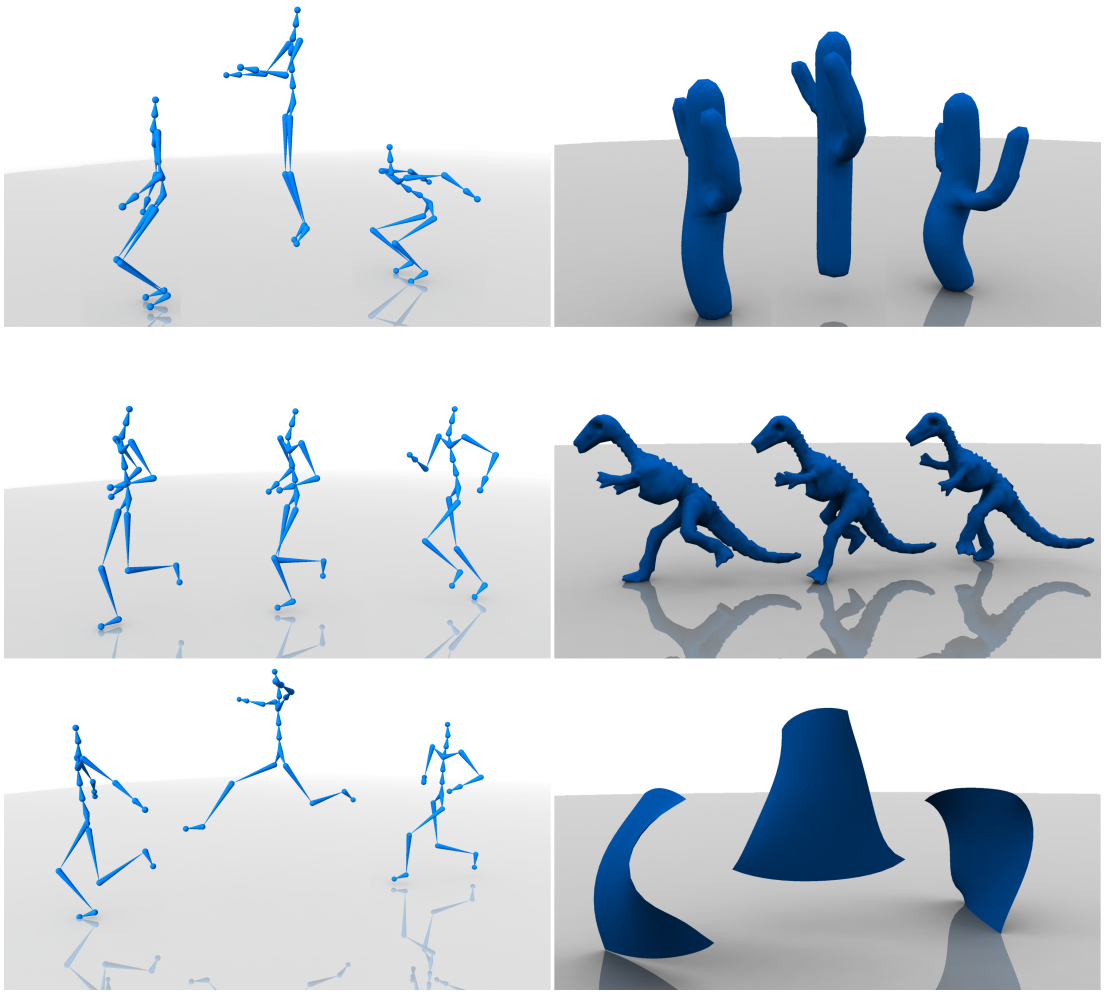


Figure 3.10: Retargeting human motions to arbitrary mesh models.

Chapter 4

Results

We have implemented our prototype system using C++ on AMD 2.1 GHz Dual-Core laptop with 4 GB RAM. For all required linear algebra calculations, we use Eigen library [39] including sparse linear system solvers, SVD and eigenvalue decompositions. For linear systems requiring boundary constraints, we use quadratic programming implementation provided by the optimization toolbox of Matlab [40].

4.1 Mesh Deformation

We have built an interactive deformation system based on the proposed deformation algorithm. In our system, the user can select control points on the mesh and deform the mesh by simply dragging them. Furthermore, the volumetric effect can be increased or decreased with a simple parameter λ_v.

Using our system, we compare our deformation results with ARAP deformation [4] and volume preserving deformation [5]. Compared to ARAP deformation [4], the main difference in practice is that rigid deformation can not properly handle cases where control point constraints impose changes in edge lengths (e.g.

