Position-based Skin Deformations for Interactive Character Animation

NADINE ABU RUMMAN

A dissertation submitted to the Sapienza University of Rome in accordance with the requirements of the degree of DOCTOR OF PHILOSOPHY in the department of computer, control, and management engineering

MARCH 2016

Word count: forty-three thousand, three hundred eighty

Position-based Skin Deformations for Interactive Character Animation

NADINE ABU RUMMAN

Faculty of Information Engineering, Informatics and Statistics
SAPIENZA UNIVERSITY OF ROME

Thesis Committee
Advisor: Prof. Marco Schaerf
Co-Advisor: Prof. Marco Fratarcangeli

Author's declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: ..... DATE: .....

Dedicated to

The Memory of My Brother Abdul Ilah and Professor Burhan Shraideh

"There is no particular mystery in animation ⋯
it's really very simple, and like anything that is simple, it is about the hardest thing in the world to do"

-BILL TYTLA

Acknowledgements

I would like to express my sincere gratitude to Professor Marco Fratarcangeli for being so patient and for all the discussions. Also, for forcing me to aim for perfection and for teaching me to never give up on my dreams. His support and vigilance have allowed me to achieve results that I could not have thought of. The outcome of this dissertation owes much to his enthusiastic encouragement and continuous guidance. Thanks must also go to my advisor Professor Marco Schaerf; I am indebted to him for his endless support, motivation and patience. Furthermore, I am especially grateful to Professor Ladislav Kavan for being a great source of inspiration. I have watched all of his amazing online lectures, in which I have learned a great deal of information from them. I am also grateful to Professor De Giacomo for his tremendous support.

I am particularly grateful to Professor Dominique Bechmann for inviting me to the ICube Lab at University of Strasbourg (France), and for letting me enjoy and learn from the IGG team during the period of my visit. It was a privilege to collaborate with such a talented group. Further, I owe infinite thanks to Bart de Keijzer for proofreading this dissertation as well as for all his suggestions for improvements. I would like to thank the members of my Ph.D. committee who kindly agreed to evaluate this dissertation. Also, I am thankful to the reviewers who provided feedback for each one of my publications. I am sincerely indebted and grateful to Avempace-Erasmus Mundus Project for Technology for funding this work. Special thanks go to my supervisors (back home at Princess Sumaya University for Technology) Professor Jalal Atoum, Professor Issa Batarseh, Professor Arafat Awajan, Professor Rawan Ghnemat, Professor Baha Khasawneh and Professor Edward Jaser for their great support and for believing in me.

Lastly but most importantly, I am very grateful to my mom for her encouragement, her mental support during my study, and for helping me deal with the many difficult moments I have been through. I would like to thank God for giving me the strength all the way up until this stage to finish my Ph.D. studies.

Abstract

Skeletal animation is a widely used technique for animating articulated characters, such as humans and animals. In skeleton-based animation, skinning is the process of defining how the geometric surface of the character deforms according to a function of the skeletal poses. One of the fundamental aspects when animating articulated character is the production of flesh-like deformations for the soft tissues when the character is moving. Creating believable and compelling skin deformations is the central challenge of animated feature films, computer games, and interactive applications. Traditionally, the skin deformations are driven by an underlying skeleton. The idea can be formulated with a simple expression that binds the character's skin mesh with its underlying skeleton, whose bones can be transformed in order to obtain a smooth non-rigid deformation of the surrounding mesh. The deformation of each mesh vertex is computed as a weighted blend of the bones transformations. This technique does not generate realistic deformations and it usually suffers from unsightly artefacts. Moreover, skeleton-based deformation methods are incapable of capturing secondary motion effects, such as volume preservation, skin contact effects and the jiggling behaviors of soft tissues when the character is moving. In contrast, by employing a physically based method into the skinning process, the believability and realism of character motions are highly enhanced. Physics-based simulations manage to bring skeleton-driven deformations beyond the purely kinematic approach by simulating secondary motions. Despite offering such interesting effects, physics-based simulation requires complex and intensive computations, and thus it is usually avoided in interactive applications such as computer games. Furthermore, once the deformation parameters are specified in the simulation, it is difficult to control the actual resulting shape of the character in every animation frame.

In this dissertation, we address the problem of creating believable mesh-based skin deformation for soft articulated characters. We present a novel two-layered deformation framework, which is able to mimic the macro-behaviors of the skin and capture secondary effects, such as volume conservation and jiggling. While minimizing the manual

post-processing time, our system provides the artist with some level of control over the secondary effects. Our system is practical, relatively easy to implement and fast enough for real-time applications. We also introduce an efficient method for detecting collisions and self-collisions on articulated models, in which we exploit the skeletal nature of the deformation to achieve a good real-time performance. The output of the collision detection algorithm is used to enhance our layered skin deformation with responsive contact handling, and supports contact skin deformation between skin parts.

TABLE OF CONTENTS

List of Figures

Figure **Page**

List of Tables

List of Algorithms

CHAPTER 1

INTRODUCTION

Character animation is a vital component of contemporary computer games, animated feature films and virtual reality applications (Fig. 1.1). The problem of character animation can best be described by the title of the animation bible: "*The Illusion of Life*" [Thomas & Johnston, 1981]. The focus is not on the problem of completing a given motion task, but more importantly on how this motion task is performed by the character. This does not necessarily require realistic behavior, but behavior that is believable, full of an expressive quality which captures the personality of the character [Demeure *et al.*, 2011]. Animation of human characters and other living creatures has long been one of the most important applications of deformable modelling in computer graphics, notably in movie production and more recently in increasingly video games, interactive medical applications and surgery simulations. However, believable skin deformation for character animation is a complex and subtle phenomenon because of the coupling between the skeleton and soft tissues of the character's skin. Moreover, modelling compelling skin deformations is difficult and computationally demanding due to the nonlinear inner mechanics of the flesh.

This dissertation addresses the problem of creating believable mesh-based skin deformation for soft articulated characters at interactive rates. In this context, the term

Figure 1.1: Character animation is a vital component of animated feature films and computer games. *Left.* An appealing character animation (video game: *Heavenly Sword*). *Right.* Characters move in a believable manner (animated feature film: *Shrek*).

character is used in a broader sense, referring to any 3D model that undergoes motion or deformation in a scene. A 3D model can be found across many application fields including virtual reality, video games and movie production, such as cartoon-like animals and life-like human characters with realistically deforming skin (Fig. 1.2). A character is classified as an articulated character, when it has a skeletal structure of bones connected by joints. Most robotic parts, animal-like and human-like creatures with limbs fall into this category. The skeletal motion of an articulated character is primarily governed by

Figure 1.2: The character motion does not necessarily require realistic behavior, but behavior that is believable, full of an expressive quality which captures the personality of the character [Park & Hodgins, 2008].

bone transformations. Deforming the character with its underlying skeleton provide purely kinematic motion for articulated rigid bodies (e.g. a robot). In this work, we con-

centrate on soft bodies, in which the term soft usually refers to the fact that the geometry of the 3D model is deformable. In the sense that there is an underlying layer of elastic tissue, as with human skin. The terms soft and deformable are very tightly connected, but not equivalent. For example the robot is deformable articulated character because it can undergo skeletal deformation along its joints. However, it is not soft, because its geometry is infinitely stiff. Deforming the skin of a character as a soft body can be done by using physical simulation, which allows the automatic synthesis of effects that are difficult to animate by hand, such as the jiggling or vibration of the soft tissues caused by gravity.

1.1 Skin Deformation

In character animation, skinning is the process of defining how a geometric surfaces deform according to skeletal poses. The most pervasive skinning method is to deform the skin of a character via an underlying skeleton. Given an animated skeleton, the deformation is computed by linearly blending the bone transformations to the skin. Traditionally, bone transformations describe the position and orientation of the joints. This technique is called "*skeletal subspace deformation*", also known as linear blend skinning (LBS [Magnenat-Thalmann *et al.*, 1988]). However, such a simple and linear blending to the bone transformations cannot be expected to capture complex deformations. Moreover, because the deformation is restricted to the subspace of affine transformations of the bones, this method has problems deforming the skin near joints due to volume loss or the well-known "*candy-wrapper*" artefacts (Fig. 1.3). By replacing linear blending with a nonlinear blending (dual quaternion skinning, [Kavan *et al.*, 2007]), the artefacts of LBS can be completely avoided. However, dual quaternion blending suffers from an undesirable joint-bulging artefact while bending, which requires artistic manual work to be fixed. Although these methods achieve good real-time performance, they are purely kinematic, lacking of secondary motion effects, such as the passive jiggling motion of fatty tissues and contact deformation effects. In contrast, by employing a physically based method into the skinning process, the believability of character motions can be highly enhanced. Physics-based simulations manage to bring skeleton-based animation beyond the purely kinematic approach by simulating secondary motions like jiggling, volume preservation and contact deformation effects. These secondary motions enrich the visual experience of the animation and are essential for creating appealing character animation for animated feature films and computer games. Despite offering such realistic effects, physically based simulation is computationally demanding and complex, thus it

is usually avoided in interactive applications. Further, physically based methods require human intervention to generate input data, which describe the physical states, before the problem can be solved by a computer program. In most cases, such preparation is tedious and the artist must master the knowledge of both the given software and the underlying physics of the phenomenon. The key challenge of producing believable deformations is to satisfy the conflicting requirements of real-time interactivity and believability. Believability requires achieving sufficient deformation detail, which means capturing the full range of desired effects, namely jiggling, volume preservation, muscle bulging and skin contact deformations. Producing these deformations demands at least an order of magnitude more computation power than current interactive deformation system.

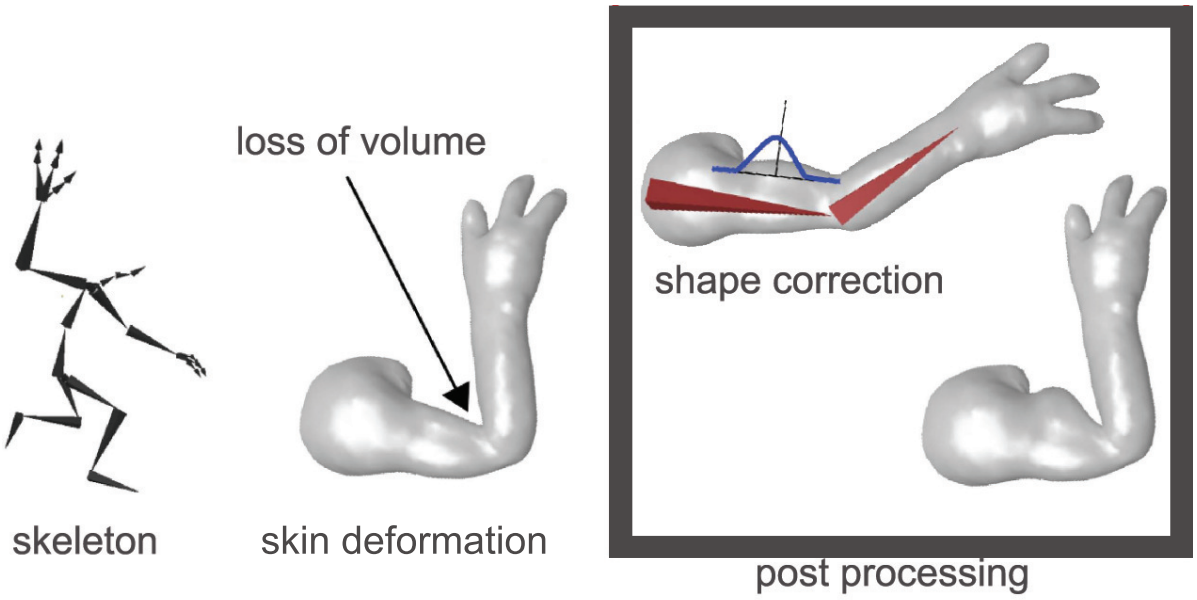


Figure 1.3: Given an animated skeleton, the deformation is computed by linearly blending bone transformations to the skin. Classic interactive skinning techniques, namely linear blend skinning suffers from loss of volume or the well-known "*candy-wrapper*" artefact, which requires manual post-processing work to be fixed [Rohmer *et al.*, 2008].

Therefore, we are motivated to (1) reduce the artefacts of the classic interactive skinning techniques, and minimize the manual post-processing time, (2) obtain believable skin deformations that support contact and collision, (3) capture appealing skin deformations including secondary motion effects in the real-time, (4) offer the artist the ability to modify and control the skin deformations at interactive rates.

1.2 Collision Handling

In order to simulate the skin's behavior in a believable manner, an appropriate collision response has to be considered. However, skin contact deformations due to collision are difficult to model using geometry-based skinning techniques, because bone transformations completely determine the resulting (deformed) shape. In contrast, physics-based methods can capture contact deformations (Fig. 1.4). In these methods, collision handling consists of two steps: collision detection and collision response. Often they are performed separately, with the result of the collision detection being used as input to the collision response algorithm.

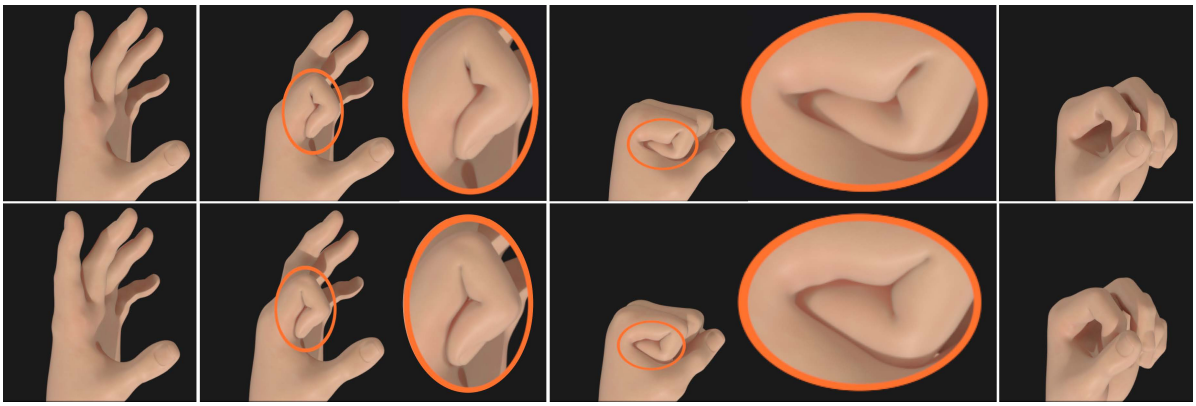


Figure 1.4: Top row: A hand deformed using geometry-based skinning method (linear blend skinning). Bottom row: The same hand deformed with physics-based method [McAdams *et al.*, 2011], which handles collision at near interactive rates. Note how the physics-based method get correct creasing and contact deformation where the finger bends.

Collision detection is the problem of checking for all possible geometric intersections between moving models in a virtual environment. Solving the collision detection problem has become of major interest in various application areas, ranging from games, animation and virtual reality to surgery simulation and robotics. Moreover, efficient and reliable collision detection is a critical part of almost all graphics applications. A significant amount of research has been done to detect interfering objects moving in space or find exact points of contact. Whereas most of the contributions concentrated on collision detection for rigid body simulations, recent approaches started focusing on deformable bodies, i.e., models whose shape changes during the simulation, for example articulated characters, soft tissues and cloth. Deformable collision detection is a challenging problem mainly because of its high computational complexity. Thus, it is difficult to devise an

efficient solution to accommodate the rapid interaction speed demanded by users. This is especially true when we are simulating the skin of a virtual character, as self-collisions of the limbs can occur and have to be handled. In this case, collision detection response requires specific information, since it is not sufficient to just detect the collisions. Handling self-collisions is a significant challenge, because collision handling algorithm has to be responsive so that skeletal and global motion are handled simultaneously and naturally with surface deformations. Nevertheless, resolving self-collisions is essential to generate believable skin deformation.

1.3 Contributions

Traditionally, the character's skin is deformed by its underlying skeleton. In this case, the believability of the deformation is limited, since the deformation is purely kinematic, lacking of secondary motions effects and skin contact deformations. The use of physics in the skinning process enhances the sense of realism: in it enables us to produce secondary effects such as jiggling, as well as to simulate the skin contact deformation due to collisions. Unfortunately, the existing techniques cannot generally achieve these effects in real-time and they do not support artistic control either. The goal of this dissertation is to develop method that provides compelling skin deformations at interactive rates. This includes the full range of desired effects such secondary motions and skin contact deformation. Further, we would like to give the artist some level of control over the deformation. The contributions of this dissertation can be summarized as follows:

Position Based Skinning (Chapter 4) We introduce a two-layered approach addresses the problem of creating believable mesh-based skin deformation. Position based skinning (PBS) is a practical framework for deforming soft articulated characters in real-time. This framework avoids the artefacts of classic interactive skinning techniques, namely the "*candy-wrapper*" artefacts of linear blend skinning (LBS) and the bulging artefacts of dual quaternion skinning (DQS). Furthermore, our layered skin deformation is: 1) able to mimic the macro-behaviors of the skin, 2) able to provide secondary motion effects such as volume preservation and jiggling of the fat tissues when the character is moving, and 3) relatively easy to implement and fast to compute.

Parallelization and Shape Control (Chapter 4) To further improve the performance, we solved the geometric constraint in parallel. In our two-layered approach, we simulate the skin as a soft body, and the deformation approach is based on position

based dynamics [Müller *et al.*, 2007] scheme. Hence, we applied our method in a parallel fashion and solved it on a multi-core CPU, which leads to a significant improvement in performance. Our method requires less mathematical complexity and provides believable animations with at least one order of magnitude faster computation time than existing state-of-the-art methods. We also allow the artist to interactively control the behavior of the skin by tuning the amount of the jiggling effect and manually specifying a specified area of the skin affected by it.

Collision Handling (Chapter 5) Despite offering interesting effects like secondary motions and volume preservation, position based skinning (Chapter 4) cannot guarantee the absence of self-intersection. To address the lack of collision and contact handling inside position based skinning, we locally handle the self-intersection problem in the areas around the joints. First, we present an efficient method for detecting collisions and self-collisions on articulated models. The proposed method employs spatial hashing with a uniform grid to detect collisions. Our collision detection method focuses on skeletal characters, where we use a lazy procedure that updates the hash table in an on demand way.

To handle collisions and capture contact deformations locally, we formulate a constraint-based contact response within our the position based skinning framework.

1.4 Publications

- The method presented in Chapter 4 has been published as an article in the proceedings of the 30th Spring Conference on Computer Graphics (SCCG, 2014) with title "Position based Skinning of Skeleton-driven Deformable Characters", pages 83-90. The paper received best paper award, best presentation award and invited to submit an extended article version as journal publication to Computer Graphics Forum.

- The overall techniques presented in Chapter 4 has been published as journal publication with title "Position-Based Skinning for Soft Articulated Characters", (Computer Graphics Forum, 2015), Volume 34, Issue 6, pages 240-250.

- Part of the material presented in Chapter 5 has been publication as an article in the proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games (MIG,

2015) with title “Collision Detection for Articulated Deformable Characters", pages 215-220.

- The literature review presented in Chapter 2 has written as state of the art paper with title “State of the Art in Skinning Techniques for Articulated Deformable Characters" has been accepted for publication in the 11th International Conference on Computer Graphics Theory and Application (GRAPP, 2016) [Rumman & Fratarcangeli, 2016].

1.5 Organization

The remainder of this dissertation is organized as follows:

Chapter 2, Literature Review. This chapter presents a literature review of the previous work on skin deformation. We mainly focus on methods that are relevant to skeleton-based deformation techniques, soft body simulations and physically based skinning methods. We also outline the research that has already been done in these areas and how the work in this dissertation contributes to it. Moreover, this chapter briefly reviews previous work on collision handling of deformable bodies or models undergoing specific types of deformation, such as skeletal deformation.

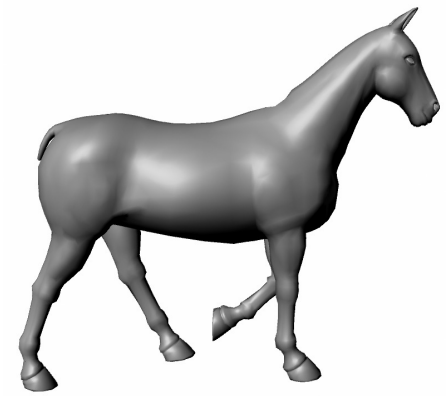
Chapter 3, Layered Skin Deformation. This chapter gives a detailed insight into the necessary background material. It also provides a quick tutorial on the computational frameworks for skeleton-based deformation and physically based simulation. Before presenting our novel skin deformation model (Chapter 4 and Chapter 5), we introduce in this chapter the concept of layered skin deformation, which serves as the basis for the developments presented in the following chapters.

Chapter 4, Position based Skinning for Soft Articulated Characters. This chapter presents a two-layered deformation model named position based skinning. Position based skinning is a new deformation method, which avoids the artefacts of classic interactive skinning techniques, and provides believable skin deformation for articulated characters at interactive rates. Moreover, it captures interesting effects, such as volume preservation and jiggling. The artist is also allowed to control the amount of jiggling and the area of the skin affected by it.

Chapter 5, Collision Handling for Soft Articulated Characters. This chapter tackles the problem of self-intersections for models deformed by position based skinning, where we exploit the skeletal nature of the deformation to achieve real-time collision detection. Then, we formulate a constraint-based contact response within the position based dynamics framework, to handle self-intersections and capture contact deformations effects.