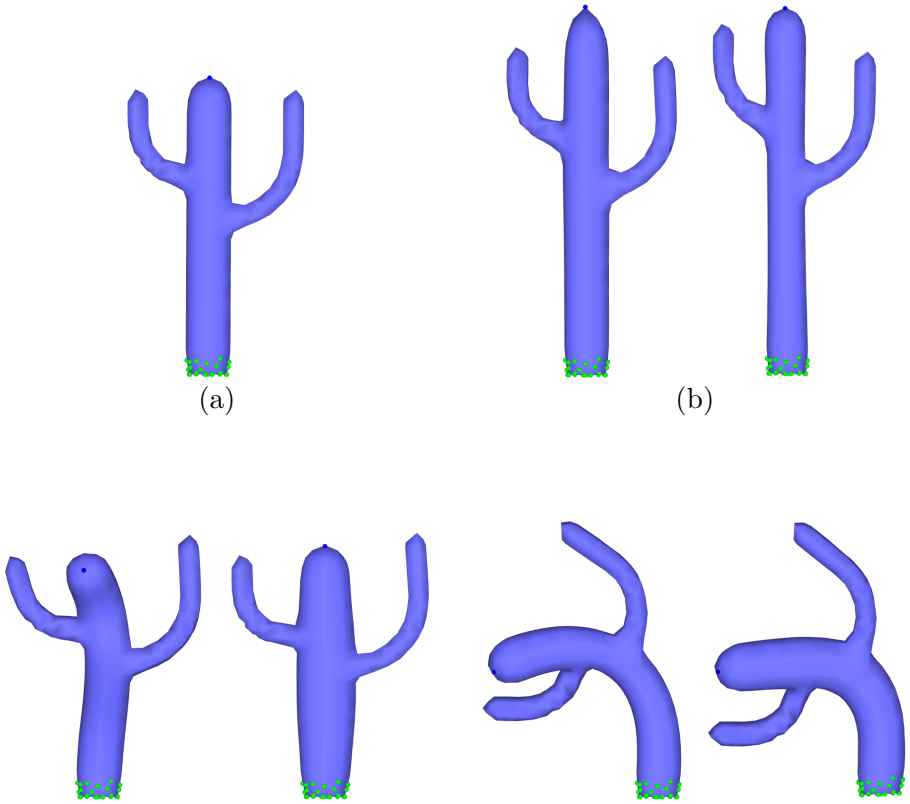


Figure 4.1: Comparison of cartoon-like and rigid deformation [4]. (a) is the original model; (b-d) contain ARAP deformation results at the left side and our deformation results at the right side. Since the nature of ARAP deformation relies on edge preservation, constraints which implicitly require non-rigid deformation (e.g. (b)-(c)) end up to unanticipated results.

squash and stretch). In such cases our results, together with volume preservation, are more expressive in providing squash-stretch effects, which reflects the true intention behind constraints. Figure 4.1 illustrates the comparison of two methods with two extremely constrained cases.

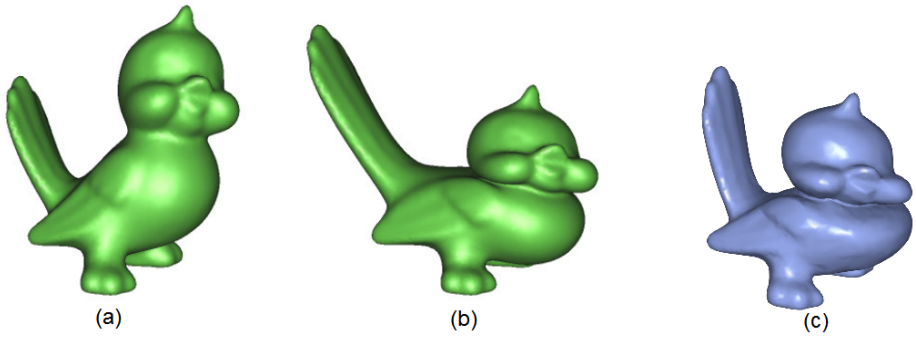


Figure 4.2: Comparison with Huang et al. [5]. (a) and (b) are reproduced here with permission. (a) The original model. (b) Their deformation result with volume preservation. (c) Our deformation result. Notice that, since volume preservation is a global effect in [5], squashing the mesh creates a balloon-like effect where the tail part of the bird is swelled. Our volume preservation works on deformed parts, and hence the tail remains intact.

In Huang et al. [5], although the algorithm is powerful, volume constraint has a global effect and satisfied regardless of deformed parts. In another words, squashing the mesh can create a balloon-like effect where any part of the mesh can get swelled. Our algorithm works on deformed parts while simulating the volume preservation, thus remaining parts do not get effected from volume related changes. In Figure 4.2, we provide a comparison.

We also compare our method with ARAP deformation in terms of performance. Our measure of performance consists of two main components: performance of a single iteration $\mathbf{\Delta_i}$, and the required number of iterations $\mathbf{N_i}$ for an acceptable convergence. In Table 4.1, we provide average response times of two methods, considering $\mathbf{\Delta_i}$, for different mesh models used in this thesis. For each frame, the ARAP method contains three iterations and our method contains two iterations of Figure 3.4-(a) and one iteration Figure3.4-(b). As expected, performance results are comparable in terms of $\mathbf{\Delta_i}$, since both methods contain SVD computation for rotations and sparse linear system solution for Laplace-Beltrami operator. For a coarse comparison of $\mathbf{N_i}$, we provide change rates of energy functions in Equation 2.9 and Equation 3.2 (Figure 4.3). Since we introduce a new

Table 4.1: $\mathbf{\Delta_i}$ - Deformation performance statistics, including the performance comparison of ARAP deformation and cartoon-like deformation in frames-per-second (fps).

Model	**# vertices**	**ARAP**	**Cartoon**
Cactus	620	86.1	69.7
Dinosaur	5621	9.0	7.5
Armadillo	10488	4.1	3.5
Bird	5302	8.9	7.6

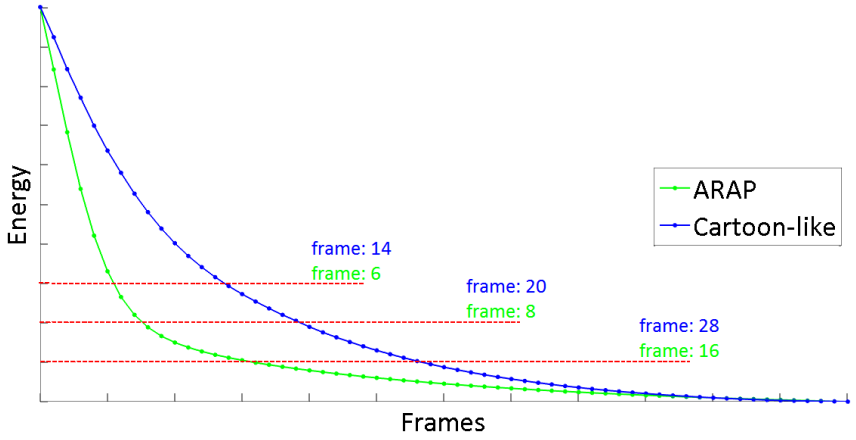


Figure 4.3: $\mathbf{N_i}$ - Change rates of energy functions in first 60 frames for a given constraints (repeated for several different constraints). Dashed red lines show frame numbers when energy functions drop at %30, %20, and %10. Similarly, we calculate average frame numbers between %30 and %5, by sampling each %1 drop, as a representative of $\mathbf{N_i}$. Combining $\mathbf{N_i}$ with $\mathbf{\Delta_i}$ leads to performance ratio of 2.4 on the average.

independent variable for scale coefficient in our equation, change rates of our function is slower than the ARAP method's. As a result, if we assume decreasing energy function between %30 and %5 is an acceptable convergence range, the performance ratio of our method and ARAP method is roughly 2.4 in terms of $\mathbf{\Delta_i N_i}$ on the average.