Chapter 6, Conclusion and Future Directions. In this chapter, we summarize the main contributions of this dissertation and suggest ideas for future research.

CHAPTER 2



LITERATURE REVIEW

Creating believable and compelling skin deformations for articulated virtual characters is a multi-disciplinary problem, which can be divided into three main problems: generating high-quality skin deformations, simulating skin contact in response to collisions, and producing secondary motion effects such as flesh jiggling when a character moves. In this chapter, we focus on the existing work that is closely related to skeleton-based deformation, soft character simulation, physics-based skinning techniques and contact handling. There has been a considerable amount of research on each of these subjects; it is beyond the scope of this dissertation to exhaustively survey the vast literature. This chapter discusses the literature that is most relevant to our work, and we highlight the advantages and disadvantages of each existing skinning method. For a more thorough treatment, we refer the reader to [Turner & Gobbetti, 1998; McLaughlin *et al.*, 2011; Sifakis & Barbic, 2012; Geijtenbeek & Pronost, 2012; Jacobson *et al.*, 2014a].

2.1 Skeleton-based Skinning Methods

The most common technique for deforming articulated characters is to define the surface geometry as a function of an underlying skeletal structure. Due to the simplicity, intuitive

manipulation, and the ability to quickly solve inverse-kinematics on a small subspace (the skeleton), skeleton-based methods are very popular and widely used in the animation industry. In modelling a skeleton-based deformation, the challenge is to obtain high-quality skin deformations in real-time, given an arbitrary skeletal posture. Several techniques have been proposed to provide these deformations, in which the current skeleton-based deformation techniques can be classified into two categories: geometric methods (Section 2.1.1) and example-based methods (Section 2.1.2).

2.1.1 Geometric Skinning Techniques

In geometric skinning techniques, skeleton-to-skin binding is defined in a direct, geometrical way [Magnenat-Thalmann *et al.*, 1988; Komatsu, 1988; Walter & Fournier, 1997; Singh & Kokkevis, 2000; Kavan & Zara, 2003; Hejl, 2004; Kavan & Zára, 2005; Rhee *et al.*, 2006; Forstmann *et al.*, 2007; Kavan *et al.*, 2007; Vaillant *et al.*, 2013]. Geometric approaches to deform articulated characters have shown reasonable results at interactive rates.

We start by discussing the standard real-time method "*skeletal subspace deformation*", also known as linear blend skinning (LBS) [Magnenat-Thalmann *et al.*, 1988]. This method has been widely adopted in real-time applications such as games, for its computational efficiency and straightforward GPU implementation. Unfortunately, linear blend skinning suffers from visual artefacts like self-intersection, volume loss or the well-known "*candy-wrapper*" artefact [1] (see Fig. 2.1), which are the result of the linear nature of the algorithm, since the linear interpolation of the transformation matrices is not equivalent to the linear interpolation of their rotations [Alexa, 2002]. The limitations of LBS have been extensively studied, where many techniques have been proposed to avoid its artefacts. One possibility is by enriching the space with skinning weights, leading to methods which are still linear, but feature more parameters than in linear blend skinning. These methods are called multi-linear skinning techniques [Wang & Phillips, 2002; Mohr & Gleicher, 2003], in which the extra weights are learned from input examples and regularization is used to prevent overfitting. Merry et al. propose a multi-linear skinning model called *Animation Space* [Merry *et al.*, 2006], which uses 4 weights per vertex-bone pair. However, this increase in the number of weights carries an additional cost in time and space, as well as parameter passing. While linear skinning techniques are popular due to their efficient implementations, which make them well suited for use

[1] the"*candy-wrapper*" artefact is the skin collapsing effect exhibited by linear blend skinning [Magnenat-Thalmann *et al.*, 1988].

in interactive applications. They cannot totally remove the "*candy-wrapper*" artefact, which is in all cases noticeable under large joint rotations. For a comprehensive survey on linear skinning techniques, we refer the interested reader to [Jacka *et al.*, 2007]. Selecting good skinning weights is critical to avoid the artefacts and generate more natural deformations. Recently, an automatic computation of skinning weights was presented in [Dionne & de Lasa, 2013]. In their method, the influence weights are determined using geodesic distances from each bone, which makes the inverse-distance weights shape-aware and can work with production meshes (that may contain non-manifold geometry). Despite that associating skinning weights with the mesh vertices can be done automatically, this method tends to either increase or decrease the volume around joints.

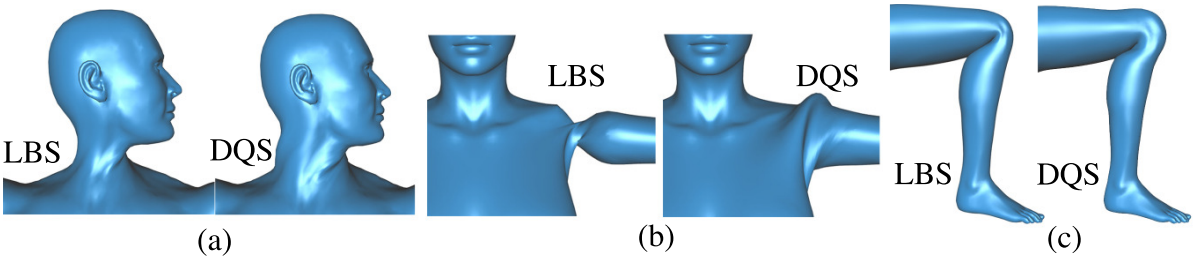


Figure 2.1: Artefacts of classic interactive skinning techniques linear blend skinning (LBS) and dual quaternion skinning (DQS). Linear blend skinning (LBS) is the most widely employed skinning technique, due to its simplicity and efficiency. Unfortunately, LBS suffers from the "*candy-wrapper*" artefact while twisting ((a) and (b)). This artefact can be eliminated by a nonlinear blending method such as dual quaternion skinning (DQS), but DQS produces an unnatural joint-bulging artefact while bending (c). Observing that LBS does not produce bulging while bending and DQS does not suffer from the "*candy-wrapper*" artefact while twisting [Kavan & Sorkine, 2012].

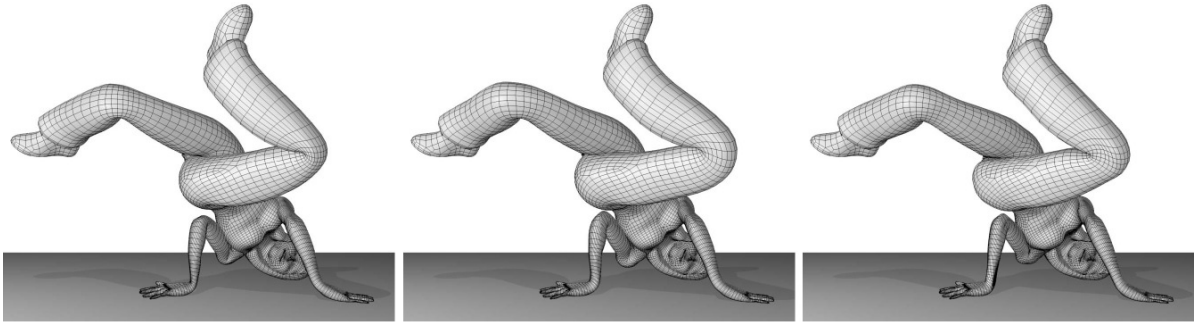
By replacing linear blending with nonlinear blending [Hejl, 2004; Kavan & Zára, 2005; Kavan *et al.*, 2007], the "*candy-wrapper*" artefact can be completely avoided. Nonlinear skinning methods convert rigid transformation matrices to (quaternion, translation) pairs and blend them instead of their matrix equivalents [Hejl, 2004; Kavan & Zára, 2005]. This works, but *Hejl's method* [Hejl, 2004] imposes some constraints on the character's rigging, whereas *spherical skinning* [Kavan & Zára, 2005] uses a computationally expensive Singular Value Decomposition (SVD) scheme. Besides, the practical impact of these two methods is limited, because of their dealing with the translational component of the skinning transformations. In contrast, dual quaternion skinning (DQS [Kavan *et al.*, 2007]) uses an approximate blending technique based on dual quaternions (essentially, two regular quaternions). Although dual quaternion skinning is able to achieve

comparable speeds to LBS, while retaining the increase in visual quality. It suffers from an undesired *joint-bulging*[2] artefact (as we can see in Fig. 2.1), which requires artistic manual work to be fixed.

Because fixing these artefacts manually is a tedious process, automatic skinning techniques are becoming increasingly popular [Baran & Popović, 2007; Wareham & Lasenby, 2008; De Aguiar *et al.*, 2008; Kavan *et al.*, 2009; Chen *et al.*, 2011; Jacobson *et al.*, 2011; Bharaj *et al.*, 2012; Jacobson *et al.*, 2014b]. Moreover, an interesting extension of linear blend skinning called *spline-skinning* comes from [Yang *et al.*, 2006; Forstmann & Ohya, 2006; Forstmann *et al.*, 2007], which often produces better skinning deformations and suppresses (but not completely eliminates) the "*candy-wrapper*" artefact. Instead of using conventional matrix rotation, spline-skinning represents each bone of the skeleton by a spline. Furthermore, an appealing extension of DQS that is successfully applied in a production setting (Disney's *Frozen*), can be seen in [Lee *et al.*, 2013]. For a extensive discussion on nonlinear skinning methods, we refer the reader to [Kavan *et al.*, 2009]. Whilst all the above-mentioned methods fully define the surface positions based on skeletal configuration, they cannot capture secondary motion effects and skin contact behavior in response to collision. Recently, more advanced geometric skinning methods were introduced to limit the artefacts of LBS, while keeping their simplicity. Kavan and Sorkine [Kavan & Sorkine, 2012] developed a new skinning method based on the concept of joint-based deformers, which avoids the artefacts of linear blend skinning as well as the bulging artefact of dual quaternion skinning. More interesting technique proposed by Jacobson et al. [Jacobson & Sorkine, 2011], where they expanded skinning to support bending, stretching and twisting by using a slight variation on the standard skinning equations. Impressive skinning results can be obtained using the technique presented in [Vaillant *et al.*, 2013], which generates visually plausible skin deformations in real-time (see Fig. 2.2). Their method automatically captures contact surfaces between skin parts, without requiring any collision detection step. Moreover, they extended their framework to handle local skin contacts and produce the effect of skin elasticity (sliding effect) [Vaillant *et al.*, 2014]. More recently, [Kim & Han, 2014] proposed a post-processing method for dual quaternion skinning, which eliminates the joint-bulging artefacts and its suitable for real-time character animation.

In spite of improvements, skinning using geometric skinning techniques remains purely

[2]joint-bulging is an unnatural skin bulging effect produced by dual quaternion skinning [Kavan *et al.*, 2007] while bending.



linear blend skinning | dual quaternion skinning | implicit skinning

Figure 2.2: Dana model in a break-dance pose. From *left* to *right*, the model is deformed with linear blend skinning, dual quaternion skinning and implicit skinning. Note the visible loss of volume produced by LBS (*left*). Implicit skinning (*right*), however, generates visually plausible skin deformations, which avoids the artefacts of linear blend skinning, as well as the bulging artefacts of dual quaternion skinning [Vaillant *et al.*, 2013].

kinematic, lacking of secondary motions effects like passive jiggling motion of the fatty tissues or muscle bulging. In the next section, we present the most influential example-based skinning methods, which are able to alleviate the limitations of geometric skinning method, and add dynamic effects to the skin.

2.1.2 Example-based Skinning Methods

In contrast to geometric approaches, example-based skinning methods [Lewis *et al.*, 2000; Sloan *et al.*, 2001; Kry *et al.*, 2002; Allen *et al.*, 2002; Kurihara & Miyata, 2004; Magnenat-Thalmann *et al.*, 2004; James & Twigg, 2005; Rhee *et al.*, 2006; Weber *et al.*, 2007; Park & Hodgins, 2008; Shi *et al.*, 2008; Lee & Hanner, 2009; Le & Deng, 2014] have permitted more complex skinning effects such as muscle bulges and wrinkles, while also addressing the artefacts of linear skinning techniques. These methods take as input a series of sculpted example poses and interpolate them to obtain the desired deformation. One of the first example-based methods is pose space deformation (PSD, [Lewis *et al.*, 2000]), which uses a radial basis function to interpolate correction vectors among the example poses. In pose space deformation method, *pose space* is a set of degrees of freedom of a character's model, which vary between the example poses. A particular *pose* is a set of particular values of these degrees of freedom. Pose space deformation comprises one family of approaches, in which example poses (or *local frame corrections*) are interpolated as a function of a character pose. A more sophisticated extension of PSD was presented in [Sloan *et al.*, 2001]. Their method interpolates an articulated character using example poses scattered in an abstract space. This abstract

space consists of dimensions describing global properties of the 3D character, such as age and gender, in addition to dimensions that are used to describe the configuration, such as the amount of bend at the elbow joint. Moreover, PSD was generalized to support weight (weighted pose space deformation WPSD, [Kurihara & Miyata, 2004; Rhee *et al.*, 2006]), which largely reduces the number of required example poses. Although WPSD can handle large-scale deformations well, it cannot provide detailed deformation and it requires more computation than the original pose space deformation (PSD). In these methods, the amount of memory grows with the number of training examples, thus they are more popular in animated feature film (DreamWorks Animation's Shrek 2) than in real-time application. To tackle this problem, [Kry *et al.*, 2002] proposed a method similar in spirit to PSD called EigenSkin. Instead of using all the displacements for example poses, they used precomputed principal components of deformation influences on individual joints. The resulting algorithm leads to considerable memory savings and enables to transfer the computations to the GPU. Despite the fact that pose space deformation methods are simple to implement, they require tremendous effort from artists, as they have to create different poses by hand for a wide variety of examples.

Another class of example-based methods, which is a direct generalization of LBS, but does not require data interpolation, is formed by methods such as single-weight enveloping (SWE, [Mohr & Gleicher, 2003]) and multi-weight enveloping (MWE, [Wang & Phillips, 2002]). Single-weight enveloping estimated single-weight per vertex with rigid character bones, with provisions made for adding additional bones. Multi-weight enveloping, however, is based on a linear framework supporting multiple weights per vertex-bone, where it provides better approximations than SWE, but at the cost of 12 weights per vertex-bone, instead of 1 weight per vertex-bone in SWE. However, linearity has certain benefits: it is fast and it can be used to derive a measure of average distance across the space of poses, but the example meshes are still necessary in order to obtain the weights. This class of methods allows a smaller number of poses to be used to generate a larger number of deformations, while introducing more weight parameters. Thus, these numerous parameters come at a cost of complicated computation of the weights.

As an alternative to using sculpted example poses; several example-based approaches use scanned or photographed data. Early work that uses 3D scanned poses of a human body in character skinning has been presented in [Min *et al.*, 2000]. Additionally, the method in [Allen *et al.*, 2002] creates a high quality posable upper body model from range scan data and markers. In their method, to learn the skinning model, they obtain

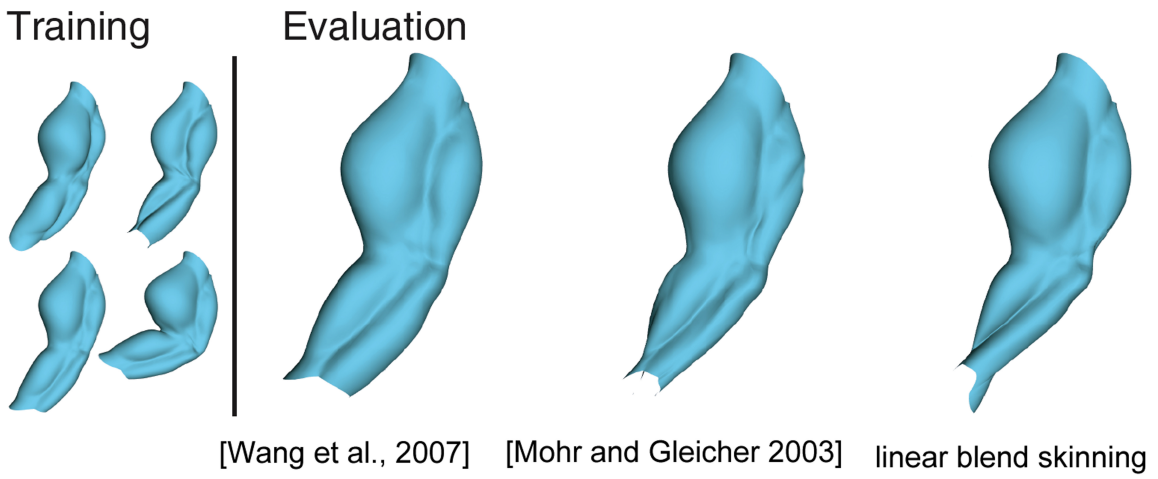


Figure 2.3: A set of example poses from an anatomically motivated arm model with both bending and twisting at the elbow. The twisting and muscle bulges are enough to prevent LBS from approximating the examples well. The technique of [Mohr & Gleicher, 2003] does better, but still differ from the given example poses. The model from [Wang *et al.*, 2007] well-approximate the examples poses.

deformations corresponding to different poses by matching a subdivision surface template to the range data. Recently, more advanced example-based techniques have been effectively integrated with mesh deformation algorithms to further improve the quality of skinning [Wang *et al.*, 2007; Shi *et al.*, 2008; Huang *et al.*, 2011; Schumacher *et al.*, 2012]. A rotational regression model was proposed in [Wang *et al.*, 2007], which captures common skinning deformation such as muscle bulging (as we can see in Fig. 2.3) and twisting, specifically in challenging regions such as the shoulders. Park and Hodgins also introduced an interesting technique that captures and synthesizes detailed skin deformations such as bulging and jiggling [Park & Hodgins, 2006; Park & Hodgins, 2008], when a character performs dynamic activities. They use a very dense and large set of markers to capture the dynamic motions (see Fig. 2.4). Then, they employ a second-order skinning scheme followed by a radial basis function of the residual errors to provide detailed skin deformations. While high-quality skin deformations can be captured accurately using scanned data, marker-based motion capture systems typically have a time-consuming calibration process and high hardware cost.

Example-based skinning methods are attractive since they can provide rich details from physical measurements and add realistic secondary deformation to the skeleton-based animations. Shi et al. presented an appealing method that is able to provide the jiggling of the fatty tissues in real-time by taking a surface mesh and a few sample sequences of

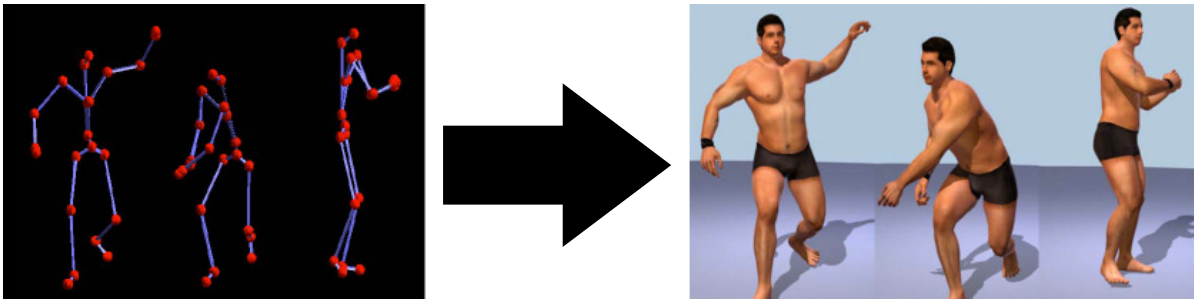


Figure 2.4: The method proposed in [Park & Hodgins, 2008] captures and synthesize detailed skin deformations given skeletal motion as input data. (a) Skeletal motion as input of different motions. (b) Detailed skin and muscle deformation.

its physical behavior [Shi *et al.*, 2008]. Moreover, the method proposed in [Huang *et al.*, 2011] is capable of synthesizing high resolution hand mesh deformation with rich and varying details, from only 14 examples poses. However, these approaches do not capture detailed soft-tissue deformations on a wide variety of body shapes. This limitation has been addressed by (Dyna, [Pons-Moll *et al.*, 2015]), which learns a model of soft-tissue deformations from examples using a high-resolution 4D capture system. Dyna captures surface deformations of the body at high spatial and temporal resolutions and constructs a mathematical model for relating these deformations to the motion and body shapes of novel characters.

The major drawback of example-based methods is the need for example poses. Besides the fact that when the example poses cannot be captured on a real actor, creating these poses requires either tremendous effort from an artist, or a complex physical simulation on a volumetric version of the skin mesh. In both cases, the mesh and its associated skeleton at rest are not sufficient, and further human intervention is required. An interesting discussion on example-based deformation methods, can be found in [Feng *et al.*, 2008]. In the next section, we discuss several volume preservation methods for skinned characters, which have been proposed to tackle the loss of volume artefact of linear skinning techniques.

2.2 Volume Preserving Skinning Methods

Volume preservation is an important aspect in the context of skin deformation that has been addressed in a variety of research papers over the last years [Desbrun & Gascuel, 1995; Guskov *et al.*, 1999; Botsch & Kobbelt, 2003; Sorkine *et al.*, 2004]. Volume preserva-

tion methods allow artists to correct the volume changes through the generation of extra bulges and/or wrinkles. The method that has been proposed in [Desbrun & Gascuel, 1995] is one of the first methods to introduce volume preserving deformation, where they use *local volume controllers* to guarantee volume conservation of implicitly described soft substances. Moreover, *multi-resolution methods* [Guskov *et al.*, 1999; Botsch & Kobbelt, 2003] can preserve surface details by decomposing a mesh into several frequency bands. Furthermore, Funck et al. presented an appealing approach that deforms the mesh vertices based on vector field integration [von Funck *et al.*, 2006; von Funck *et al.*, 2008]. However, these two methods are either computationally expensive or do not fit into the standard animation pipeline. Angelidis and Singh developed a skinning algorithm based on a powerful embedding into the volumetric space, which enables to preserve volume locally and globally [Angelidis & Singh, 2007]. In their method, a degree of freedom is left to the artist to control the final shape, although its combination with skinning weights variation along the mesh makes this control somewhat indirect. Recently, [Rohmer *et al.*, 2008; Rohmer *et al.*, 2009] presented an automatic volume correction method to model the constant volume behavior of soft tissues. It corrects the resultant deformations of LBS using a set of local deformations. In their work, they used an automatic way to segment an organic shape into a set of regions corresponding to the main muscle and fatty tissue areas, in which volume is computed and locally corrected (see Fig. 2.5). Huang et al. employed a nonlinear version of the volumetric graph Laplacian, which features nonlinear volume preservation constraints [Huang *et al.*, 2006]. Lipman et

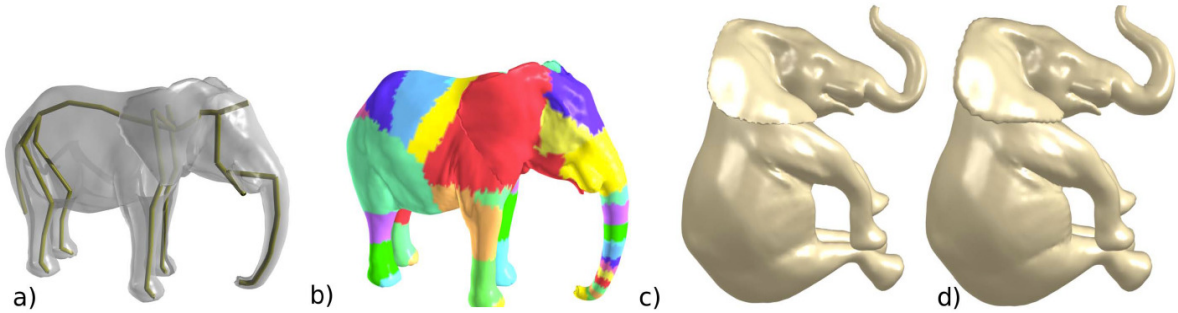


Figure 2.5: Illustration of the volume correction using the method presented in [Rohmer *et al.*, 2008] in a complex character. a) Skinned mesh and skeleton. b) Automatic segmentation. c) Standard LBS that suffer from the loss of volume. d) The volume correction method of [Rohmer *et al.*, 2008], where the volume is locally preserved in belly and trunk areas.

al. introduce a shape and volume preserving mesh editing technique [Lipman *et al.*, 2007b], where meshes are represented by moving frames. These frames are scaled during

deformation such that the volumetric shape properties are preserved. Several impressive works that create an inner scaffolding of spring, which resist compression to maintain volume are described in [Hong *et al.*, 2006; Zhou *et al.*, 2005]. The method in [Zhou *et al.*, 2005] provides an excellent introduction to these *interior lattice methods*. Lattice-Based freeform deformation (FFD) are widely-used in commercial software (such as Autodesk 3D Studio Max [Autodesk, 1990–2015] and Maya [Autodesk, 1998–2015]) for providing smooth deformations and preserving the volume of the skin [Sederberg & Parry, 1986; Bloor & Wilson, 1990; Coquillart, 1990; Milliron *et al.*, 2002]. For example, Autodesk Maya 2007 supports the notion of *flexors*. This lattice flexor uses a local FFD lattice, which can then be driven by joint transformations. However, the flexors do not support skinning transfer and the use of flexors can require significant setup and tweaking because of the multitude of lattice points. FFD was first formally proposed in [Sederberg & Parry, 1986] both as a representation for free-form solids and as a method for sculpturing solid models. Using FFD, a complex character can be deformed by positioning the control vertices of the coarse control grid. A more general extension of FFD (EFFD) was later presented by [Coquillart, 1990]. Moreover, Hsu et al. provided a method that directly manipulates the FFDs [Hsu *et al.*, 1992] and the method in [Aubert & Bechmann, 1997] uses an independent deformation function to provide a more flexible FFD. Although lattice-based methods give the artist the flexibility of creating the desired deformation, they require additional setup work and the deformation is sometimes difficult to predict.

On the other hand, cage-based skinning techniques consider an appealing way to control the deformation of an enclosed fine-detailed mesh and help to preserve the volume of skin deformations [Ju *et al.*, 2005; Joshi *et al.*, 2007; Lipman *et al.*, 2007a; Ju *et al.*, 2008; Savoye & Franco, 2010]. Cage-based techniques can be considered as a generalization of the lattice-based freeform deformation. Instead of a regular control lattice, a cage is defined by a fixed-topology control lattice that is fitted to the character skin. The cage can be seen as a low-resolution abstraction of the character, which enables the user to deform a character using a simpler mesh. Most cage-based deformation methods are special case of linear blend skinning, where the handle (cage vertex) transformations are restricted to be translations and the focus is on choosing the weights. The method presented in [Ju *et al.*, 2008] uses cage-based deformations to implement skinning templates, which offer a flexible design space within which to develop reusable skinning behavior. In their method, the skeleton drives the motion of the cage vertices using an example-based skinning technique, where the cage smoothly deforms the character model (see Fig. 2.6). Joshi et al. proposed a powerful cage-based deformation method based on harmonic coordinates

[Joshi *et al.*, 2007] for use in high-end character articulation. Their technique guarantees that the influence of each cage vertex is non-negative and falls off with distance as measured within the cage. It generates a pleasing deformation, but computing the harmonic coordinates is not easy. In spite of that, cage-based techniques allow smooth deformation of skin geometry. Posing the cage requires manual manipulation of the cage vertices. For an overview discussion on volume-preserving deformation methods, we refer the reader to [Gain & Bechmann, 2008; Nieto & Susin, 2013]. Another promising way to preserve the volume of the skin and to achieve realistic deformation is by applying physics-based simulation into the skin layer around the skeleton. The following section describes the vast literature on physics-based methods.

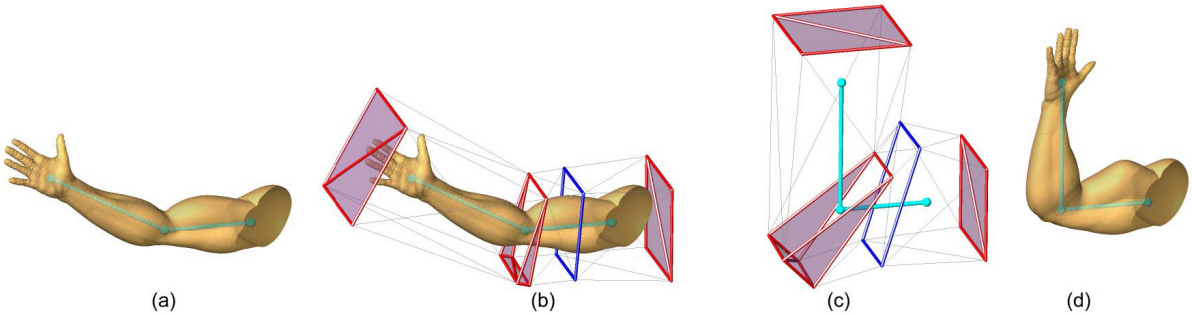


Figure 2.6: Skinning with cage: (a) Input geometry with skeleton. (b) An initial cage constructed from four templates, which are associated with the hand joint, elbow joint, upper arm bone, and the shoulder joint. (c) The skeleton deforms the mesh templates. (d) The geometry is deformed by the cage, yielding a non-pinching elbow and muscle bulging [Ju *et al.*, 2008].

2.3 Physics-based Methods

In order to model dynamic phenomena, such as the vibration of fatty tissues, muscles bulging and skin contact deformations due to collisions, the animator must configure the deformation for each keyframe. While manually posing a character for each animation keyframe allows artists to create such realistic effects, this process is tedious. Therefore, an alternative method is to employ physics into the skinning process, which highly enhances the believability and realism of character motions. Accordingly, physics-based simulation manages to bring skeleton-driven animation beyond the purely kinematic approach by simulating secondary motions, such as jiggling of soft tissues when the character is moving. Those secondary motions enrich the visual experience of the animation and are essential for creating appealing characters for movie productions and virtual

reality applications. After the pioneering work of Terzopoulos et al. [Terzopoulos *et al.*, 1987] and concurrently Lasseter's animation principle "*squash and stretch*" [Lasseter, 1987]. Physical simulation has taken an important role in the animated feature film industry and computer games [Goktekin *et al.*, 2007], where many physically based methods encouraged to simulate soft bodies or add dynamic effects to the skin. In the following subsections, we first discuss soft body simulations in (Section 2.3.1), and then physically based skinning methods in (Section 2.3.2).

2.3.1 Deformable and Soft Bodies Simulations

Simulating soft bodies can be achieved in different ways, and the design choice often has to balance the required accuracy and performance [Moore & Molloy, 2007]. The most popular techniques for simulating soft bodies in computer animation are force-based methods. In particular, most of the techniques used to simulate dynamics rely on mass spring systems, because of the simplicity and efficiency. The general idea is to represent the vertices of the mesh as mass points, governed by Newton's second law of motion, and the edges as elastic massless links (spring). Hence, the mesh is deformed when the lengths of the elastic links change. This happens when the relative position of the mass points changes due to external forces. Mass-spring systems are based on a local description of the material, in which the physics of such systems is straightforward and the simulator is easy to implement. However, to simulate a particular material, it is important to select carefully the parameters of the springs, such as the stiffness and damping. Despite that these systems are fairly easy to implement, they suffer from instability and overshooting problems under large time steps. Moreover, mass-spring systems are often not accurate, since they are strongly topology dependent and are not built based on elasticity theory. On the other hand, finite element methods (FEM) are based on elasticity theory, in which both the masses and the internal and external forces are lumped to the vertices. The vertices in the mesh are treated like mass points in a mass spring system while each element acts like a generalized spring connecting all adjacent mass points. Here, the physical material properties can be described using only few parameters that are used to model soft bodies in an accurate manner. Unlike mass-spring systems, finite element methods are easy to simulate for any particular material. This makes things easier for artists in charge of modelling different types of soft bodies. Unfortunately, finite element methods are avoided in real-time applications, because they are computationally expensive and complex to implement. Various methods have been proposed to address the drawbacks of mass-spring systems and finite element

methods [Chadwick *et al.*, 1989; Pentland & Williams, 1989; Gourret *et al.*, 1989; Turner & Thalmann, 1993; Lee *et al.*, 1995; Popović *et al.*, 2003; Larboulette *et al.*, 2005]. A comprehensive survey of Nealen et al. [Nealen *et al.*, 2006] provides the details about these techniques. The methods presented in [Terzopoulos *et al.*, 1987; Chadwick *et al.*, 1989] are the first to demonstrate the effectiveness of comparatively simple mass-spring based approaches. In their methods, they applied the Lagrangian equations of motion using a finite difference scheme to simulate elastic objects with regular parametrizations.

The concept of employing dynamic simulations into skinning for the purpose of character animation was introduced over two decades ago [Girard & Maciejewski, 1985], where many techniques were proposed to reduce the accuracy of the simulation to help improve performance and interactivity. Capell et al. [Capell *et al.*, 2002] used a volumetric finite element mesh to represent the deformation of skin, driven by the underlying skeleton motion. They extended their method to include rigging forces, which guide the deformation to a desired shape [Capell *et al.*, 2005]. In their method, they effectively handled the effect of skin movement by using skeletal constraints, but by using forces that can violate the conservation of momentum makes their simulation unstable under large time steps. Shinar et al. [Shinar *et al.*, 2008] presented a framework of a two-way coupling between rigid and deformable bodies, in which they use a time integration scheme for solving dynamic elastic deformations in soft bodies interacting with rigid bodies. However, their method does not facilitate the development of an interactive animation system, because of the massive computation required for the finite elements representing the deformable body. In contrast, a possible way to accelerate the simulation of soft bodies is to focus on the surface rather than the volume [Bro-nielsen & Cotin, 1996; James & Pai, 1999; Galoppo *et al.*, 2007]. In particular, Galoppo et al. [Galoppo *et al.*, 2007] presented a fast method to compute the skin deformation on the surface of a soft body with rigid core. Their formulation only considers the elastic energy from skin-layer deformation, and does not include the deformation inside the volume. This may lead to inaccuracies when capturing pose-dependent deformations.

All the above methods are only valid for small deformations and are unsuitable for an articulated character's large deformations. On the other hand, Müller et al. [Müller *et al.*, 2002] achieved a good real-time performances for large rotational deformations, by using a pre-computed linear stiffness matrix to generate the deformations; their method is simple and rotationally invariant. Recently, Kim and Pollard [Kim & Pollard, 2011] proposed an approach relying on finite element method to simulate the skin deforma-

tion, able to handle both one-way and two-way simulations. Their method generates compelling dynamic effects, and the deformations are obtained at near interactive rates. The method proposed in [Gilles *et al.*, 2011a] simulates dynamic skinned deformation models using frame-based degrees of freedom with unreduced force evaluation. Jain and Liu presented a robust approach that realistically simulate characters with soft tissue at the site of contact, where they used two-way coupling between articulated rigid bodies and deformable objects [Jain & Liu, 2011]. Liu et al. developed a framework that simulates and controls skeleton driven soft body characters [Liu *et al.*, 2013a]. Their method couples the skeleton dynamics and the soft body dynamics to enable two way interactions between the skeleton, the skin geometry, and the environment at interactive rates (as we can see in Fig. 2.7).

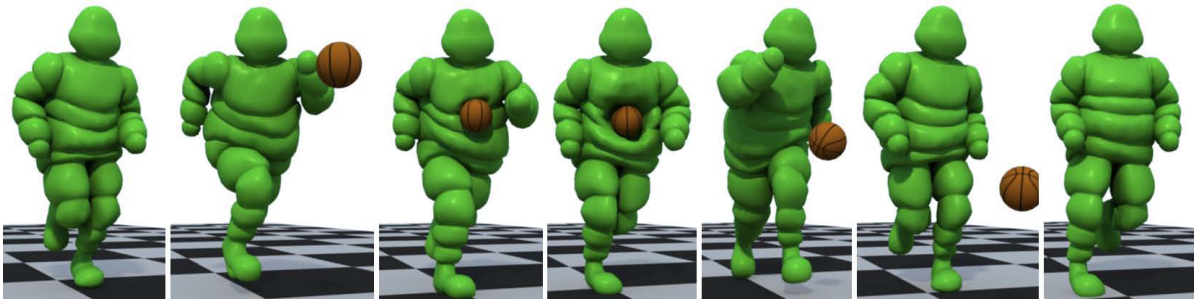


Figure 2.7: Employing dynamic simulation into skinning process allows two-way interactions between the skeleton, the skin geometry, and the environment at interactive rates [Liu *et al.*, 2013a].

2.3.2 Physically based Skinning

Physics-based methods are the natural choice for creating secondary motion effects such as flesh jiggling when a character is moving [Turner & Thalmann, 1993; Wilhelms, 1994; Lee *et al.*, 2009; Gilles *et al.*, 2011b; Hahn *et al.*, 2013]. Turner and Thalmann model the elasticity of skin for character animation and simulate the fat layer by Hookian spring forces [Turner & Thalmann, 1993]. However, they treat muscles as purely controlled elements. Thus, they do not model muscles with deformable methods. Wilhelms [Wilhelms, 1994] presented an approach for animated animals by simulating individual bones, muscles, soft tissues and skin. The use of muscles, soft tissues and flesh elements makes it hard to fit this approach into the skinning framework. Moreover, Hahn et al. [Hahn *et al.*, 2012; Hahn *et al.*, 2013] generated secondary skin dynamics based on the rig degrees of freedom (see in Fig. 2.8). Their methods simulate the deformation of a

character's fat and muscles in the nonlinear subspace induced by the character rig. In the other direction, Kim and James [Kim & James, 2011] proposed a domain-decomposition method to simulate articulated deformable characters entirely within a subspace framework, where they combined locally-rotated nonlinear subspace models to simulate the detailed deformations of the models. In order to simulate the musculotendons of the human hand and forearm, [Sueda *et al.*, 2008] add anatomic detail using the tendons and bones.

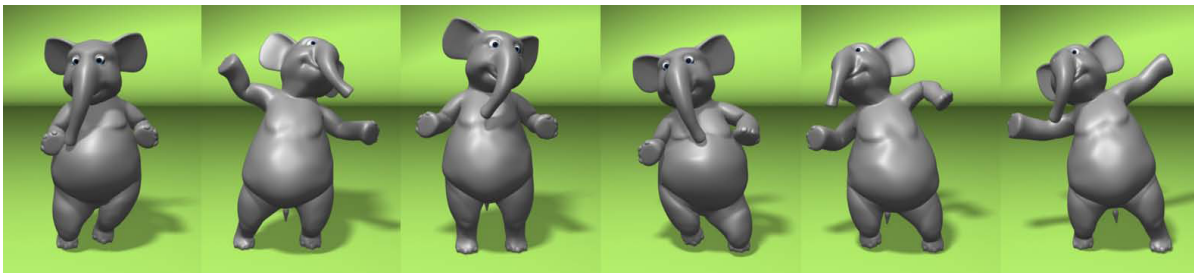


Figure 2.8: The method proposed in [Hahn *et al.*, 2012] takes a character rig as an input and automatically produces physically plausible motions, which maintains the original artistic intent and is easily editable.

While physics-based skinning methods can automatically generate secondary motion with high visual quality, they entail a significant computational burden that slows production and prohibits its use in interactive environments. McAdams et al. [McAdams *et al.*, 2011] presented a robust method using a uniform hexahedral lattice, which provides convincing deformations of the skin with contact handling. In addition, they introduce a one-point quadrature scheme and a multi-grid solver in order to improve the performance and stabilize the simulation.

Although their method can capture appealing skin deformations and guarantee pinch-free geometry, it works at best at near interactive performance (as we can see in Fig. 2.9). Recently, Deul and Bender [Bender *et al.*, 2013] introduced a multi-layer character skinning based on shape matching with oriented particles, used to simulate the elastic behavior of a closed triangular mesh as a representation of a skin model. They make a use of position-based constraints for coupling the skeleton with the skin and handling self-collisions. Lastly, Gao et al. [Gao *et al.*, 2014] proposed a physics-based skinning method for skeleton-based characters. They introduced a material model of ex-rotated elasticity, which uses a procedural skinning technique to approximate co-rotated elasticity with an affine force model and pose-dependent coefficients. Their method can

simulate high-resolution human flesh models with full external and self-collision processing, without the ability to generate secondary motions effects like jiggling of the fat tissues.

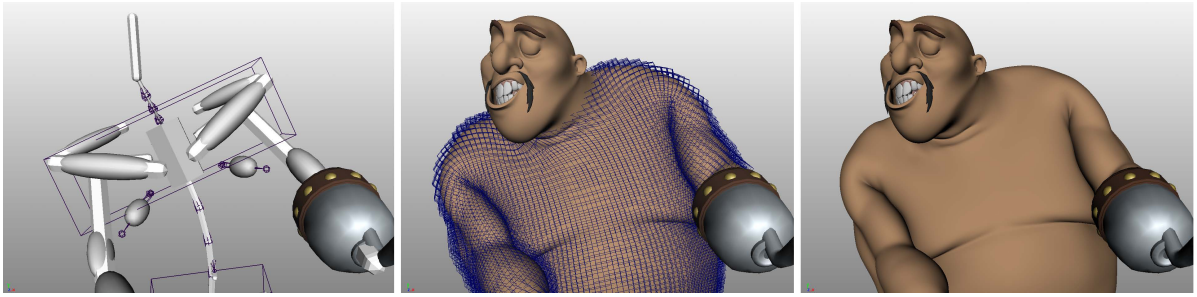


Figure 2.9: The method of [McAdams *et al.*, 2011] takes a skeleton and a surface mesh as input. Based on a hexahedral lattice with 106,567 cells (*center*), their method simulates the deformed surface (*right*) obeying self-collision and volumetric elasticity at 5.5 seconds per frame.

2.4 Collision Handling for Deformable Bodies

In order to simulate the skin's behavior realistically, an appropriate collision response has to be considered. However, skin contact deformations due to collision are difficult to model using geometric skinning techniques, because the combination of bone transformations and blend weights, completely determine the resulting (deformed) mesh. In contrast, physics-based methods can capture skin contact deformations. In these methods, collision handling consists of two steps: collision detection and collision response. Often they are performed independently, with the result of the collision detection algorithm being used as input to the collision response algorithm. Both collision detection and collision response are challenging problems and active areas of research in computer graphics. In the following sections, we discuss literature that addresses these problems.

2.4.1 Collision Detection

Because of its importance, a substantial amount of research in computer animation is related to collision detection. In order to achieve interactive performance of collision detection, efficient data structures based on bounding volume hierarchies (BVHs) and spatial partitioning have been widely used. In this section, we mainly focus on those methods that detect collisions and self-collisions on deformable bodies or models undergoing specific types of deformation, such as skeletal deformation. For a more thorough

treatment of the collision detection literature, we refer the reader to the following surveys [Jiménez *et al.*, 2000; Lin & Manocha, 2003; Ericson, 2004; Teschner *et al.*, 2005].