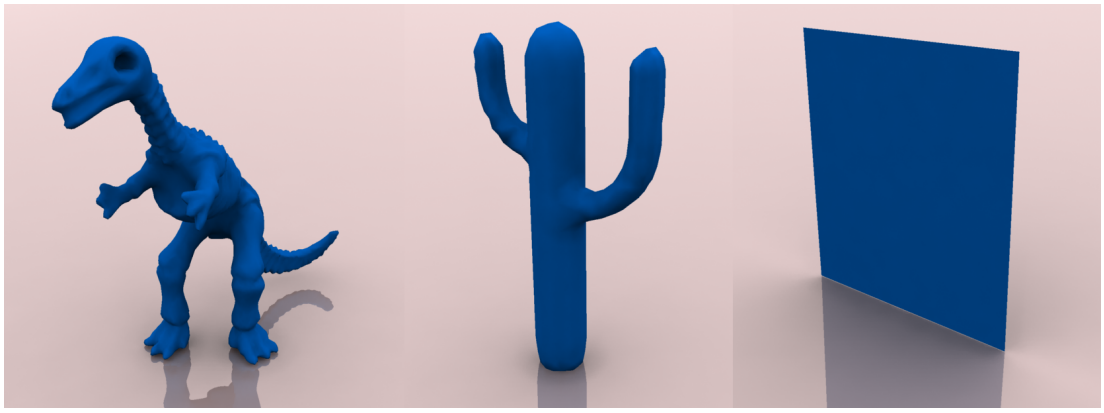


Figure 4.4: Reference mesh models used in motion retargeting system.

4.2 Motion Retargeting

In order to demonstrate abilities of our system, we use three well known mesh models (Figure 4.4) along with several motion capture data sequences obtained from CMU's motion capture database [41]. The accompanying video demonstrates the retargeting results.

During our try-out period with our system, we have observed the user expectations and behaviour. One of the mostly used features of the system is marking symmetric control points, so that the system can automatically double training data by simply using the symmetry plane. Another handiness of symmetrical marking is that generation of symmetric training data leads to symmetric results in retargeted motion, which is actually a generally desired property.

Another useful feature is constraint generation for retargeting process, which can be either done by the user for a case where constraint position does not represent a generic correspondence (hence not suitable for integrating in training data), but specific to a single frame. Also the system can automatically introduce constraints for forcing control points to have the same y-axis value when contact anchor is transferred from one to another.

The only possible parameter for our retargeting system is λ_A (i.e. weight of the acceleration constraint). Larger values set to this parameter may lead to lose

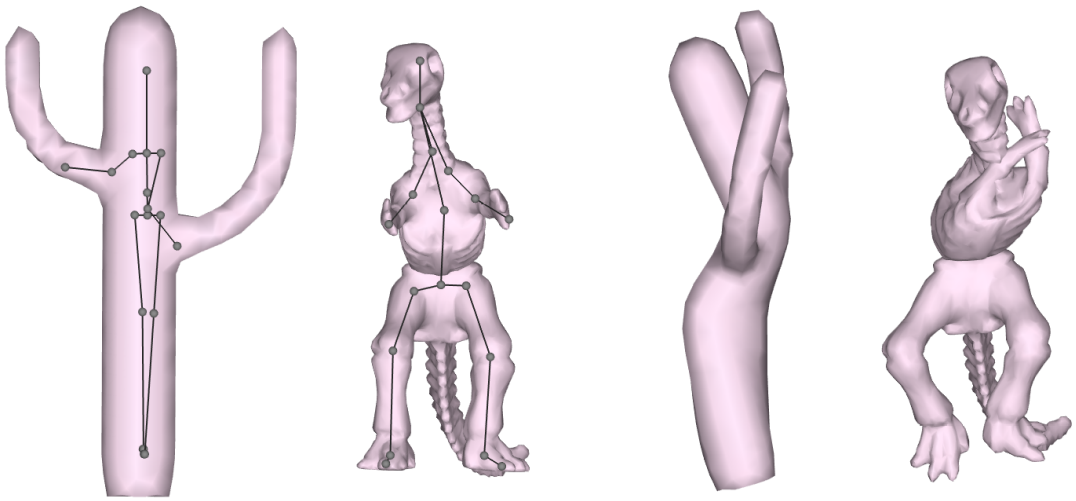


Figure 4.5: Rigging and animation results using [6] for jumping motion. One can see that legs of the dinosaur model are correctly rigged and animated since it is the most similar part to humanoid model.

high frequency details in retargeted motion, by making it look lifeless; smaller values might fail to reduce jitter and create unrealistic results. According to our experiments, we set λ_A to 0.3 for all our examples in this work, and we also demonstrate its effect in the video material.

We also use our models as inputs for Baran et al. [6], and Mixamo rigging and animation tool [42]. We accept that it is not exactly proper to compare our results with their results, since both methods require no training, and also target mesh models are considered to be humanoid models. However, our aim is investigating how automatic or semi-automatic approaches developed for humanoid structures work on such models rather than a comparison. Results produced by the automatic method of [6] are illustrated in Figure 4.5. Most satisfactory results are taken for legs of the dinosaur model, owing to its similarity to humanoid legs. Mixamo tool failed to rig cactus model because of the absence of legs. For the dinosaur model, rigging is completed but produced animation was not satisfactory in terms of rig quality and end effector placements.