2.4.1.1 Bounding Volume Hierarchies

Bounding volume hierarchies (BVHs) have been commonly used to accelerate collision detection algorithms. By using a BVH, the time complexity of a geometric query can be reduced from $\mathcal{O}(n)$ to $\mathcal{O}(\log n)$, where n is the number of object primitives. Many types of bounding volumes (BVs) have been investigated, e.g., spheres [Spillmann *et al.*, 2007], axis-aligned bounding boxes (AABBs) [Bridson *et al.*, 2002], oriented bounding boxes (OBBs) [Gottschalk *et al.*, 1996], discrete oriented polytopes (DOPs) [Klosowski *et al.*, 1998], boxtrees [Zachmann, 1995]. Widely used hierarchies are those based on simple BVs such as spheres and AABBs [He *et al.*, 2015]. For a comprehensive discussion on bounding volume hierarchies, we refer the reader to [Zachmann & Langetepe, 2002]. In contrast to rigid objects, deformable bodies change their shapes almost at each time step of the simulation. Therefore, their hierarchies must be updated accordingly and the cost of these updates can be high. Traditional approaches require $\mathcal{O}(n)$ operations for each update, where n is the number of object primitives. Several techniques have been proposed to speed up the updates, including refitting algorithms [van den Bergen, 1998; Larsson & Akenine-Möller, 2001; Larsson & Akenine-Möller, 2006; Zachmann & Weller, 2006] and dynamic restructuring [Otaduy *et al.*, 2007]. In order to further reduce the computational complexity, on-demand refitting algorithms have been proposed. These algorithms exploit information provided by the deformation model, in which bounding volumes are only recomputed when required by the collision detection algorithm. Such refitting is presented in [James & Pai, 2004] for the reduce deformation model, in [Larsson & Akenine-Möller, 2003] for morphing, in [Kavan & Zara, 2005] for linear blend skinning and in [Kavan *et al.*, 2006] for spherical blend skinning (see Fig. 2.10). Another interesting collision detection method specialized for skeletally deformable 3D models can be found in [Heim *et al.*, 2004], but their refitting procedure is restrictive especially for detailed 3D models.

On the other hand, deformable bodies often suffer collisions against themselves, and detecting self-collisions is usually a bottleneck for real-time simulations. Self-collision queries with BVHs are executed by starting a recursive query on the root BV against itself. Unfortunately, BVHs lose all their advantages in this case, because tests between triangles adjacent to each other cannot be culled away at high levels in the hierarchy. Thus, detecting the self-collisions using BVHs is computationally expensive. Moreover,

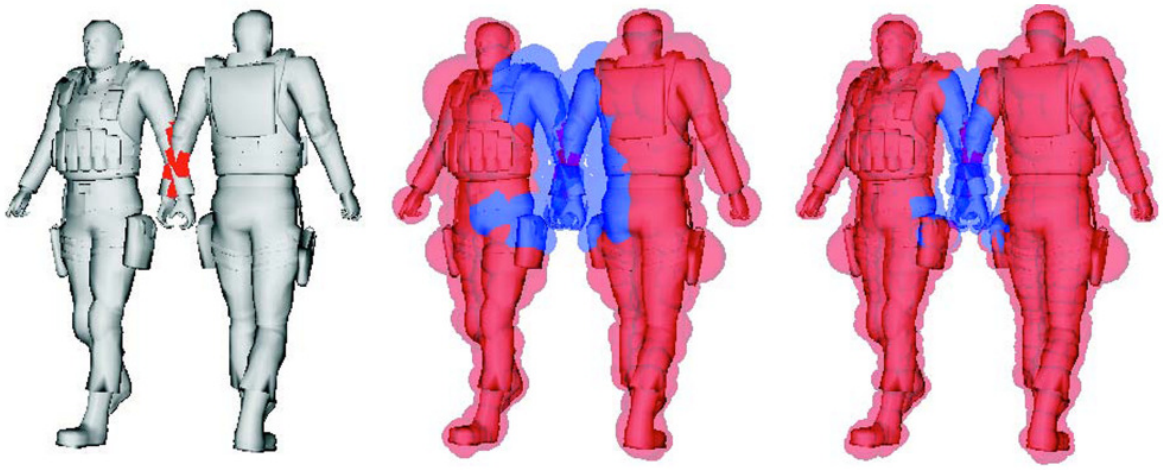


Figure 2.10: Fast collision detection method introduced in [Kavan *et al.*, 2006]. Their CD algorithm is based on BVH, in which they produce a tree with 5 levels (*center*) and 6 levels (*right*) to detect collisions on models deform by spherical blending skinning.

BV approaches typically detect the intersections, but they require an additional processing step to compute the penetration depth for collision response. In order to address these problems, various approaches have been introduced, which are based on surface normals and curvature [Volino & Thalmann, 1994; Provot, 1997; Mezger *et al.*, 2003; Schvartzman *et al.*, 2009] or based on star-shaped decomposition [Schvartzman *et al.*, 2010]. An efficient algorithm to handle self-collisions is proposed in [Govindaraju *et al.*, 2005], where they employed both visibility-based culling and chromatic decomposition to radically improve the performance of self-collision detection using BVHs. For an extensive description and analysis of the use of BVHs for collision detection, please refer to Gottschalk's PhD dissertation [Gottschalk, 2000].

2.4.1.2 Spatial Subdivision Partitioning

Spatial subdivision partitioning methods are simple and fast approaches to speed up collision and self-collision detection on deformable bodies. These methods have similarities with bounding volume hierarchies. However, the idea here is subdividing the space instead of the objects. Several spatial subdivision schemes have been proposed for collision detection, such as octrees-like structures [Madera *et al.*, 2006], BSP trees [Luque *et al.*, 2005], kd-trees [Teller & Sequin, 1991], uniform grids [Zhang & Yuen, 2000] and spatial hashing [Turk, 1989; Teschner *et al.*, 2003; Alcantara *et al.*, 2009]. An important structure to be highlighted here is spatial hashing, because it has become widely used for collision detection with deformable bodies [Sud *et al.*, 2006; Eitz &

Lixu, 2007]. Instead of using complex data structure and explicitly performing a spatial subdivision, a hash function is used to map 3D cells into a hash table. In [Maciel *et al.*, 2007] a spherical hash is used to detect collisions on biomechanical models. Despite that this method has shown a fast performance in the main processing stage, there is a drawback: In case the objects deform in a non-radial direction, the pre-processing stage must be repeated, which is slow. Jund et al. presented an efficient collision detection method based on spatial subdivision, which supports geometric and topological changes on deformable bodies [Jund *et al.*, 2009].

2.4.2 Collision Response

In order to produce convincing skin deformations and organic effects such as contact deformations between skin parts (especially in the areas around a joint), collisions between skin parts must be handled. However, resolving collisions requires specific information, and it is not sufficient to just detect the collisions. Nevertheless, collision detection is the first step in resolving the dynamic behavior of skin in contact and it has a strong influence on the continuity of a collisions response algorithm. Hence, collision detection algorithms have to provide extra information such as penetration depth, which is required for realistic collision response. Accurate collision handling for an articulated body with j bones and p contacts has $\mathcal{O}(jp)$ complexity [Baraff, 1996]. Therefore, handling collision would slow down the computation, but the resulting deformations would be more convincing.

Several approaches have been proposed to address the problem of efficiently handling global and local collision response [Capell *et al.*, 2005; Jain & Liu, 2011; Ye & Liu, 2012]. In particular, Capell et al. presented a framework that imposes constraints on a control lattice for FEM simulation [Capell *et al.*, 2005]. Their framework has been extended to include rigging force fields, self-collision handling, and automatic linearization of deformations in pose space. In contrast, one broadly applied technique to handle collision is to formulate a linear complementarity problem (LCP). The formulation of the LCP uses an implicit time-stepping method and robustly guarantees the enforcement of the constraints at the end of the time step [Stewart & Trinkle, 1996]. In this context, Erleben proposed an LCP formulation [Erleben *et al.*, 2005], in which he combines joint constraints, joint limits, and joint motors with rigid contact constraints in a velocity based linear complementarity for the purpose of contact modelling.

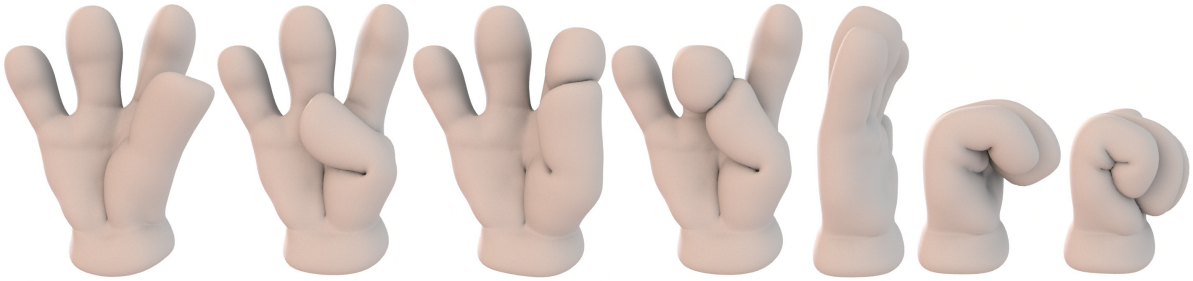


Figure 2.11: The contact handling algorithm presented in [Teng *et al.*, 2014] allows both self-collision detection and contact response to be simulated in 5.8 FPS (171 ms) for a hand mesh composed of 458K tetrahedra.

Recently, Teng et al. presented an efficient subspace method for simulating self-contact of articulated deformable bodies ([Teng *et al.*, 2014], see Fig. 2.11). They explored self-collision in pre-computation schemes, but not on skinned bodies. More recently, Sheth et al. proposed a framework for simulating reduced deformable bodies, which accounts for linear and angular momentum conservation [Sheth *et al.*, 2015]. Their framework fully supports collision, contact, articulation, and other desirable effects.

2.5 Conclusion

In this chapter, we have reviewed the common techniques for modelling deformations, especially those for character animation purposes. A greater attention was paid to skeleton-based methods, and physics-based methods. In skeleton-based skinning such as linear blend skinning, dual quaternion skinning and pose space deformation, surface deformation is restricted to the skeletal pose that fully defines the surface deformation. While skeleton-based methods can produce good results, the believability of the deformation using these methods is limited. Because they cannot capture secondary motions effects and skin contact deformations. In contrast, physics-based simulations bring skeleton-based animation beyond the purely kinematic approach by simulating secondary motions such as jiggling of soft tissues when the character is moving, as well as capturing skin contact deformation. Simulating such flesh-like deformations is difficult due to the coupling between the skeleton and soft skin. Moreover, the resultant deformation has a high number of independent degrees of freedom, in which it does not respect any manipulation done by the artist. Therefore, once the deformation parameters are specified, it is difficult to control the actual resulting shape of the character in every animation frame. Furthermore, physics-based methods are computationally expensive and usually avoided in interactive applications.

The goal of this dissertation is to develop methods that can interactively capture high quality skin deformation in a plausible way at a low computational. In the next chapter, we recapitulate the mathematical concept and essential background information that will be necessary for understanding and reproducing the technical contributions in the following chapters.

CHAPTER 3

LAYERED SKIN DEFORMATION

To put this dissertation in a proper perspective, it is important to give some insight into the background material, mathematical descriptions and the computational frameworks for skeleton-based deformation and physically based soft-body animation. In this chapter, we provide the context for these topics, which serves as the basis for the developments presented in following chapters. Moreover, the main aim of this chapter is to introduce the reader to the concept of *layered skin deformation*. The general idea of layered skin deformation is to create a chain of layered deformations, the result of which approximates the behavior of the skin. Typically, character meshes are deformed with a single skeleton-based skinning algorithm. However, traditional skeleton-based deformation methods produce unsightly artefacts and fail to capture the full range of desired effects, namely jiggling, volume preservation, muscle bulging and skin contact deformations. To enforce elastic behavior and preserve the volume of the skin, new deformer types can be developed and integrated with the traditional skeleton-based deformer. This combination of simple deformers is sufficient for reproducing the macroscale volume changes observed in moving creatures.

First, we briefly introduce the concept of layered skin deformation (Section 3.1). Then, we discuss the fundamental theories of skeleton-based deformation (Section 3.2) and

review the basic definitions and formulations in physics-based deformation methods (Section 3.3). This chapter also serves as a quick tutorial to physically based approaches. For readers who are interested in more details on these subjects, we would like to refer them to [Dong *et al.*, 2002; Erleben *et al.*, 2005; Kenwright, 2012; Sifakis & Barbic, 2012; Bender *et al.*, 2015].

3.1 Layered Skin Deformation Framework

The skinning process for articulated characters is an indispensable component of the content creation pipeline for feature films, for video games, and in the special effects industry. In character animation, both the skeleton of a virtual character and the skin that defines its external shape are required. Therefore, the deformation of the skin is computed in every animation frame and depends on the actual positions of skeletal joints. Lasseter and Chadwick emphasized that computers provide the advantage of working with an animation in layers [Lasseter, 1987; Chadwick *et al.*, 1989]. The ability to additively build an animation in layers provides an effective means for creating complex motion. Therefore, the layered deformation framework is commonly employed in animated films and computer games. Layered deformers use a stack (or chain) of deformers to compute the final deformation of the skin shape. For simple deformations, this stack may have only one deformer, which is usually purely kinematic that binds the skeleton-to-skin in a geometrical way. However, to provide more life-like deformations, it is important to add more physically based deformers.

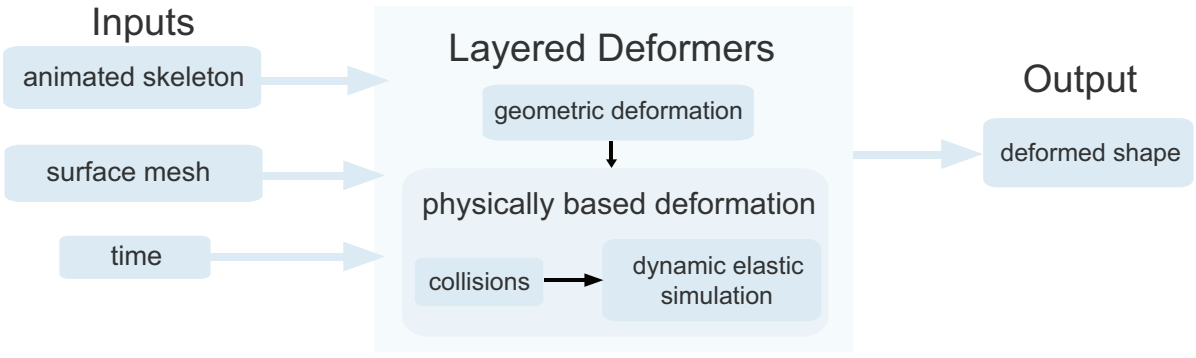


Figure 3.1: Layered skin deformation composed of two-layered deformers. The first layer modifies the skin shape in response to the changes in skeleton position. While in the second layer, the skin is deformed using physically based deformer, which is mainly used to simulate the elasticity of the skin and contact reaction.

The design of this stack allows for concatenation of multiple types of deformers, in which each deformer in the stack operates on the output of the previous deformer (see Fig. 3.1). Consequently, this framework offers a compact way for modelling both geometric deformations and the physically realistic behaviors of the skin, which is also capable of processing collisions and contacts. Moreover, it gives the artists the freedom to create a flexible combination of deformations so as to achieve almost any desired effect. This concept is also easy to implement: the mesh vertices are passed through a linear series of deformations, and each of which adds some contributions to the final position of the vertices. The following pseudo-code demonstrates how to develop a simple two-layered skin deformation (see Alg. 1). Despite the simplicity, there is an obstacle that prevents adaptation of this concept into real-time applications: The impediment is the lack of understanding of how to exactly construct such a system that achieves visually pleasing deformations, while keeping the computation cheap.

Algorithm 1: A sample pseudo-code for creating two-layered skin deformation, consisting of a skeletal deformer and an elastic deformer.

Input : *Vector* inputVertices, *Matrices4x4* boneTransformations
Output : *Vector* deformedVertices

```
1: Establish nextDeformer as stack of [skeletalDeformer,elasticDeformer]
2: updateSkin ()
3: {
4: Deform(inputVertices,boneTransformations)
5: if nextDeformer then
6:    nextDeformer->updateSkin()
7: end if
8: }
9: skeletalDeformer(inputVertices,boneTransformations)
10: {
11: for each v_i in inputVertices do
12:    // binding each vertex only to the nearest bones
13:    bindSkin(v_i, closestBones)
14: end for
15: }
16: elasticDeformer(inputVertices)
17: {
18: // elastic layer, which adds physics-based elastic components to improve
19: // the realism. A simple example is the jiggly-looking skin of the fat
20: // tissues
21: jiggle(v_i)
22: }
```

The most intuitive deformation method to model the skin of a virtual character is via its underlying skeleton. In the following, we present the fundamentals of skeleton-based deformations in details.

3.2 Skeleton-based Deformations

The fundamental technique to drive the deformation of a character skin is via an underlying skeleton. Among the many proposed skeleton-based deformation techniques, linear blend skinning (LBS) is the most popular technique due to its effectiveness, simplicity, and efficiency [Magnenat-Thalmann *et al.*, 1988]. It has been given many different names over the years, including "*skeletal subspace deformation*", matrix palette skinning, enveloping, vertex blending, smooth skinning (Autodesk Maya), bones skinning (Autodesk 3D Studio Max), or linear blend skinning (the open-source Blender) [Le & Deng, 2012]. What follows is an explanation of both the concepts and mathematics necessary to understand and implement linear blend skinning. We will also review the algorithm and its limitations.

3.2.1 Linear Blend Skinning

In linear blend skinning, the basic operation is to deform the skin according to a given list of bone transformations (Fig. 3.2). The formulation of the LBS model requires the following input data:

- **Surface mesh**, a 3D model represented as a polygon mesh, where only vertex positions will change during deformations. We denote the rest pose vertices as $\mathbf{v}_i$, ... ,$\mathbf{v}_n$, and $\mathbf{v}_i \in \mathbb{R}^4$ is given in homogeneous coordinates.

- **Bone transformations**, representing the current deformation using a list of matrices $\mathbf{T}_1$, ... ,$\mathbf{T}_m \in \mathbb{R}^{4\times4}$. The matrices can be conveniently defined using an animated skeleton. $\mathbf{T}_j$ is the transformation matrix associated with bone j in its current (animated) pose, $\mathbf{R}_j^{-1}$ is the inverse transformation of the same bone in the rest pose $\mathbf{R}$. Additionally, the matrix $\mathbf{R}$ remains constant, so its inverse is also constant and can safely be pre-computed. Each bone transformation influences part of the mesh.

- **Skinning weights**, For each vertex $\mathbf{v}_i$, we have weights $w_{i,1} + ... + w_{i,m} \in \mathbb{R}$, in which each $w_{i,j}$ describes the amount of influence of bone j on vertex $\mathbf{v}_i$. The

binding weights are normally assumed to be convex, so $w_{i,1} + ... + w_{i,m} = 1$ and $w_{i,j} \geq 0$.

The surface mesh is driven by a set of bones. Every vertex is associated with the bones via a bone-vertex bind weight, which quantifies the influence of each bone to the vertices. Therefore, the basic idea behind LBS is to linearly blend the transformation matrices. The skin is deformed by transforming each vertex through a weighted combination of bone transformations from the rest pose. Thus, the final transformed vertex position $\mathbf{v}'_i$ is a weighted average of its initial position transformed by each of the attached bones. The whole process can be expressed with the following equation [Merry *et al.*, 2006]:

$$\mathbf{v}'_i = \sum_{j=1}^{m} w_{i,j} \mathbf{T}_j \mathbf{R}_j^{-1} \mathbf{v}_i \tag{3.1}$$

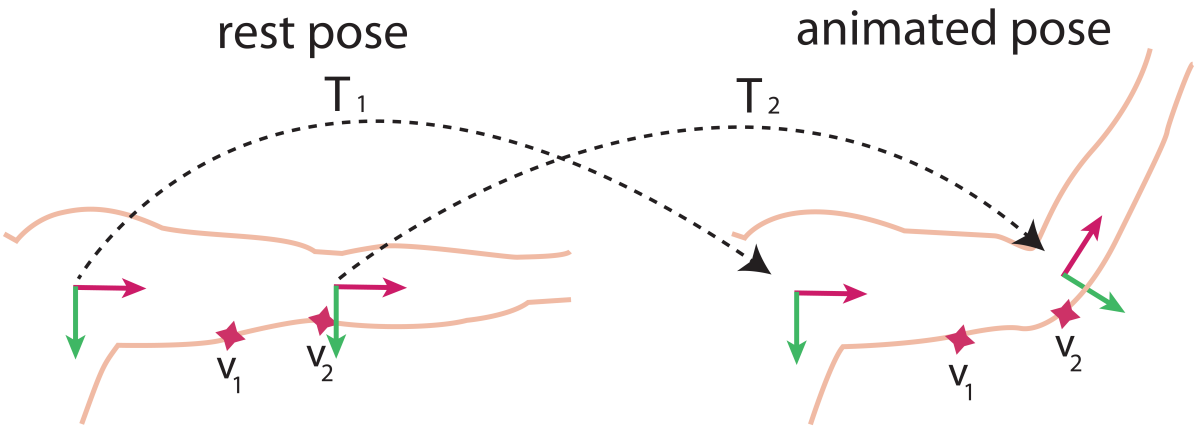


Figure 3.2: An example illustrates the main concept of LBS. There are two transformations $\mathbf{T}_1$ and $\mathbf{T}_2$, corresponding to the transformations of shoulder and elbow joints from the rest pose to an animated posture.