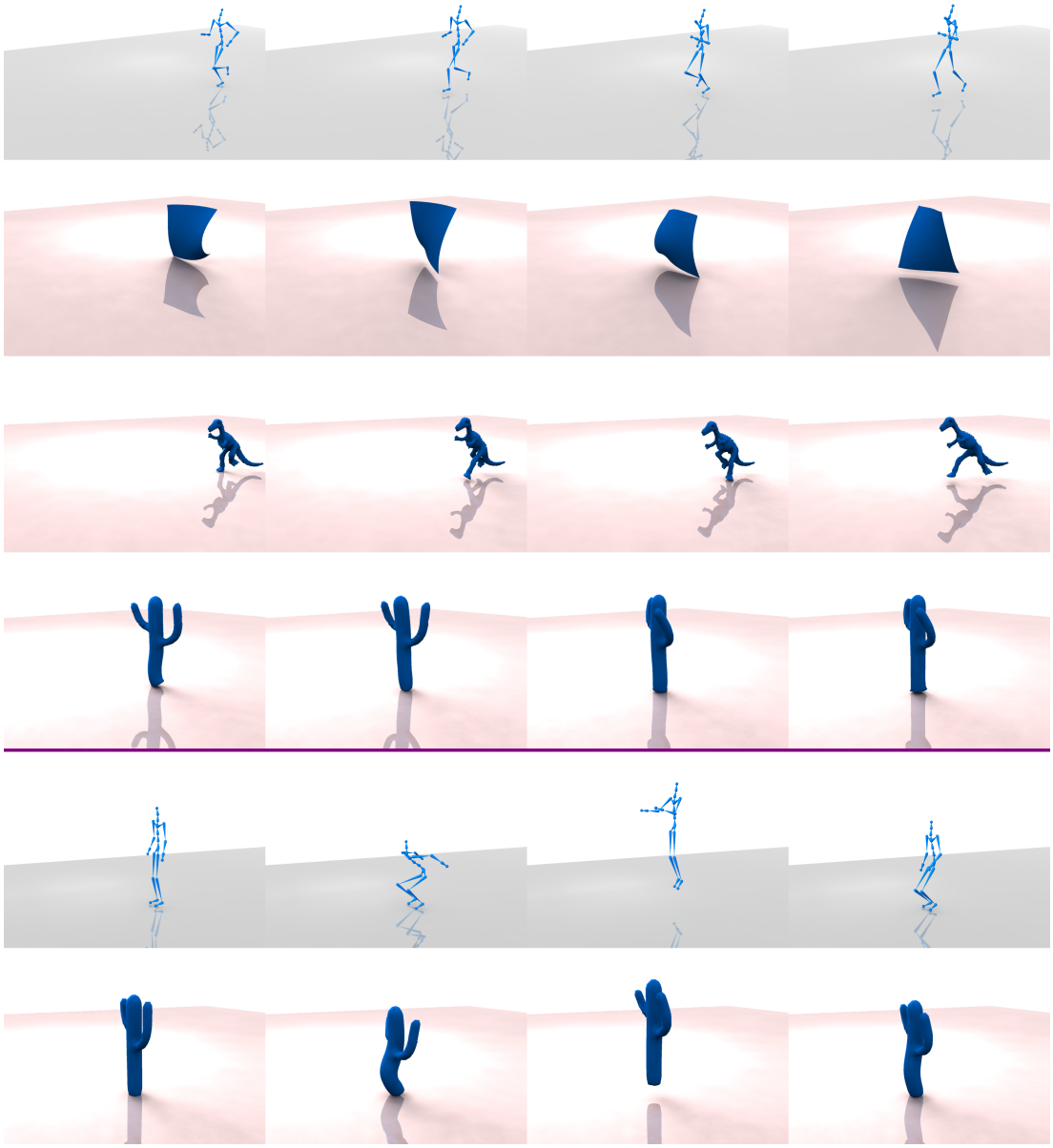


Figure 4.6: Chosen frames for demostraing retargeting results. More can be found in video material.

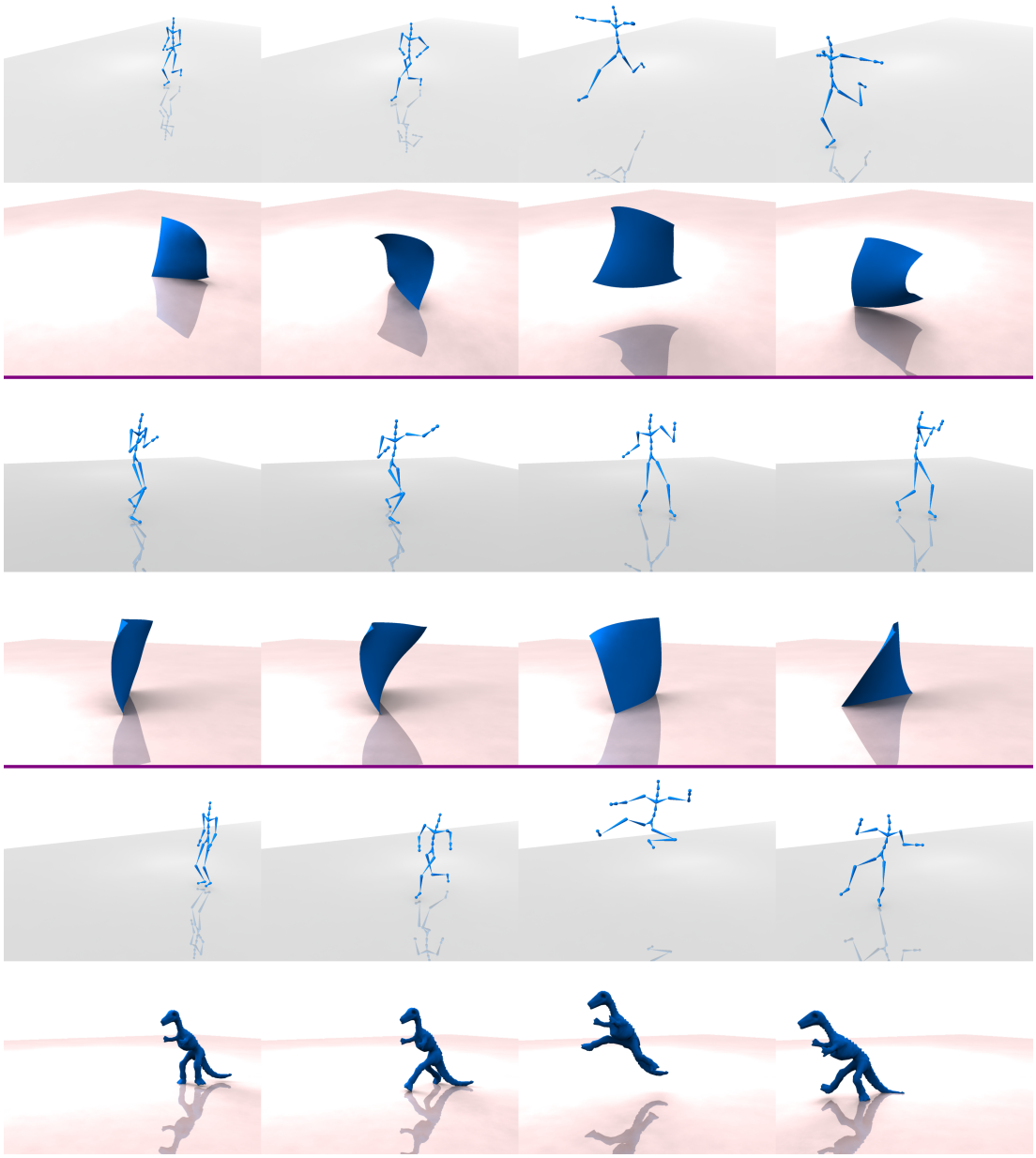


Figure 4.7: Chosen frames for demostraing retargeting results. More can be found in video material.

Chapter 5

Conclusion

We have introduced a new technique for retargeting motions from humanoid skeletons to arbitrary meshes by combining specialized deformation system with example-based spacetime trajectory retargeting. Our deformation system is suitable for generating squash and stretch effects by providing cartoon-like and volume preserving deformation. Our retargeting system is capable of retargeting human motions to arbitrary mesh models with the help of example poses provided by the user.

One limitation of our system is generation of global position for target mesh after trajectory retargeting. While our system is capable of generating plausible global trajectory for simple motions, effectiveness of more advanced techniques and extra post-processing steps can be seen clearly in results of [19]. Similarly our results can be improved by the help of refinements towards physical realism and more advanced global transformation generation system.