In practice, it is rarely makes sense to attach a vertex to more than four joints. However, some old games systems used a variant dubbed rigid binding, which corresponds to allowing only one influencing bone per vertex. *Rigid binding* leads to unrealistic non-smooth deformations near joints and suffers from self-intersections. With increasing polygon budgets, linear blend skinning quickly replaced rigid binding because it allowed for smooth transitions between individual transformations (see Fig. 3.3). Linear blend skinning begins by assigning weights for every vertex on the skin mesh to the underlying bones. The binding weights can be constructed automatically based on the distance between the skin vertices and line segments representing the bones, but it is hard to

reliably create good weights automatically. Usually the artists must paint the weights on the mesh directly, using their knowledge of anatomy[1]. Technically, the implementation of a linear blend skinning algorithm is straightforward, and a pseudo-code is presented in Alg. 2. Linear blend skinning works very well when the blended transformations $\mathbf{T}_j$ are not very different. Issues arise if we need to blend transformations, which differ significantly in their rotation components.

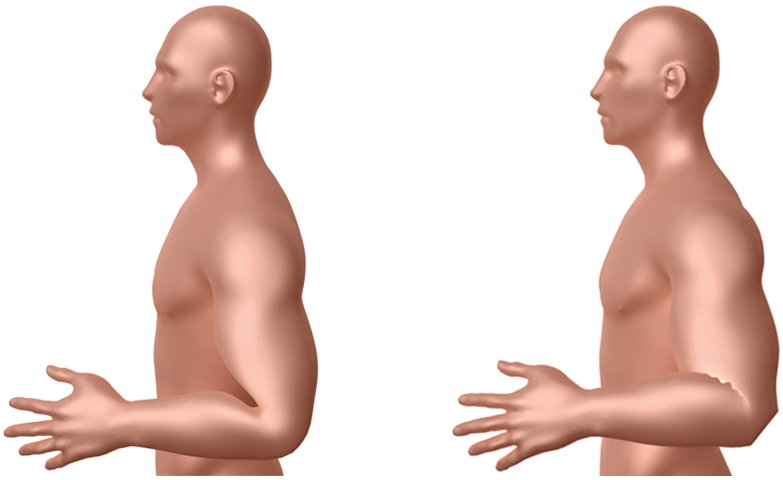


Figure 3.3: *Left.* Linear blend skinning. Note the loss of volume at the elbow joint. *Right.* Rigid binding. Note the self-intersections and unnatural deformations in the areas around the elbow joint.

Algorithm 2: Simple pseudo-code of the linear blend skinning algorithm

Input : *Vector* inputVertices, *Matrices4x4* currentbonesTransformations, *Matrices4x4* restbonesTransformations, *Matrices4x4* bindingweights

Output : *Vector* deformedVertices

```
// Apply current animation (bone transform) to the rest pose to get the final
// skinning matrices
for each j in Bones do
    skinningmatrices_j = currentbonesTransformations ×
    restbonesTransformations_j^-1
end for
// Loop through every vertex and compute the blended position
for each v_i in inputVertices do
    deformedVertices_i = inputVertices_i + bindingweights_i,j ×
    skinningmatrices_j
end for
```

Despite its fast and straightforward implementation, linear blend skinning suffers from

[1] In Chapter 4, we describe how binding weights can be efficiently computed.

some visible artefacts when we rotate joint more than 90°. In a rotating joint, we expect the skin to rotate around the joint too, maintaining the volume. But the linear blend model instead interpolates skin vertex positions linearly between where the bones expect them, which shrinks the volume of the skin. Therefore, this linearly blending rotation leads to the well-known "*candy-wrapper*" artefact (as we can see in Fig. 3.4).

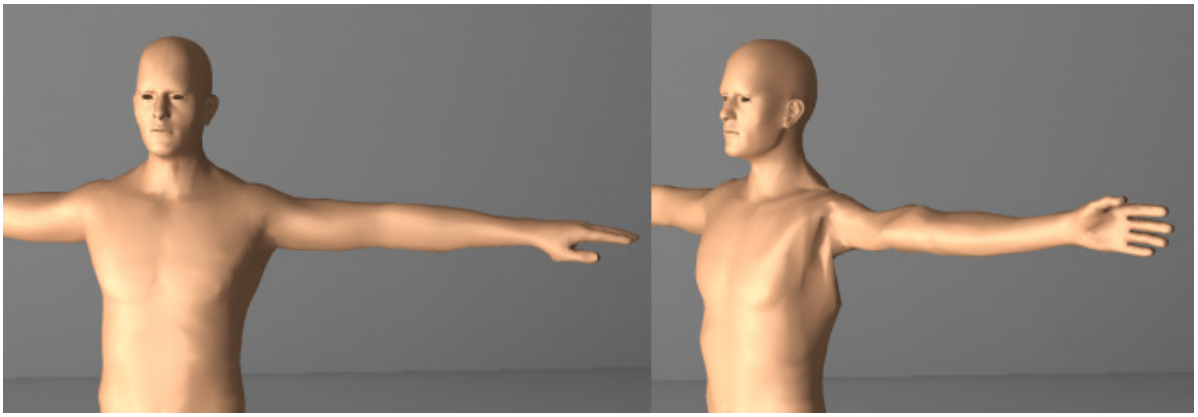


Figure 3.4: The well-known "*candy-wrapper*" artefact of linear blend skinning. *Left.* The character model in its rest pose. *Right.* The model deformed with linear blend skinning, where the areas around the shoulder joint suffer from the "*candy-wrapper*" artefact and volume loss when twisting.

Fig. 3.5 below illustrates the problems of LBS, which are loss of volume when bending and the "*candy-wrapper*" artefact when twisting:

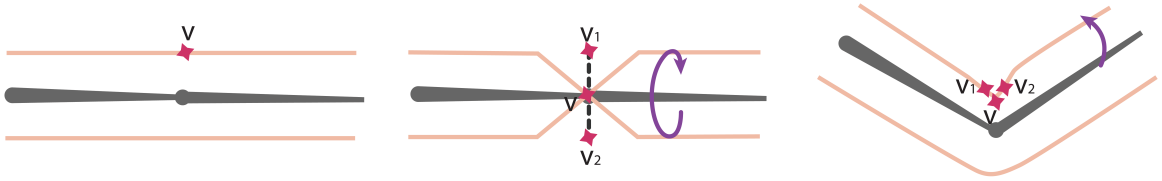


Figure 3.5: *Left to right:* The skin in its rest pose. Rigid transformations (express rotation and translation). While twisting, the weighted combination of vertices $\mathbf{v}_1$ and $\mathbf{v}_2$ is not guaranteed to be a rigid transformation, which result in the "*candy-wrapper*" artefact. When bending, the linear interpolation of LBS between the vertices $\mathbf{v}_1$ and $\mathbf{v}_2$ produces $\mathbf{v}$ at an inadequate location, which result in a loss of volume.

Fixing these artefacts requires some intervention. One possibility is to improve the skinning algorithm itself. One of most the popular improvements based on geometric algebra, is called dual quaternion skinning, which completely eliminates these artefacts. We discuss this algorithm in the next section.

3.2.2 Dual Quaternion Skinning

To avoid the "*candy-wrapper*" effect and volume loss of linear blending skinning (LBS), dual quaternion skinning (DQS) [Kavan *et al.*, 2007]) is a good alternative. DQS delivers a rigid transformation in all cases and it is almost as fast as LBS. While the underlying mathematics may not be trivial, an actual implementation of dual quaternion skinning is quite straightforward. Instead of using matrices to express the rigid transformations of the bones, dual quaternion skinning uses the geometric algebra of quaternions. The formulation of DQS is based on dual quaternions [Clifford, 1882], which is a generalization of regular quaternions that can be used to express both the translation and the rotation. We are blending rigid transformations, however, the blended matrices $\mathbf{M}$ are no longer rigid transformations under rotations, but general affine transformations (potentially containing scale and shear factors). Therefore, instead of blending matrices $\mathbf{M} = \sum_{j=1}^{m} w_{i,j}\mathbf{T}_j\mathbf{R}_j^{-1}$. In DQS, we blend dual quaternions $\hat{\mathbf{q}}_\mathbf{j}$. The figure below illustrates the intuition of why DQS is better than LBS.

Figure 3.6: *Left.* Linear blend skinning, which computes a linear interpolation between two vertices and the new position will be somewhere lying on the line segment between $\mathbf{v}_1$ and $\mathbf{v}_2$. *Right.* Dual quaternion skinning, where instead of a linear interpolation it is a spherical one. The new position will be lying on the arc circle, and will avoid volume loss [Kavan *et al.*, 2007].

To compute the deformed position of a vertex with DQS. First, the transformation matrices are converted to unit dual quaternions. Then, these unit dual quaternions are blended linearly, similarly to linear blend skinning. In other words, we blend dual quaternion transformation $\hat{\mathbf{q}}_\mathbf{j}$, weighted by $w_{i,j}$. The result is normalized with $\| \sum_{j=1}^{m} w_{i,j}\hat{\mathbf{q}}_\mathbf{j} \|$ to produce the final dual quaternion used to transform a vertex from its rest pose to the deformed position. This guarantees to represent only a rotation and translation. While linearity is lost (due to the projection on unit dual quaternions), the resulting algorithm can be implemented very efficiently. Dual quaternion skinning computes deformed vertex positions according to the following formula (a more detailed explanation on dual

quaternion skinning can be found in [Kavan *et al.*, 2008]):

$$\mathbf{v}'_i = \hat{\mathbf{q}}_\mathbf{j} \mathbf{v}_i \hat{\mathbf{q}}_\mathbf{j}^* \tag{3.2}$$

where $\hat{\mathbf{q}}_\mathbf{j} = \frac{\sum_{j=1}^{m} w_{i,j} \hat{\mathbf{q}}_\mathbf{j}}{\| \sum_{j=1}^{m} w_{i,j} \hat{\mathbf{q}}_\mathbf{j} \|}$ is a unit dual quaternion and $\hat{\mathbf{q}}_\mathbf{j}^*$ is the conjugate of $\hat{\mathbf{q}}_\mathbf{j}$.

Unlike matrices, by using dual quaternions there will not be any scale factor that shrinks the mesh around the joints. Dual quaternion skinning thus successfully eliminates the "*candy-wrapper*" effect, but it reveals its own artefact, called joint-bulging artefact (as we can see in Fig. 3.7). Moreover, dual quaternion skinning has a number of limitations especially in a production environment, as dual quaternions are unable to represent non-rigid transformations, such as shearing.

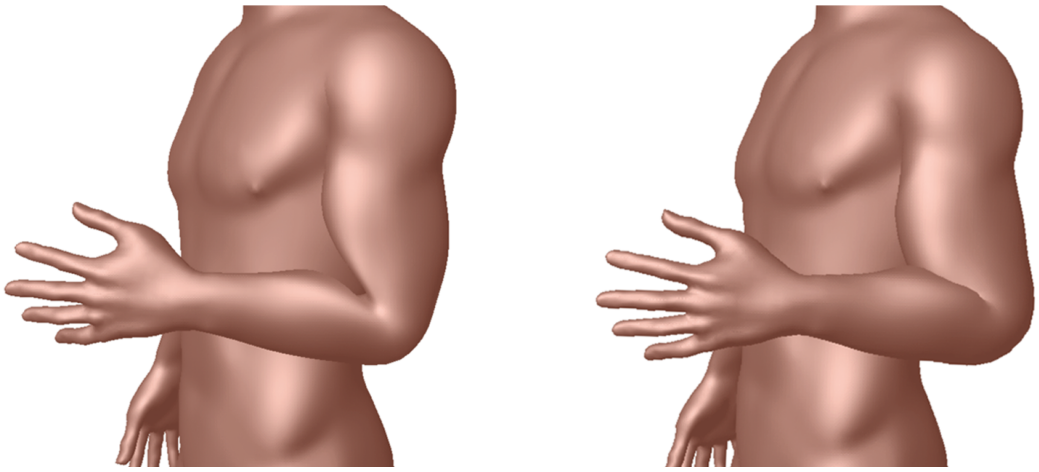


Figure 3.7: A demonstration of the artefacts of linear blend skinning (LBS) and Dual quaternion skinning (DQS). *Left.* LBS suffers from volume loss while bending. *Right.* DQS successfully eliminates the "*candy-wrapper*" effect and preserve the volume of the skin, but produces the joint-bulging artefact while bending.

Skeleton-based deformation alone is not sufficient for capturing believable skin deformations, such as skin stretching, secondary motion effects and skin contact due to collisions. In contrast, physics-based simulation offers believable and compelling deformations and enables to create physically based secondary motions, such as flesh jiggling and realistic collision reaction. The next section elaborates on the main physically based methods for simulating dynamic behavior of articulated characters.

3.3 Physics-based Deformation

Popular physics-based simulations for producing efficient, realistic-looking animations for computer graphics applications, including physics engines (constraint-based techniques) [Müller *et al.*, 2007; Fratarcangeli, 2012] like NVIDIA PhysX, and the mass-spring (MS) methods [Galoppo *et al.*, 2007]. High-accuracy applications in science, engineering, and surgical applications typically require a more accurate and computationally complex physics-based technique for simulations, such as the finite element methods (FEM) [Capell *et al.*, 2005; Kim & Pollard, 2011]. The classic approach of "*layered skin deformation*" [Chadwick *et al.*, 1989] uses a skeleton that drives soft tissue motions and accommodates kinematic and dynamic deformations. However, to produce dynamic deformation, a simulation model is required. This section provides an introduction to various aspects of physically based deformation models, including time integration schemes and physics-based simulation techniques, namely mass-spring (MS, Section 3.3.2.1) models, finite element methods (FEM, Section 3.3.2.2) and position based dynamics (PBD, Section 3.3.2.3). Moreover, in this section, we evaluate the stability, performance and visual quality of each method. In this dissertation, we choose to employ a constraint-based technique, which is *position based dynamics* (PBD) [Müller *et al.*, 2007]. The use of position based dynamics adds believability to our skin deformation model and ensures real-time performance.

3.3.1 Time Integration

The simulation techniques to produce physically based deformations are often force based. Internal and external forces are accumulated from which accelerations are computed based on Newton's second law of motion (Eq. 3.3). A set of ordinary or partial differential equations defines the force fields applied to the different elements that represent the system. Then, a numerical integration scheme is employed to integrate the acceleration, in order to obtain the velocity and then the position of the element in a given time step.

$$\mathbf{F} = m\mathbf{a} \tag{3.3}$$

where $\mathbf{F} = \sum_{j=1} \mathbf{F}_j$ is the internal and external forces vector. $\mathbf{F}$ acts to push and pull an object around in space. The $\mathbf{a}$ in this equation expresses the acceleration vector ($\mathbf{a} = \frac{\mathbf{F}}{m}$), and m is the mass of the object, which is constant.

Given the positions $\mathbf{x}_i(t)$ and velocities $\mathbf{v}_i(t)$ of a point at time t, new positions $\mathbf{x}_i(t+h)$ and new velocities $\mathbf{v}_i(t+h)$ need to be determined for the next time step of the simulation $t+h$, according to the equation of motion (Eq. 3.3). For this, we first transform (Eq. 3.4) into a first order differential equation (Eq. 3.5):

$$\mathbf{F}_{external} = \mathbf{M}\frac{\partial^2 \mathbf{x}}{\partial t^2} + \mathbf{D}\frac{\partial \mathbf{x}}{\partial t} + \mathbf{F}_{internal} \tag{3.4}$$

$$\frac{\partial}{\partial t}\begin{pmatrix}\mathbf{x}\\ \mathbf{v}\end{pmatrix} = \begin{pmatrix}\mathbf{v}\\ \mathbf{F}_{external} - \mathbf{D}\mathbf{v} - \mathbf{F}_{internal}(\mathbf{x},\mathbf{v})\end{pmatrix} \tag{3.5}$$

where $\mathbf{D}$ is the damping matrix that represents viscous damping applied to each element in the system, and $\mathbf{M}$ the mass matrix.

Solving Eq. 3.5 can be done using a numerical integration method. Traditionally, numerical integration schemes can be divided into three general categories: explicit (forward) Euler integration (Section 3.3.1.1), implicit (backward) Euler integration (Section 3.3.1.2) and Verlet integration (Section 3.3.1.3). Therefore, these numerical integration methods deal with how to advance a simulation from one time step to the next step. In the following, we provide a short review of these methods, explaining their differences and drawbacks.

3.3.1.1 Forward Euler Integration

The simplest method of numerical integration is the explicit (forward) Euler scheme, which uses the positions, velocities, and external forces at the current time step to compute the corresponding values of the following step. The explicit Euler scheme is given by the following update rules:

$$\mathbf{v}_{t+h} = \mathbf{v}_t + h\mathbf{a}_t \tag{3.6}$$

$$\mathbf{x}_{t+h} = \mathbf{x}_t + h\mathbf{v}_t \tag{3.7}$$

This scheme guarantees first order precision of the computed positions and velocities. We are treating h as a time step and it is sufficiently small, where h controls stability and speed of the computation. $\mathbf{v}$ is the velocity vector, $\mathbf{x}$ is the position vector, and t is a specific time during the simulation. A solution obtained using explicit Euler scheme is quite unstable, suffers of overshooting problems and exhibits inferior accuracy. The smaller the time step, the higher the stability, and vice versa. The major drawback of this scheme is that the resultant deformation tends to vibrate and even explode, when time steps are too large.

3.3.1.2 Backward Euler Integration

In contrast to the explicit Euler scheme, the implicit (backward) Euler scheme assumes for the solution of the next simulation state that future variables are known. In other words, with the backward Euler scheme, we update the current position, not with the current velocity and acceleration but with the resulting velocity and acceleration. The problem with this approach is that the velocities and accelerations of the next time step are unknown. We need to solve for them, which can be done by treating implicit Euler integration as a linear equation and solve for it. However, the resulting set of equations can be large, which makes solving them each time step is computationally more expensive than the explicit method. An alternative solution is to use an explicit Euler scheme and then use the result to compute the implicit integration. This is known as a predictor-corrector method, since we predict a solution using the explicit Euler equation and then correct for errors using the implicit scheme.

The simplest implicit integration scheme is given by the following update rules:

$$\mathbf{v}_{t+h} = \mathbf{v}_t + h\frac{\mathbf{F}_{external} - d\mathbf{v}_{t+h}/dt}{m} \tag{3.8}$$

$$\mathbf{x}_{t+h} = \mathbf{x}_t + h\mathbf{v}_{t+h} \tag{3.9}$$

Although implicit methods are computationally expensive, they have the advantage of stability and are useful for large particle systems.

3.3.1.3 Verlet Integration

Verlet integration [Verlet, 1968] is a velocity-less integration scheme, which is frequently used in video games, mainly due to [Jakobsen, 2001] (Jakobsen is the first to employ Verlet integration in games). He developed the PhysX engine of a popular game called "*Hitman: Codename 47*", where he employed the Verlet scheme to simulate rag dolls, cloth and plants. The Verlet integration method works by using the current and previous position to create the velocity, instead of using the exact velocity (Eq. 3.10). In this formulation, the velocity term disappears and the position at the next time step $\mathbf{x}_{t+h}$ depends only on the current forces applied to the point, current position and position at the previous time step. However, because the velocity is now given only implicitly by using the previous position, the time step need to be kept constant between each call to the numerical integrator.

$$\mathbf{x}_{t+h} = 2\mathbf{x}_t - \mathbf{x}_{t-h} + \mathbf{a}_t h^2 \tag{3.10}$$

Verlet integration offers stability as well as other properties that are important in physical systems, such as time-reversibility, area preserving properties, and conservation of the energy of the system. In addition, Verlet integration is easy to implement. Simple pseudo-code is presented in Alg. 3. While standard Verlet integration is unconditionally stable, it does not incorporate velocity, which makes it difficult to use with velocity dependent forces.

Algorithm 3: Verlet integration pseudo-code

Input : *Double* timeStep, *Vector* previousPositions, *Vector* currentPositions, *Vector* currentAccelerations

Output : *Vector* deformedPositions

1: // VerletTimeStep
2: *Double* oldPosition
3: **while** currentPositions **do**
4: oldPositions = currentPositions
5: currentPositions= $1.99 \cdot currentPositions - 0.99 \cdot previousPositions + currentAccelerations \cdot timeStep \cdot timeStep$
6: // The two factors 1.99 and 0.99 are included
7: //, in order to introduce a small amount of drag in the system
8: previousPositions=oldPositions
9: **end while**

3.3.2 Physics-Based Simulation Techniques

Both the differential equations and numerical integration scheme constitute a physical model. A physical model is assessed according to its generality, accuracy and efficiency. First, we discuss the most used physical model in computer graphics community, which is the mass-spring model.

3.3.2.1 Mass-Spring Model

The mass-spring model (MS) is indubitably the simplest and most intuitive deformation model. In the mass-spring method (Fig. 3.8), a continuous body is approximated by a system of mass points connected by massless springs that are simulated using some variant of Hooke's law. This model allows to perform interactive deformations and to handle complex interactions with ease [Liu *et al.*, 2013b]. It is well-suited for generating visually plausible animations.

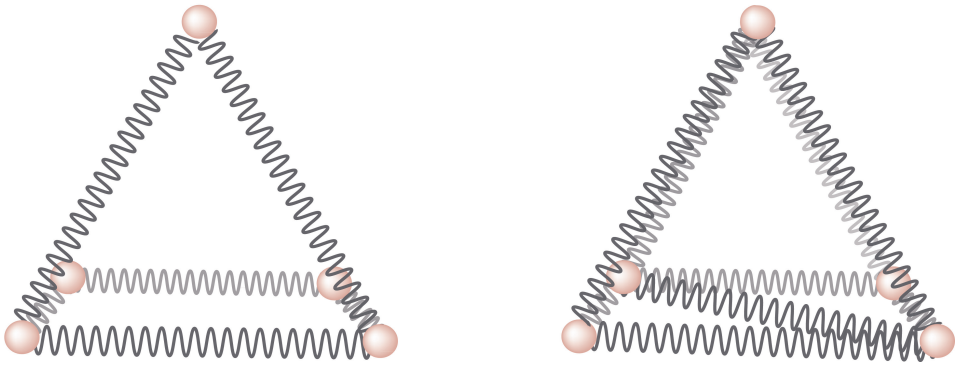


Figure 3.8: Two connected tetrahedra represented as a mass-spring model with different connected spring topologies. *Right.* Simple mass-spring model. *Left.* Volumetric mass-spring model with additional springs to preserve the volume.

The force applied on a mass point is given by rest length l^0, current length l, and spring constant k, according to Hooke's law $F = k(l^0 - l)$. The spring force acting on a point i with position $\mathbf{x}_i$ due to a connected spring to point j with position $\mathbf{x}_j$ is given below:

$$\mathbf{F}_i = \sum_j k_{i,j}(\|\mathbf{x}_j - \mathbf{x}_i\| - l^0_{i,j})\frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_j - \mathbf{x}_i\|} \tag{3.11}$$

Mass-spring methods vary according to the discretization mesh, the way the springs are set between the mass points, and the functions used to model the springs. Because mass-spring model is a discrete approach, it can only roughly approximate the true physics governing the deformation of an elastic object. Moreover, the integration of realistic tissue properties into this model is not straightforward, and often requires manual parameter tuning. In addition, the simulated mass spring elements have no volume; meaning that properties such as volume preservation are often simulated using additional springs (see Fig. 3.8) or heuristic functions. Mass-spring models are usually quite efficient and suitable for real-time animation, but they are unable to model accurately the exact properties of soft tissues [Chen *et al.*, 1998] and it is difficult to predict and generate the desired behavior using such models. For more information, we refer the interested reader to [Gibson & Mirtich, 1997].

3.3.2.2 Finite Element Method

The partial differential equations (PDEs) of continuum mechanics describe how soft body deforms under the presence of external forces. Continuum mechanics is used to model the kinematics and mechanical behavior of objects. It allows to approximate various physical quantities, such as energy and momentum. Therefore, continuum mechanics

defines the governing equations of an object based on its material properties. However, in continuum mechanics, the behavior of an object is modelled assuming continuous matter instead of a set of discrete mass points. There exist a variety of approaches to convert the continuum PDEs to a discretized system of ordinary differential equations (ODEs), which can then be time-stepped numerically. A commonly used discretization approach is the finite element method (FEM) [Shabana, 1989]. The Finite element method involves finding an approximate solution to a set of differential equations that define the dynamics of a continuum. The displacement-based finite element method can be used for soft body dynamics, such as soft-tissue simulation. This involves discretizing the continuous body into a finite number of elements, such as tetrahedra, hexahedra and polyhedra. Therefore, in the finite element method, the object is represented as a 3D volumetric mesh, by dividing the object into a large number of elements (see Fig. 3.9).

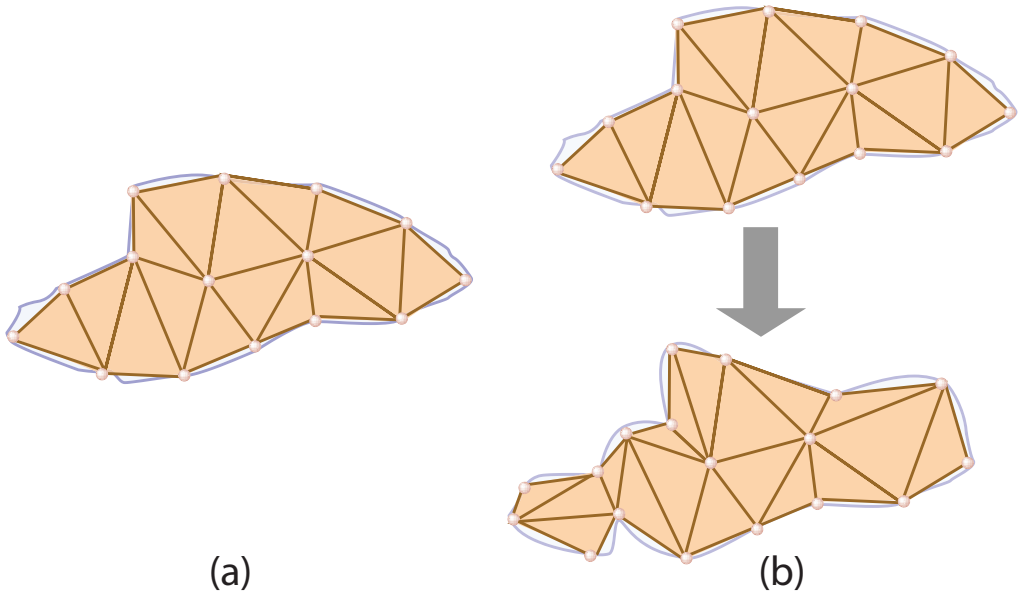


Figure 3.9: A 2D example of a finite element mesh approximating a soft body, in which it deforms over time. (a) The soft body in its rest position (initial state). (b) The soft body deformed, changing shape from rest position to the deformed position (deformed state).

This allows to represent the continuous displacement field using interpolation of nodal displacements. Thus, displacement is approximated via:

$$\mathbf{u} = \mathbf{N}\mathbf{u}' \tag{3.12}$$

where $\mathbf{N}$ is a matrix containing the interpolating shape functions and $\mathbf{u}'$ is the unknown vector of nodal displacements.

Therefore, the deformation is prescribed (controlled) by the displacements of the vertices of the mesh. Displacement at an arbitrary position is then defined via interpolation. Such an approximation will of course only be accurate if elements are sufficiently small, so that the interpolation is able to sufficiently describe the deformations within each element [Reddy, 93]. FEM is widely considered a reliable and accurate (assuming the mesh is fine enough) method to simulate soft bodies and deformable objects. Its main drawback is that it is computationally demanding. Unlike the mass-spring (MS) model, the FEM is based on continuum mechanics, making it more suitable for modelling continuous materials like soft tissue. However, it also requires knowledge of solid mechanics, where FEM of the deformable bodies are often considered nontrivial to implement. Despite offering realistic effects, FEM requires complex and intensive computations, and thus it is usually avoided in real-time applications. For more detailed information, we refer the interested reader to [Sifakis & Barbic, 2012].

3.3.2.3 Position Based Dynamics

In our deformation model, we simulate the character skin inside the position based dynamics (PBD) [Müller *et al.*, 2007; Bender *et al.*, 2014; Bender *et al.*, 2015] framework. We choose PBD for its unconditionally stable time integration and robustness. The most popular methods for simulating dynamics in computer graphics are force based. Force based methods compute the accelerations based on Newton's second law of motion. Then, a time integration method is used to update the velocities, and finally the positions of the object. Unlike force based methods, PBD omits the velocity and immediately works on the positions. The main advantage of this approach is its controllability. In PBD, the physical system is still modelled through equations governing external and internal forces that are applied to soft bodies, but these equations are formulated as a set of constraints. The integration scheme used in PBD is very similar to the Verlet method (Section 3.3.1.3). It is also closely related to the Nucleus solver ([Stam, 2009], Autodesk Maya), which also uses constraints to describe the objects to be simulated. However, the main difference is that Nucleus solves the constraints for velocities, not positions. Being based on an integration scheme that is very similar to the Verlet method, PBD does not suffer from instability and overshooting problems. The basic idea of PBD is to model the soft body as a particle system, simulate the dynamics according to external forces (if present), and then solve nonlinear constraints that express the geometric relations between particles. In the following, we briefly summarize the core idea of PBD, which is useful to understand our deformation model (in Chapter 4). The soft bodies

are modelled as a set of n particles whose motion is governed by a set of m nonlinear geometric constraints. Each particle $\mathbf{p}_i$ corresponds to a vertex in the input mesh, where geometric constraint $\mathbf{C}_j$ is a mathematical relationship between particles. Each particle has three attributes, a mass m_i, position $\mathbf{x}_i$ and velocity $\mathbf{v}_i$. Each constraint has a stiffness parameter k which defines the strength of the constraint in a range from zero to one. This gives a user more control over the elasticity of the body. The set of constraints is composed by nonlinear equality and inequality equations such that:

$$C_j(\mathbf{p}) \succ 0, \quad i = 1, \ldots, m \tag{3.13}$$

where the symbol $\succ$ stands for either $=$ or $\geq$. Inequality constraints are mainly used to resolve collisions, where m is the number of constraints. Given the set of n particles and m constraints, the simulation proceeds as described by Alg. 4, where Δt is the time step.

Algorithm 4: Position based dynamics algorithm

Input : *Vector* inputVertices, *Double* Δt, *Integer* solverIterations
Output : *Vector* deformedVertices

1: **for all** inputVertices i **do**
2: $\mathbf{v}_i = \mathbf{v}_i^0$
3: $\mathbf{x}_i = \mathbf{x}_i^0$
4: $w_i = 1/m_i$
5: **end for**
6: **loop**
7: **for all** inputVertices i **do**
8: $\mathbf{v}_i = \mathbf{v}_i + \Delta t w_i \mathbf{F}_{external}(x_i)$
9: $\mathbf{p}_i = \mathbf{x}_i + \Delta t \mathbf{v}_i$
10: **end for**
11: **for all** inputVertices i **do**
12: generateCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
13: **end for**
14: **for** solverIterations **do**
15: projectConstraints($\mathbf{C}_1, \ldots, \mathbf{C}_m, \mathbf{p}_i, \ldots, \mathbf{p}_n$)
16: **end for**
17: **for all** deformedVertices i **do**
18: $\mathbf{v}_i = (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
19: $\mathbf{x}_i = \mathbf{p}_i$
20: **end for**
21: velocityUpdate($\mathbf{v}_i, \ldots, \mathbf{v}_n$)
22: **end loop**

The positions and velocities of the particles are initialized in (1-5) before the simulation

loop starts. Lines (6-10) perform simple explicit forward Euler integration step on the velocities and the positions, where gravity is represented by external force $\mathbf{F}_{external}$. Collision constraints are generated in lines (11-13), where collision constraints can be handled easily and penetrations can be completely resolved inside the PBD solver loop. The geometric constraints are iteratively solved in lines (14-16). The idea is to repeatedly solve each constraint sequentially one after the other, in a Gauss-Seidel type fashion. The process is repeated *solverIterations* times. The corrected positions $\mathbf{p}_i$ are finally used to update the positions and the velocities in (17-21).

Constraint Projection: the set of constraints must always be satisfied, or at least, the error should be as small as possible. During the simulation, if the particle configuration $\mathbf{p}_i$ does not satisfy the set of the constraints, then the solver *projects* the particle positions into a valid state by finding a displacement $\Delta\mathbf{p}$ that satisfies the constraints. For example, the distance constraint:

$$C(\mathbf{p}_1, \mathbf{p}_2) = |(\mathbf{p}_1 - \mathbf{p}_2)| - d^2 = 0 \tag{3.14}$$

is used to keep particles $\mathbf{p}_1$ and $\mathbf{p}_2$ at distance d. The constraints are generally nonlinear, like in the just mentioned example, and they are solved sequentially through Gauss-Seidel iterations. Each equation is linearized individually to find the correction $\Delta\mathbf{p}$:

$$C_i(\mathbf{p} + \Delta\mathbf{p}) \approx C_i(\mathbf{p}) + \nabla_{\mathbf{p}} C_i(\mathbf{p}) \cdot \Delta\mathbf{p} = 0 \tag{3.15}$$

where $\nabla_{\mathbf{p}} C_i(\mathbf{p})$ is the vector containing the derivatives of the equation C_i w.r.t. the n components of $\mathbf{p}$.

The correction $\Delta\mathbf{p}$ is imposed to be in the direction of $\nabla_{\mathbf{p}} C(\mathbf{p})$:

$$\Delta\mathbf{p} = \lambda_i \nabla_{\mathbf{p}} C_i(\mathbf{p}) \tag{3.16}$$

This condition implicitly conserves the linear and angular momenta for the single constraints while, at the same time, allowing us to solve the under-determined system of constraints. Combining Eqs. 3.15 and 3.16 yields:

$$\lambda_i = -\frac{C_i(\mathbf{p})}{\left|\nabla_{\mathbf{p}} C_i(\mathbf{p})\right|^2} \tag{3.17}$$

Chapter 4 provides more detail about this method and how we employ it in our skin deformation model.

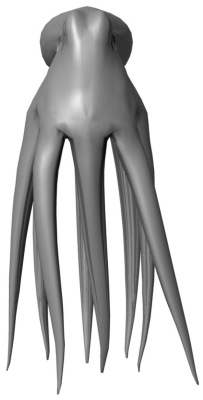
3.4 Conclusion

In this chapter, we have given a thorough description of the theoretical and mathematical background of real-time skinning techniques. In addition, we provided an introduction to various aspects of physically based soft body simulations, including physics-based simulation techniques and time integration schemes.

Interactive skeleton-based skinning techniques, namely linear blend skinning and dual quaternion skinning are purely kinematic approaches and suffer from visual artefacts. In contrast, physically based simulations provide believable deformations and able to capture the elasticity of the soft tissues. However, they are either hard to implement, suffer from instability or require intensive computations. Thus, they are usually avoided in real-time applications.

This chapter also introduced the concept of layered skin deformation, and we discussed how the layered skin deformation framework could extend the existing deformation techniques to include volume preservation, elasticity and jiggle. In the next chapter, we investigate the possibility of combining classic interactive skinning techniques with a physics based method in a layered skin deformation framework. We would like this combination to be able to avoid the artefacts of classic interactive skinning techniques, and to be capable of capturing believable skin deformations at interactive rates.

CHAPTER 4



Position based Skinning for Soft Articulated Characters

Believable skin deformation for soft articulated characters is essential to enrich the visual experience of the animation and for creating appealing character animation in movie productions, computer games, and virtual reality applications. The production of life-like deformations includes capturing a range of desirable effects, like secondary motions, volume preservation, and contact deformations. In character animation, the skin deformation of an articulated character is determined primarily by its underlying skeleton, which leads to purely kinematic deformations. Our goal is to produce compelling skin deformations and capture the desired effects of skin in an efficient and robust simulation. We would like the skin of a character to follow the motion of a skeleton, and at the same time exhibit realistic elastic behaviors within the computational limits imposed by interactivity. Therefore, the deformation model must trade off between the following requirements: it must (1) be fast enough to achieve an interactive rate (i.e., > 30 fps), (2) produce believable animation to minimize manual post-processing time, and (3) be controllable and stable.

In this chapter, we introduce a novel two-layered approach addressing the problem

of creating believable mesh-based skin deformation (Section 4.1). The deformation model discussed in this chapter mimics the behavior of the skin and achieves effects like volume preservation and jiggling. We also allow the artist to tune the amount of jiggling and specify the areas of the skin affected by it (Section 4.7). To address the above-mentioned requirements, we present a CPU-based implementation (in Section 4.6). Moreover, we demonstrate the visual quality and the performance of our deformation model with a variety of skeleton-driven soft body characters (Section 4.8), where we also show that our approach enables stable simulation for appealing deformations of a wide range of animated characters (including humans and animals).

4.1 Position Based Skinning

Our new deformation model, called position based skinning (PBS), is based on the concepts that have been discussed in (Section 3.1). Our method is conceptually very straightforward. We employ a robust combination of two popular techniques well known in the computer graphics community: linear blend skinning (Section 3.2.1) and position based dynamics (Section 3.3.2.3). The idea is to create a two-layered deformation scheme, the result of which approximates the behavior of the skin. For each animation frame, the deformation of the character is decoupled in two steps. First, we apply classic linear blend skinning (LBS, Section 4.2) to the character. Then, in a second step, we simulate the skin as a soft body, using a deformation approach that is based on position based dynamics (PBD, Section 4.3). We define a system of geometric constraints, which are used to model the volume conservation and skin jiggling in real-time. These geometric constraints are applied in a parallel fashion and solved on a multi-core CPU (in Section 4.6). The resulting skinned animations are unaffected by the well known artefacts proper of LBS, namely the "*candy-wrapper*" effect and loss of volume. Being based on PBD, this second step is efficient, controllable and unconditionally stable, even when large time steps are employed for advancing the character's dynamics.

4.1.1 Method Overview

The inputs of our model are a fine surface mesh representing the character in its rest pose and an animated skeleton embedded within the mesh. From the surface mesh, we generate a volumetric (tetrahedral) mesh of the same shape using the method of [Boissonnat & Oudot, 2005], which preserves the original outer surface geometry (Fig. 4.1).

(a) (b) (c)

Figure 4.1: Inputs to our method; (a) the embedded skeleton within the tetrahedral mesh, (b) the fine surface corresponding to the coarse tetrahedral mesh, and (c) the fine surface.

We use the tetrahedral elements for defining the geometric constraints used to compute the elastic motion of the character skin while moving, including volume preservation and jiggling. In the initialization phase, the vertices of the original skin mesh are mapped to the tetrahedral elements by using barycentric coordinates, and the geometric constraints are defined according to the rest pose of the character. Then, at every animation frame, the skeleton moves and the mesh vertices are first deformed through a standard linear blend skinning (LBS) process. Thereafter, the vertex positions are adjusted by solving the constraints. The geometric constraints are solved using a position based dynamics scheme. We employ a graph coloring algorithm for parallelizing the computation of the constraints. Finally, the resulting vertex positions are used for updating the fine mesh, which is employed for rendering. The whole mechanism is summarized in Fig. 4.2.

Tetrahedral Mesh Generation: The skin is represented as a volumetric mesh consisting of 3D tetrahedra called elements. We use the first-order tetrahedron element in all of our experiments; this element is a polyhedron with four faces (a tetrahedron) and linear shape functions (Fig. 4.3). A tetrahedral mesh is a tetrahedralization of the interior structure of a triangular surface. In order to generate a tetrahedral mesh that preserves the original outer surface geometry, we use a method based on Delaunay refinement [Boissonnat & Oudot, 2005]. A Delaunay tetrahedralization (DT) is a tetrahedralization where for each tetrahedron there exists a circumsphere such that no point lies inside it. Optimal properties of such tetrahedralization include that it maximizes the minimum angle and minimizes the maximum circumradii, which therefore helps prevent highly irregular tetrahedron being generated.

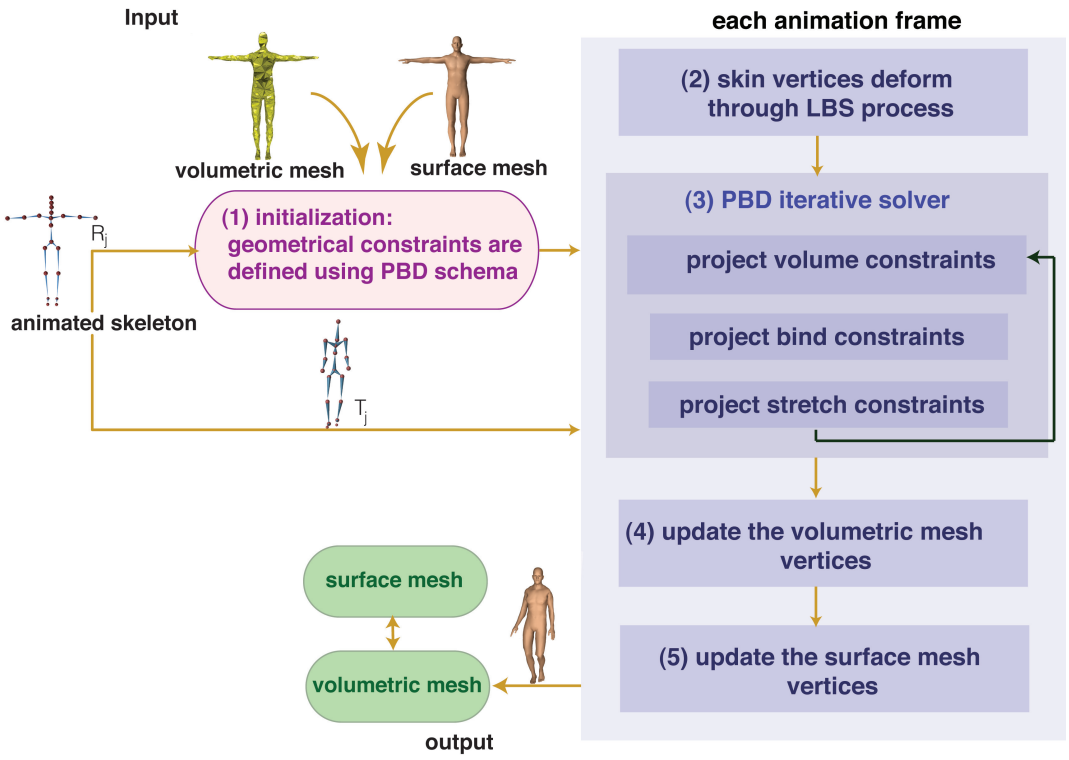


Figure 4.2: In the initialization phase, the fine surface mesh is converted to a tetrahedral mesh, which is used to define soft geometric constraints. During the animation of the skeleton, the volumetric mesh is first deformed using linear blend skinning, after which the constraints are solved using a parallel position based dynamics scheme.