Another limitation is that fully articulated deformation of the mesh model is not possible with our deformation system. Actually, this problem is inherited from surface-based deformation approaches where no extra information about articulation of the mesh is considered. However, automatic mesh segmentation algorithms can be used for extracting mentioned articulation information by partitioning the surface into segments. Then, an augmented deformation system,

where the rigidity of segments are higher than the rigidity of segment boundaries, can provide articulated deformation as shown in applications of [43].

More improvements can be made to lessen or simplify required user interactions. Since our system allows the user to determine example-poses, our assumption is that the user provides enough extreme poses which represent the source motion. It might be possible to suggest a set of key skeleton-poses by processing the source motion and extracting a few poses which are as far as possible from each other as in [24]. Another improvement can be estimating the number of example-poses from a given source motion which balances the trade-off between retargeting quality and amount of user interaction.

Another possible improvement to our system is using different set of control structures other than points, since generating desired pose on target mesh by just using control points is relatively troublesome. Using curves for this purpose can be suitable, as demonstrated in [9], but the required user interaction is increased and the process of generating example poses becomes more complex.

Bibliography

[1] M. Gleicher and Z. Popović, "Motion editing, principles, practice and promise," *SIGGRAPH 2000 Course Notes*, p. 26, 2000.

[2] Y. Lipman, D. Levin, and D. Cohen-Or, "Green coordinates," *ACM Transactions on Graphics*, vol. 27, pp. 78:1–78:10, 2008.

[3] A. Wernicke, R. Joost, *et al.*, "GNU Image Manipulation Program: User Manual." http://docs.gimp.org/2.6/en/, 2010.

[4] O. Sorkine and M. Alexa, "As-rigid-as-possible surface modeling," in *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '07, pp. 109–116, 2007.

[5] J. Huang, X. Shi, X. Liu, K. Zhou, L.-Y. Wei, S.-H. Teng, H. Bao, B. Guo, and H.-Y. Shum, "Subspace gradient domain mesh deformation," *ACM Transactions on Graphics*, vol. 25, pp. 1126–1134, 2006.

[6] I. Baran and J. Popović, "Automatic rigging and animation of 3d characters," *ACM Transactions on Graphics*, vol. 26, 2007.

[7] Y. Lipman, O. Sorkine, D. Cohen-or, D. Levin, C. Rssl, and H. peter Seidel, "Differential coordinates for interactive mesh editing," in *Proceedings of Shape Modeling International*, pp. 181–190, 2004.

[8] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel, "Laplacian surface editing," in *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '04, pp. 175–184, 2004.

[9] K. Zhou, J. Huang, J. Snyder, X. Liu, H. Bao, B. Guo, and H.-Y. Shum, "Large mesh deformation using the volumetric graph laplacian," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 496–503, 2005.

[10] P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki, "Harmonic coordinates for character articulation," *ACM Transactions on Graphics*, vol. 26, 2007.

[11] D. K. Blandford, G. E. Blelloch, and D. E. Cardoze, "Compact representations of simplicial meshes in two and three dimensions," *International Journal of Computational Geometry & Applications*, vol. 15, pp. 3–24, 2005.

[12] U. Pinkall and K. Polthier, "Computing discrete minimal surfaces and their conjugates," *Experimental Mathematics*, vol. 2, pp. 15–36, 1993.

[13] O. Sorkine, "Least-squares rigid motion using SVD." http://igl.ethz.ch/projects/ARAP/, 2009.

[14] S. Zhang, A. Nealen, and D. Metaxas, "Skeleton based as-rigid-as-possible volume modeling," in *Proceedings of Eurographics 2010 - Short papers*, pp. 21–24, 2010.

[15] C. Zhang and T. Chen, "Efficient feature extraction for 2D/3D objects in mesh representation," in *Proceedings of International Conference on Image Processing*, pp. 935–938, 2001.

[16] M. Gleicher, "Retargetting motion to new characters," in *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pp. 33–42, 1998.

[17] J. sbastien Monzani, P. Baerlocher, R. Boulic, and D. Thalmann, "Using an Intermediate Skeleton and Inverse Kinematics for Motion Retargeting," *Computer Graphics Forum*, vol. 19, pp. 11–19, 2000.

[18] L. Ikemoto, O. Arikan, and D. Forsyth, "Generalizing motion edits with gaussian processes," *ACM Transactions on Graphics*, vol. 28, pp. 1–12, 2009.

[19] K. Yamane, Y. Ariki, and J. Hodgins, "Animating non-humanoid characters with human motion data," in *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pp. 169–178, 2010.