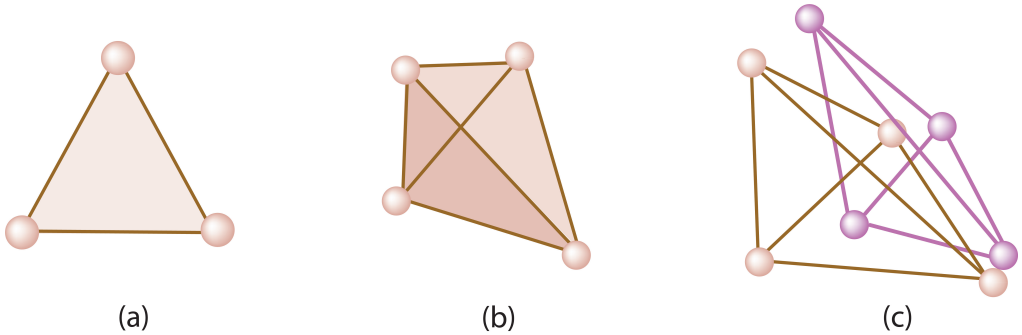


Figure 4.3: (a) A surface composed of one triangle strip. (b) First-order tetrahedron element, and (c) the tetrahedron element before (shown in orange) and after deformation (shown in purple).

Mesh Coupling The surface mesh is deformed according to the simulation performed on the tetrahedral mesh (Fig. 4.4). We apply linear surface interpolation, similar to [Müller & Gross, 2004]. For each surface vertex, the closest tetrahedron is found and the barycentric coordinates of the vertex with respect to the linked tetrahedron are computed and stored. When the tetrahedral mesh deforms, the position of each vertex in the surface is interpolated using the linked tetrahedron and the stored barycentric coordinates.

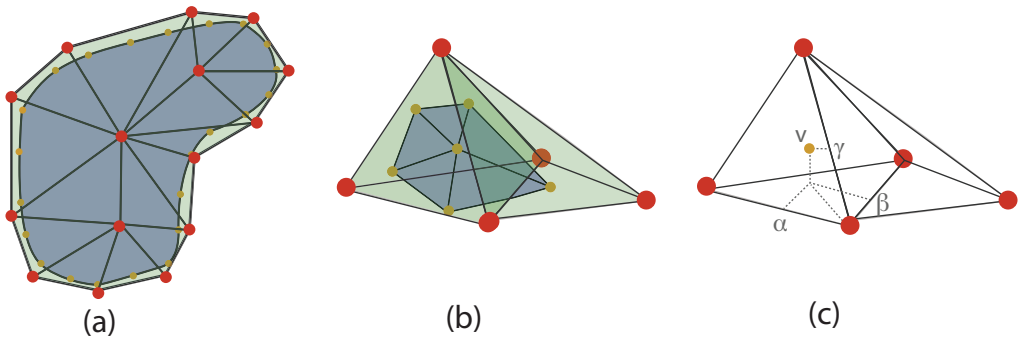


Figure 4.4: 2D example of the coupling between surface and tetrahedral mesh. (a) The surface (shown in blue) deformed according to the corresponding deformed tetrahedral mesh (shown in green). (b) A close-up view of two tetrahedrons with the embedded surface composed by six vertices, where for each vertex $\mathbf{v}$ the containing tetrahedron to $\mathbf{v}$ is found using barycentric coordinates, as in (c).

4.2 Linear Blend Skinning

In the LBS deformation layer, we apply linear blend skinning on the tetrahedral mesh. As explained in the previous chapter, the LBS method attaches each vertex $\mathbf{v}_i$ in the input mesh to one or more skeletal bones; each attachment affecting the vertex with a different strength, or *weight*. The final deformed vertex position $\mathbf{v}'_i$ is a weighted average of its initial position transformed by each of the attached bones, according to (Eq. 3.1).

Binding Weights: In order to use the LBS algorithm in the first step of our method, we need to determine the bone weights $\mathbf{w}_{i,j}$ for each vertex v_i. As we explained in (Chapter 3), smooth linear blend skinning does not guarantee rigid transformation. The simplest way to guarantee rigid transformation is by consider all the weights are either one or zero. Then, the result is exactly rigid blending. To provide a good foundation for the weights, we add more smoothness to the rigid blending and we try to avoid the manually adjusting the weights. We attach each vertex $\mathbf{v}_i$ to at most 3 bones. To compute the

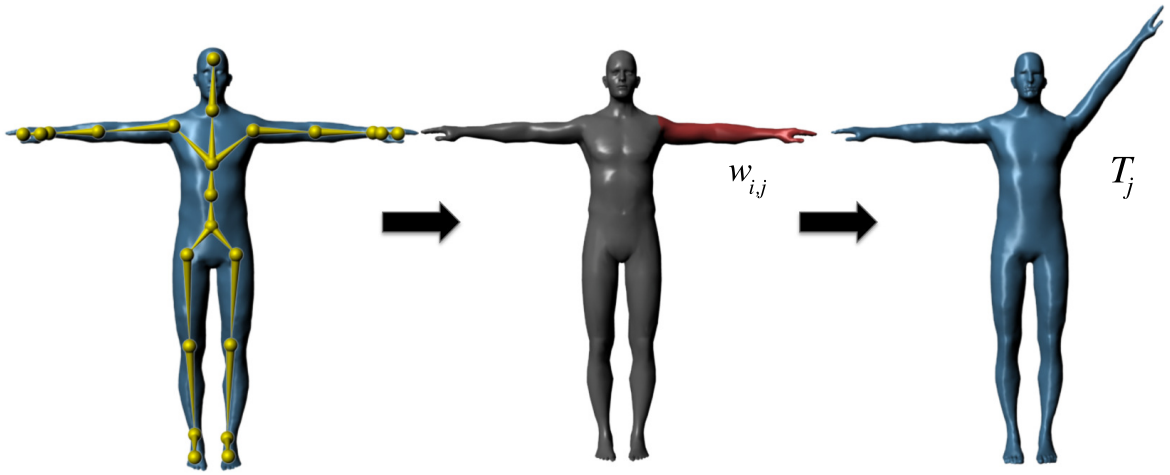


Figure 4.5: A character skinned by linear blend skinning. *Left.* The skeleton is embedded within the mesh in its initial position. *Middle.* The skinning weights are assigned for the character's left arm, in which these weights are determined using (Eq. 4.1). *Right.* The character deformation is computed using (Eq. 3.1).

weights, we use the following formula, which is based on the distance from the vertex to the bone:

$$w_{i,j} = \begin{cases} 1.0 & \text{if } \frac{1}{6} \leq \frac{(\mathbf{v}_i - \mathbf{s}_j)}{|\mathbf{d}_j - \mathbf{s}_j|} \cdot \frac{(\mathbf{d}_j - \mathbf{s}_j)}{|\mathbf{d}_j - \mathbf{s}_j|} \leq \frac{5}{6} \\ 1.0/n_{joints} & \text{otherwise} \end{cases} \tag{4.1}$$

where $\mathbf{s}_j$ and $\mathbf{d}_j$ are the parent and child joint positions (end points) of the bone j nearest to $\mathbf{v}_i$, and n_{joints} is the number of bones attached to the end points bone nearest to $\mathbf{v}_i$. In this way, vertices far away from the end points of a bone are rigidly influenced by only that bone, while vertices near the end points are equally influenced by the bones in that area. The resulting deformation is smooth but suffers from joint collapse and loss of volume, issues which are fixed by the second step of our method, as described in the next section.

4.3 Position-based Dynamics

The core idea of PBD has been explained (in Section 3.3.2.3). The skin is modelled as a set of n particles whose motion is governed by a set of m nonlinear constraints. The system of constraints is solved using Gauss-Seidel iterations by directly updating the particle positions. PBD avoids the use of internal forces, and the positions are updated such that the angular and the linear momenta are implicitly conserved. In this way, the whole process is not affected by the typical instabilities of interactive physically based

methods. In the following sections (4.3.1.1- 4.3.1.3), we define the geometric constraints used in this deformation layer and how they are solved.

4.3.1 Geometric Constraints

We use geometric constraints for modelling the skin, the inner volumetric structure, and the binding of the skin with the skeleton. These constraints are iteratively satisfied, leading to primary and secondary motions of the external skin in real-time.

4.3.1.1 Stretch Constraint

We define one *stretch* constraint for the particles $(\mathbf{p}_1, \mathbf{p}_2)$ at the end points of each edge of the tetrahedral mesh, including the edges of the internal tetrahedrons (Fig. 4.6). The *stretch* constraint is formed according to the following equation:

$$C(\mathbf{p}_1, \mathbf{p}_2) = |\mathbf{p}_1 - \mathbf{p}_2| - d = 0 \tag{4.2}$$

is used to keep particles $\mathbf{p}_1$ and $\mathbf{p}_2$ at distance d. where d is the rest length of the edge. Given the configuration $(\mathbf{p}_1, \mathbf{p}_2)$ of two particles connected by a stretch constraint, the corrections to the positions (Eq. 3.16) in order to satisfy the constraint are:

$$\begin{aligned} \Delta\mathbf{p}_1 &= -\tfrac{1}{2} k_s (|\mathbf{p}_1 - \mathbf{p}_2| - d) \tfrac{\mathbf{p}_1 - \mathbf{p}_2}{|\mathbf{p}_1 - \mathbf{p}_2|} \\ \Delta\mathbf{p}_2 &= +\tfrac{1}{2} k_s (|\mathbf{p}_1 - \mathbf{p}_2| - d) \tfrac{\mathbf{p}_1 - \mathbf{p}_2}{|\mathbf{p}_1 - \mathbf{p}_2|} \end{aligned} \tag{4.3}$$

where $k_s \in [0,1]$ is a stiffness scalar parameter which slows the convergence of the constraint and provides a "springy" behavior to the corresponding edge.

4.3.1.2 Tetrahedral Volume Constraint

We define one *volume* constraint for the particles $(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4)$ at the corners of each tetrahedron of the mesh. The volume constraint maintains the rest volume of four particles forming a tetrahedron (Fig. 4.6), enforcing the conservation of the total volume of the character's body:

$$C(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4) = \frac{1}{6}(\mathbf{p}_{2,1} \times \mathbf{p}_{3,1}) \cdot \mathbf{p}_{4,1} - V_0 \tag{4.4}$$

where $\mathbf{p}_{i,j}$ is the short notation for $\mathbf{p}_i - \mathbf{p}_j$ and V_0 is the rest volume of the tetrahedron.

The gradient with respect to each particle is:

$$\nabla_{\mathbf{p}_2} C(\mathbf{p}_{2,3}) = \frac{1}{6}(\mathbf{p}_{2,1} \times \mathbf{p}_{3,1}) \tag{4.5}$$

$$\nabla_{\mathbf{p}_3} C(\mathbf{p}_{3,4}) = \frac{1}{6}(\mathbf{p}_{3,1} \times \mathbf{p}_{4,1}) \tag{4.6}$$

$$\nabla_{\mathbf{p}_4} C(\mathbf{p}_{4,2}) = \frac{1}{6}(\mathbf{p}_{4,1} \times \mathbf{p}_{2,1}) \tag{4.7}$$

$$\begin{aligned} \nabla_{\mathbf{p}_1} C(\mathbf{p}_1) &= -(\nabla_{\mathbf{p}_2} C(\mathbf{p}_{2,3}) + \nabla_{\mathbf{p}_3} C(\mathbf{p}_{3,4}) + \\ &\quad + \nabla_{\mathbf{p}_4} C(\mathbf{p}_{4,2})) \\ &= -\frac{1}{6}(\mathbf{p}_{2,1} \times \mathbf{p}_{3,1} + \mathbf{p}_{3,1} \times \mathbf{p}_{4,1} + \\ &\quad + \mathbf{p}_{4,1} \times \mathbf{p}_{2,1}) \end{aligned} \tag{4.8}$$

The correction of each particle belonging to the tetrahedron is:

$$\Delta\mathbf{p}_1 = -\frac{1}{6} s \cdot k_v (\mathbf{p}_{2,1} \times \mathbf{p}_{3,1} + \mathbf{p}_{3,1} \times \mathbf{p}_{4,1} + \mathbf{p}_{4,1} \times \mathbf{p}_{2,1}) \tag{4.9}$$

$$\Delta\mathbf{p}_2 = \frac{1}{6} s \cdot k_v (\mathbf{p}_{2,1} \times \mathbf{p}_{3,1}) \tag{4.10}$$

$$\Delta\mathbf{p}_3 = \frac{1}{6} s \cdot k_v (\mathbf{p}_{3,1} \times \mathbf{p}_{4,1}) \tag{4.11}$$

$$\Delta\mathbf{p}_4 = \frac{1}{6} s \cdot k_v (\mathbf{p}_{4,1} \times \mathbf{p}_{2,1}) \tag{4.12}$$

where k_v is the stiffness parameter and s is the scaling factor:

$$s = \frac{\frac{1}{6}(\mathbf{p}_{2,1} \times \mathbf{p}_{3,1}) \cdot \mathbf{p}_{4,1} - V_0}{\Sigma_{i=1}^{4} |\nabla_{\mathbf{p}_i} C(\mathbf{p}_i)|^2} \tag{4.13}$$

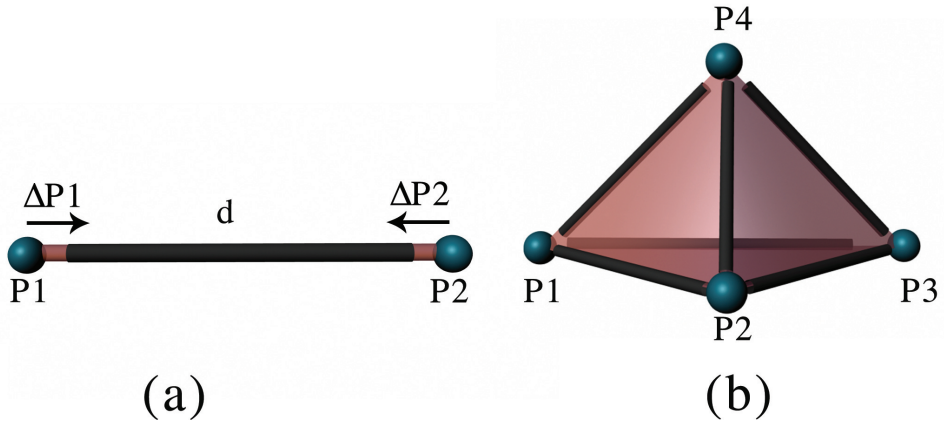


Figure 4.6: A volume constraint is defined for each tetrahedron, together with six stretch constraints (one for each edge).

4.3.1.3 Bind Constraint

We define a *bind* constraint between each particle of the tetrahedral mesh and its nearest bone. Basically, a bind constraint is a stretch constraint between a particle and its projection on the nearest skeleton bone. When the skeleton moves and the joints rotate, the projection point for each particle is updated accordingly and the bind constraint pushes or pulls the particle to maintain the rest distance. This mechanism is depicted in Fig. 4.7.

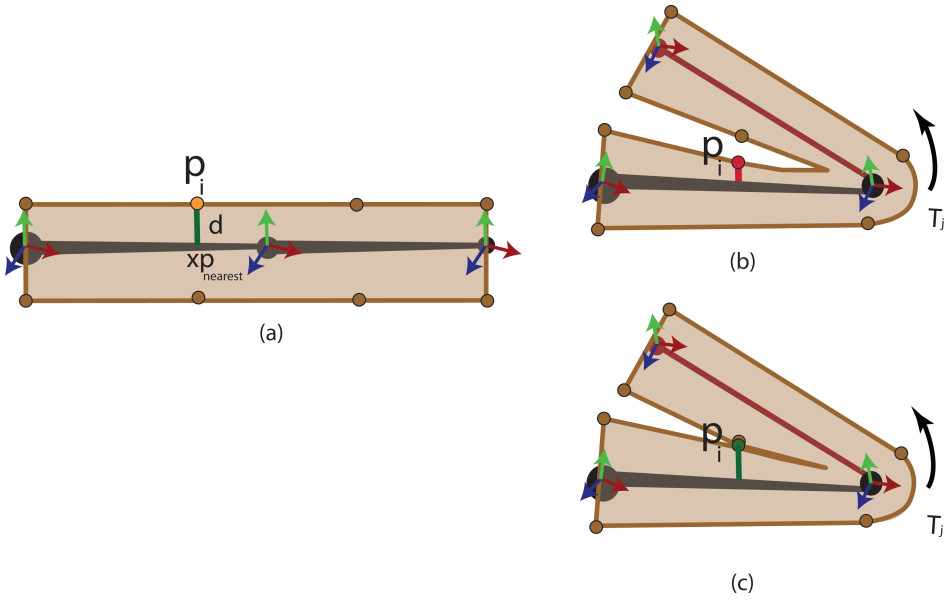


Figure 4.7: (a) The particle $\mathbf{p}_i$ in the rest pose, d is the rest distance from the nearest bone. (b) The particle $\mathbf{p}_i$ is displaced from the first step of our approach, by using LBS. (c) The *bind* constraint maintains the rest distance from the bone.

4.4 Final Algorithm

In this section, we summarize our two-layered deformation model in Alg. 5. First, we define a system of geometric constraints, which are used for modelling the skin, the inner volumetric structure, and the binding of the skin with the skeleton. In each animation frame, the skin first deforms with LBS, and then the vertices positions are adjusted using the PBD scheme.

Algorithm 5: Position based skinning algorithm

Input : *Mesh* surfaceMesh, *Matrices4x4* currentbonesTransformations, *Matrices4x4* restbonesTransformations, *Matrices4x4* bindingweights, *Double Deltat*, *Integer* solverIterations

Output : *Mesh* surfaceMesh

// Coupling between the surface and tetrahedral mesh
generatetetrahedralMesh(*Mesh* surfaceMesh)
findBarycentricCoordinateForVertex(surfaceMesh,tetrahedralMesh)
// For each tetrahedron, we define a volume constraint
for each t_i in tetrahedralMesh **do**
    defineGeometricConstraints(TetrahedralVolume)
end for
// For each vertex, we define a bind constraint
for each v_i in tetrahedralMesh **do**
    defineGeometricConstraints(Bind)
end for
// For each edge, we define a stretch constraint
for each e_i in tetrahedralMesh **do**
    defineGeometricConstraints(Stretch)
end for
for each animation frame **do**
    // Loop through every vertex and compute blended positions through LBS
    for each v_i in tetrahedralMesh **do**
        $deformedVertices_i = inputVertices_i + bindingweights_{i,j} \times$
        $currentbonesTransformations \times restbonesTransformations_j^{-1}$
    end for
    // Simulating the skin as a soft body using PBD
    for each v_i in tetrahedralMesh **do**
        $\mathbf{v}_i = \mathbf{v}_i^0$
        $\mathbf{x}_i = \mathbf{x}_i^0$
        $w_i = 1/m_i$
    end for
    loop
        for each v_i in tetrahedralMesh **do**
            // We apply gravity as external force
            $\mathbf{v}_i = \mathbf{v}_i + \Delta t w_i \mathbf{F}_{ext}(x_i)$
            $\mathbf{p}_i = \mathbf{x}_i + \Delta t \mathbf{v}_i$
        end for
        for solverIterations **do**
            projectTetrahedralVolumeConstraints($\mathbf{C}_1, \ldots, \mathbf{C}_m, \mathbf{p}_i, \ldots, \mathbf{p}_n$)
            projectBindConstraints($\mathbf{C}_1, \ldots, \mathbf{C}_m, \mathbf{p}_i, \ldots, \mathbf{p}_n$)
            projectStretchConstraints($\mathbf{C}_1, \ldots, \mathbf{C}_m, \mathbf{p}_i, \ldots, \mathbf{p}_n$)
        end for
        for each v_i in tetrahedralMesh **do**
            $\mathbf{v}_i = (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
            $\mathbf{x}_i = \mathbf{p}_i$
        end for
        velocityUpdate($\mathbf{v}_i, \ldots, \mathbf{v}_n$)
    end loop
    UpdatesurfaceMeshVertices()
end for

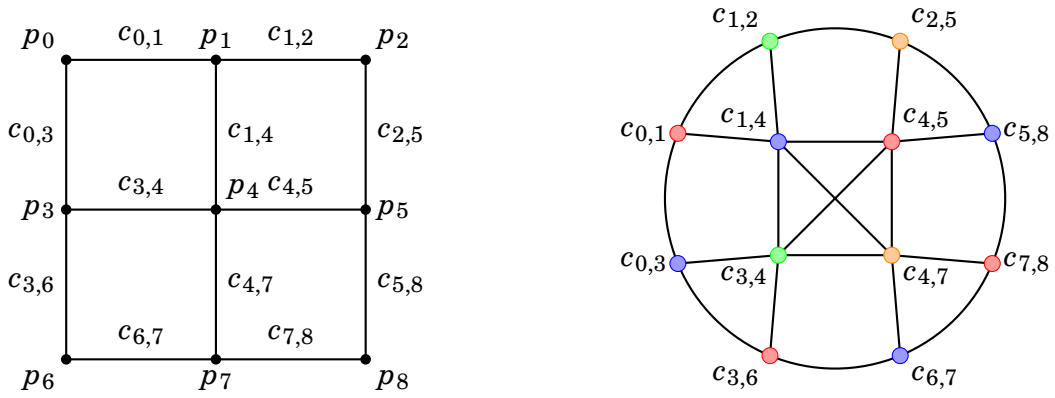
4.5 Gauss-Seidel Solver

The goal of the solver step (34-38) is to correct the predicted positions $\mathbf{p}_i$ of the particles such that they satisfy all constraints. We are using a nonlinear Gauss-Seidel iterative method, which solves each constraint equation separately. Each constraint yields a single scalar equation $C(\mathbf{p}) \succ 0$ for all the particle positions associated with it. Therefore, each equation is linearized individually in the neighborhood of C around the current configuration $\mathbf{p}$ to find the correction $\Delta\mathbf{p}$. The solver linearizes the constraint functions, which corrects and updates the particle position in a single step. Because the positions are immediately updated after a constraint is processed, these updates will influence the linearization of the next constraint, as the linearization depends on the actual positions. In the next section, we discuss how these geometric constraints are applied in a parallel fashion and solved on the multi-core CPU.