[20] C. Hecker, B. Raabe, R. W. Enslow, J. Deweese, J. Maynard, and K. V. Prooijen, "Real-time motion retargeting to highly varied user-created morphologies," *ACM Transactions on Graphics*, vol. 27, 2008.

[21] G. Bharaj, T. Thormahlen, H.-P. Seidel, and C. Theobalt, "Automatically rigging multi-component characters," *Computer Graphics Forum*, vol. 31, pp. 755–764, 2012.

[22] O. K.-C. Au, C.-L. Tai, H.-K. Chu, D. Cohen-Or, and T.-Y. Lee, "Skeleton extraction by mesh contraction," *ACM Transactions on Graphics*, vol. 27, pp. 44:1–44:10, 2008.

[23] J. Li, G. Lu, and J. Ye, "Automatic skinning and animation of skeletal models," *The Visual Computer*, vol. 27, pp. 585–594, 2011.

[24] I. Baran, D. Vlasic, E. Grinspun, and J. Popović, "Semantic deformation transfer," *ACM Transactions on Graphics*, vol. 28, pp. 36:1–36:6, 2009.

[25] C. Bregler, L. Loeb, E. Chuang, and H. Deshpande, "Turning to the masters: motion capturing cartoons," *ACM Transactions on Graphics*, vol. 21, pp. 399–407, 2002.

[26] K. Na and M. Jung, "Hierarchical retargetting of fine facial motions," *Computer Graphics Forum*, vol. 23, pp. 687–695, 2004.

[27] H. Pyun, Y. Kim, W. Chae, H. W. Kang, and S. Y. Shin, "An example-based approach for facial expression cloning," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pp. 167–176, 2003.

[28] Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or, "Linear rotation-invariant coordinates for meshes," *ACM Transactions on Graphics*, vol. 24, pp. 479–487, 2005.

[29] R. Zayer, C. Rössl, Z. Karni, and H.-P. Seidel, "Harmonic guidance for surface deformation," *Computer Graphics Forum*, vol. 24, pp. 601–609, 2005.

[30] A. Sheffer and V. Kraevoy, "Pyramid coordinates for morphing and deformation," in *Proceedings of the 3D Data Processing, Visualization, and Transmission*, 3DPVT '04, pp. 68–75, 2004.

[31] X. Shi, K. Zhou, Y. Tong, M. Desbrun, H. Bao, and B. Guo, "Mesh puppetry: cascading optimization of mesh deformation with inverse kinematics," *ACM Transactions on Graphics*, vol. 26, 2007.

[32] S. Chenney, M. Pingel, R. Iverson, and M. Szymanski, "Simulating cartoon style animation," in *Proceedings of the International Symposium on Non-photorealistic Animation and Rendering*, NPAR '02, pp. 133–138, 2002.

[33] M. Botsch and O. Sorkine, "On linear variational surface deformation methods," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, pp. 213–230, 2008.

[34] D. O'Leary, "Solving sparse linear systems: taking the direct approach," *Computing in Science Engineering*, vol. 7, pp. 62 – 67, 2005.

[35] M. Seeger, "Low rank updates for the Cholesky decomposition." http://infoscience.epfl.ch/record/161468/files/cholupdate.pdf, 2004.

[36] D. P. O'Leary, "Updating and downdating matrix factorizations: A change in plans," *Computing in Science and Engineering*, vol. 8, pp. 66–71, 2006.

[37] P. E. Gill, W. Murray, and M. H. Wright, *Practical optimization*. London: Academic Press Inc., 1981.

[38] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard, "Interactive control of avatars animated with human motion data," *ACM Transactions on Graphics*, vol. 21, pp. 491–500, 2002.

[39] G. Guennebaud, B. Jacob, *et al.*, "Eigen v3." http://eigen.tuxfamily.org, 2010.

[40] MATLAB, *version 7.7.0 (R2008b) Optimization Toolbox*. Natick, Massachusetts: The MathWorks Inc., 2008.

[41] CMU, "CMU graphics lab motion capture database." http://mocap.cs.cmu.edu/, 2012.

[42] Mixamo-Inc., "Mixamo: Automatic rigging and animation tool." http://www.mixamo.com/, 2012.

[43] A. Golovinskiy and T. Funkhouser, "Randomized cuts for 3d mesh analysis," *ACM Transactions on Graphics*, vol. 27, pp. 145:1–145:12, 2008.