4.6 Parallel Position Based Skinning

The set of constraints in the original Position-Based approach are solved in a Gauss-Seidel fashion; the constraints are solved in a serial way, in succession, directly correcting the position of the particles. This process is iterated several times for each animation frame; for each iteration, the difference between the current and the optimal solution of the system decreases. This approach is efficient when the number of constraints is relatively small and thus the number of iterations needed for reaching a good approximation of the global solution is low. However, when a high number of constraints is involved, both the computational cost of a single iteration and the number of iterations for each frame increases. In this case, the serial solution of the set of constraints quickly becomes unsuitable for interactive animation.

As noted in [Fratarcangeli & Pellacini, 2015; Bender *et al.*, 2014], this process can be parallelized by using a graph coloring algorithm. A dual graph is defined, where each node represents a constraint and two nodes are connected if they share at least one particle. Then the graph is colored with a distance-1 algorithm, such that two adjacent nodes are not assigned with the same color. The constraints belonging to the same color do not share any particle and can be solved in parallel by using a number of dedicated CPU threads. When all the constraints belonging to a color are solved, the constraints assigned to the following color are solved, until all the colors are considered. The process of coloring the graph happens just once in the initialization phase, and does not affect



(a) A set of particles connected by stretch constraints. Here $c_{i,j}$ is the short notation for $C(\mathbf{p}_i, \mathbf{p}_j)$.

(b) The corresponding dual graph. Two constraints are connected if they share at least one particle. The graph is colored with a distance-1 algorithm and the set of constraints belonging to the same color are solved in parallel, reducing the number of serial steps from 12 to 4.

Figure 4.8: A graph coloring algorithm is applied to a simple particle system to parallelize the computation of the constraints; (a) a simple particle system of 9 particles and 12 constraints, (b) a dual graph is defined, where each node represents a constraint and two nodes are connected if they share at least one particle.

the performance during the animation. In this way, the asymptotic complexity for each iteration improves from $\mathcal{O}(m)$ where m is the number of constraints, to $\mathcal{O}(q)$ where q is the total number of colors. We used Sequential Vertex Coloring [Coleman & More, 1983] which usually leads to a number of colors equal to the size of the biggest clique in the graph, or slightly bigger. Fig. 4.8 depicts this mechanism for a simple example.

4.7 Soft Control

We allow the artist to specify the amount of jiggling in a manually specified area of the mesh surface. In a pre-processing step, the artist selects a surface area using an external 3D modeling tool (we used Autodesk Maya). This set of surface vertices is then mapped to the nearest vertices in the tetrahedral mesh (Fig. 4.9). All the tetrahedral vertices included between the selected vertices and the nearest bone are also selected. The average distance $\overline{d}$ from the nearest bone is then computed. If the distance of a vertex from the nearest bone is smaller than $\overline{d}$, the motion of the vertex is governed by both LBS and PBD. If the distance is greater than $\overline{d}$, only PBD is applied in that area. The artist can change interactively the stiffness of all the constraints influenced by these latter vertices, reducing or increasing the amount of jiggling until the desired effect is

reached.

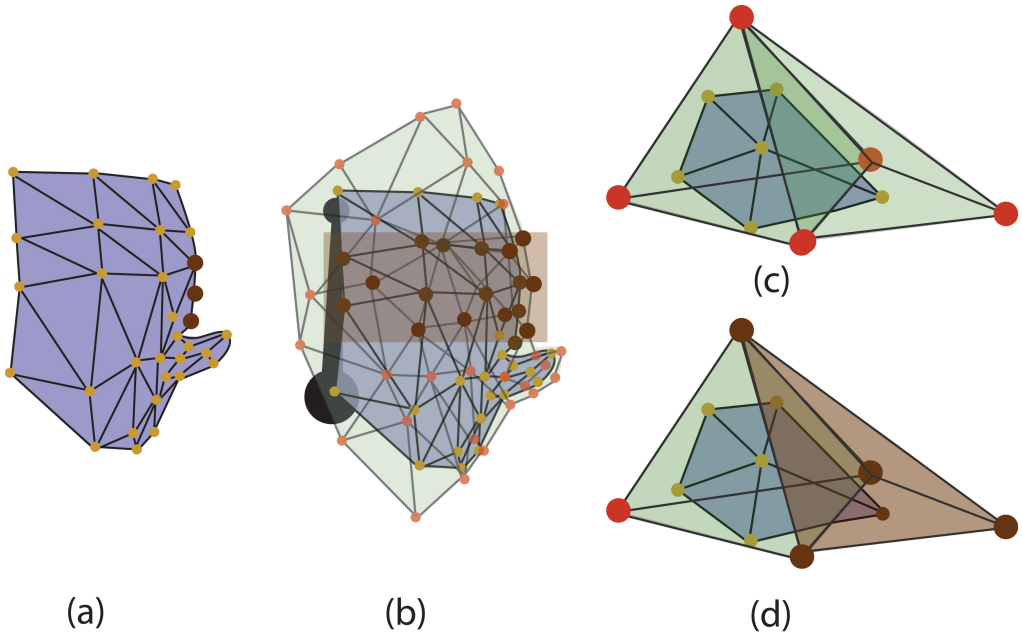


Figure 4.9: Soft selection mechanism. (a) Surface mesh, including the selected vertices. (b) Surface mesh embedded in the tetrahedral mesh. The tetrahedron vertices placed between the selected surface vertices and the nearest bone are selected automatically. (c) A close-up view of two tetrahedrons with the embedded surface composed of six vertices. (d) The surface vertex is selected by the artist, and the nearest vertices on the tetrahedron are selected automatically. All of these vertices are shown in brown, and the brown rectangle highlights these vertices.

4.8 Experiments and Results

We tested our method on a variety of articulated characters of different shapes, polygonal resolutions and topologies. The tetrahedral meshes were generated using [Boissonnat & Oudot, 2005], the *walking* and the *kick* motion capture data was available from the Carnegie Mellon University Motion Capture DataBase [Gross & Shi, 2001], and the remaining motions were hand-made by animator using Autodesk Maya.

4.8.1 Visual quality

Figs. 4.10, 4.11 and 4.12 show the TORSO, the HORSE and the BIGBUNNY models animated with a walking cycle. All of these models exhibit large jiggling effects in the area of the belly. Fig. 4.13 shows an octopus model with articulated tentacles and jiggling

effects in the head zone. Fig. 4.14 provides a visual comparison between the deformations achieved with our method, plain linear blend skinning (LBS), and dual quaternion skinning (DQS). Our system is not affected either by the "*candy-wrapper*" effect or by undesired skin bulging. Fig. 4.15 shows the HUMAN model performing a highly dynamic motion sequence involving large joint rotations, which demonstrates the robustness of our system. Please refer to the video for the animated results.

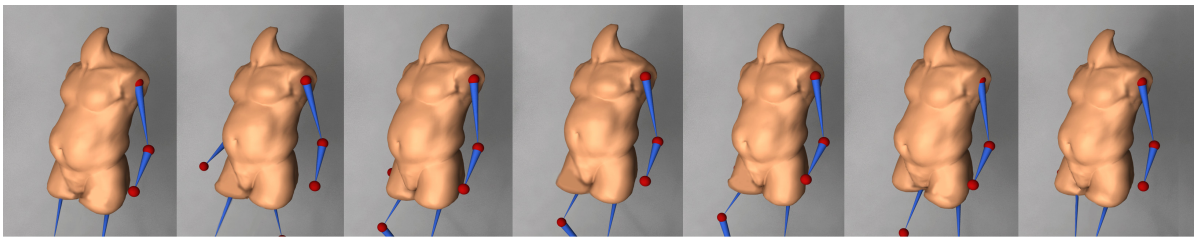


Figure 4.10: Believable skin deformation including secondary motions and volume preservation, TORSO (11K constraints, 149 fps).

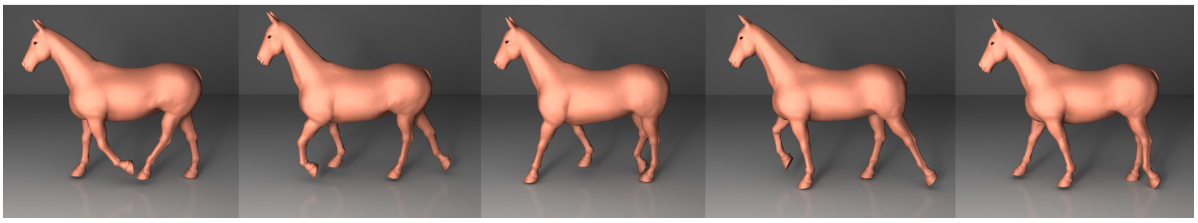


Figure 4.11: Real-time animations of a complex articulated character, HORSE (17K constraints, 130 fps).

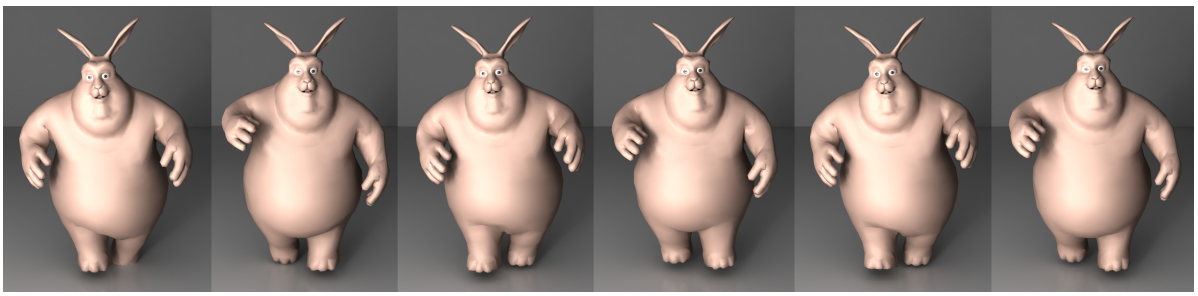


Figure 4.12: Realistic jiggling motion in the belly area for the BIGBUNNY character at interactive rates (34k constraints, 71 fps).

To assess the accuracy of our method, we compute the relative error w.r.t. to the volume

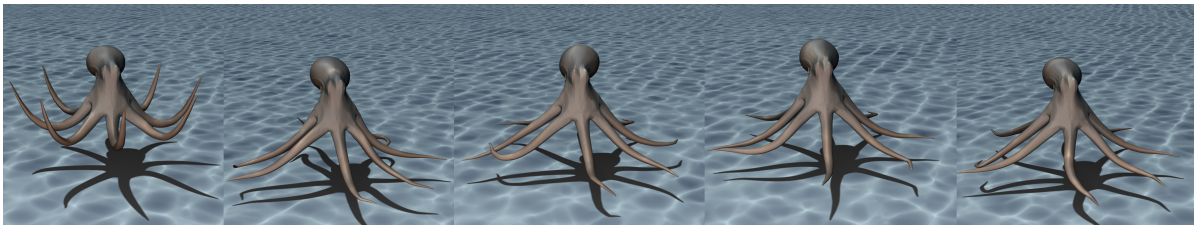


Figure 4.13: Our method automatically produces believable skin deformations of the soft tissues for an octopus moving at interactive rates (14k constraints, 145.6 fps).

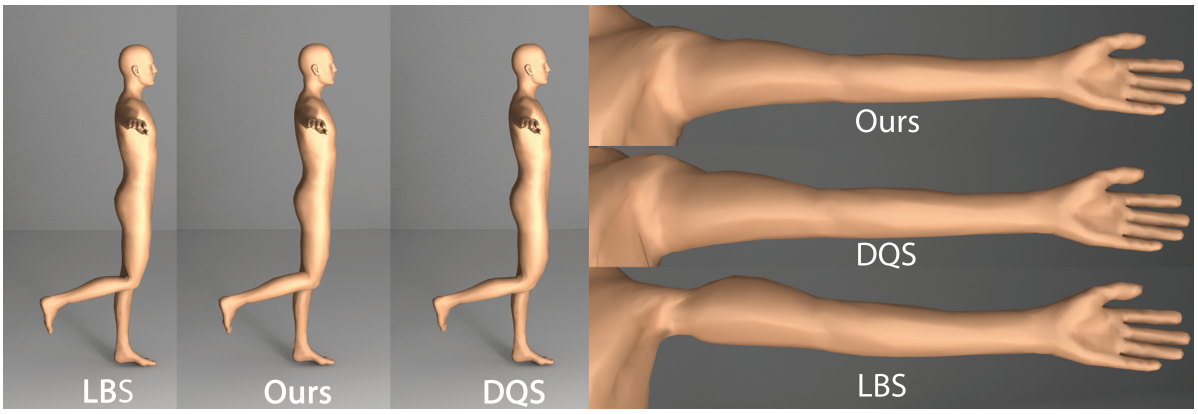


Figure 4.14: Our method does not suffer from the "*candy-wrapper*" artefacts of linear blend skinning (LBS) and the bulging artefacts of dual quaternion skinning (DQS).

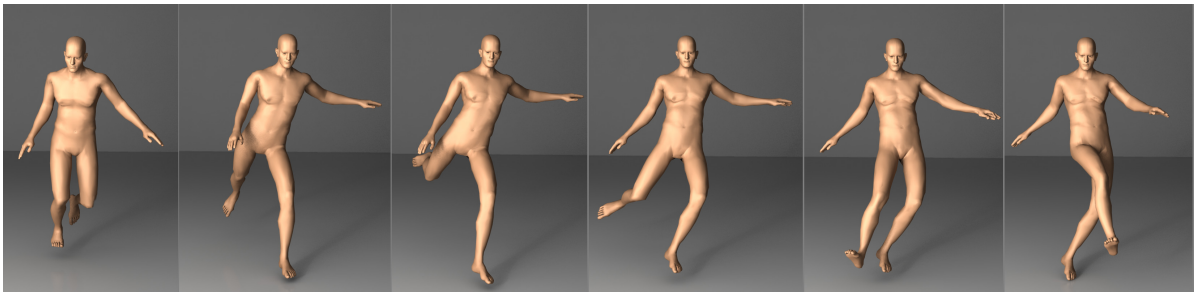


Figure 4.15: Highly dynamic sequence, HUMAN character (12k constraints, 146 fps).

of the character's mesh: $e = V/V_0$, where V is the volume of the deformed mesh at the current frame, and V_0 is the volume of the mesh in the initial rest position. As shown from the quantities reported in Table 4.2, our method successfully preserves the volume of the character during the animation ($e \leq 0.5\%$). Fig. 4.16 shows an example of the deformation of the arm with and without volume conservation.

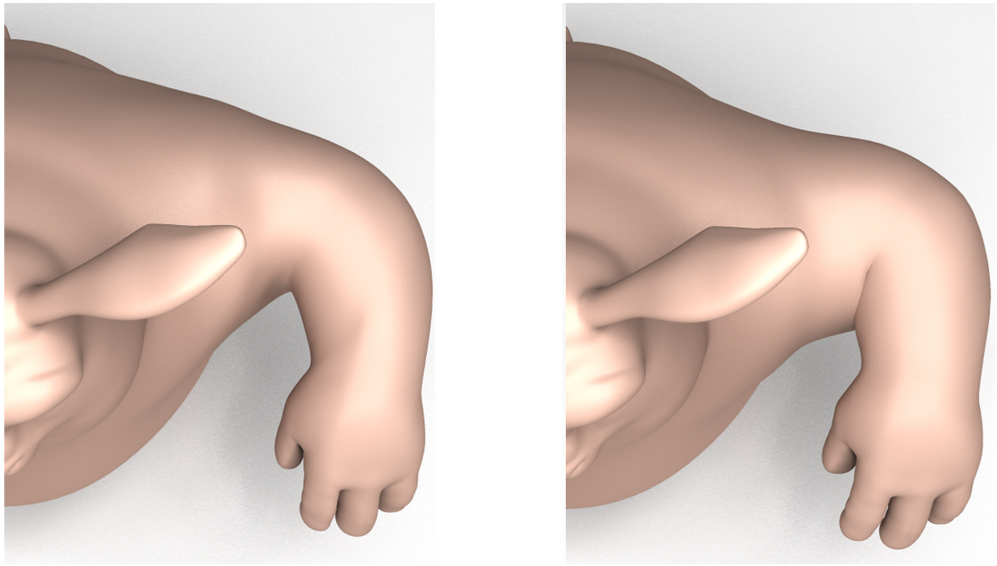


Figure 4.16: Comparison with and without volume preservation. The volume is preserved by solving the PBD constraints. *Left.* Step 1: LBS is applied. Note the loss of volume at the elbow joint. *Right.* Step 2: Positions of the vertices are adjusted, by solving the PBD constraints (stretch, volume and bind), preserving the volume and avoiding the "*candy-wrapper*" effect.

4.8.2 Performance

We implemented our method in C++ and timings for various animation sequences were measured on a mass-market laptop equipped with an Intel i5 2.50 GHz processor and 4GB RAM.

4.8.2.1 First Experiment

In our first experiment, the tetrahedral meshes have roughly 2K vertices and 10K tetrahedral elements [Abu Rumman & Fratarcangeli, 2014]. We used a 10 ms time step and 24 iterations per frame. Note that in this experiment, we used the sequential Gauss-Seidel scheme to solve the geometric constraints. We tested our method on tetrahedral meshes of only biped characters, as we can see in Figs. 4.17, 4.18 and 4.19. Please refer to the video for the preliminary results and for more sophisticated results of this experiment, please refer to the video. The mean computation times are reported in Table 4.1.

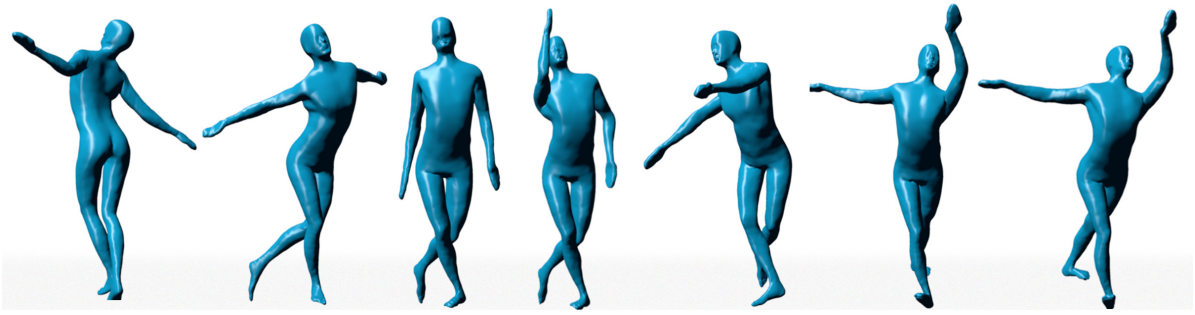


Figure 4.17: Real-time character animation driven by a kinematic skeleton, including secondary motions and volume preservation (71.1 fps).

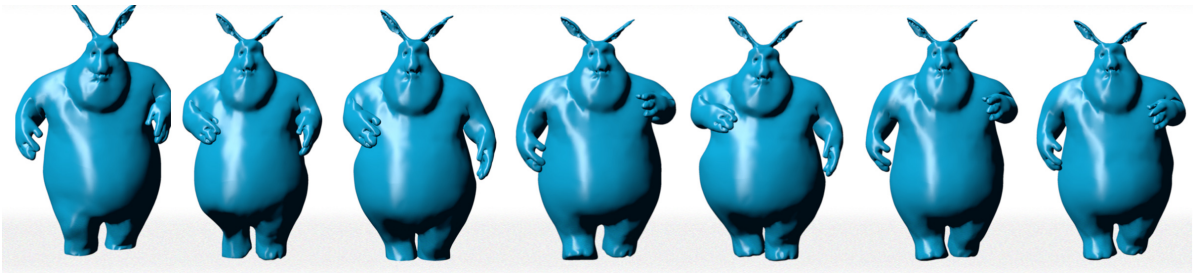


Figure 4.18: Real-time character animation driven by a kinematic skeleton, including secondary motions and volume preservation (65.8 fps).

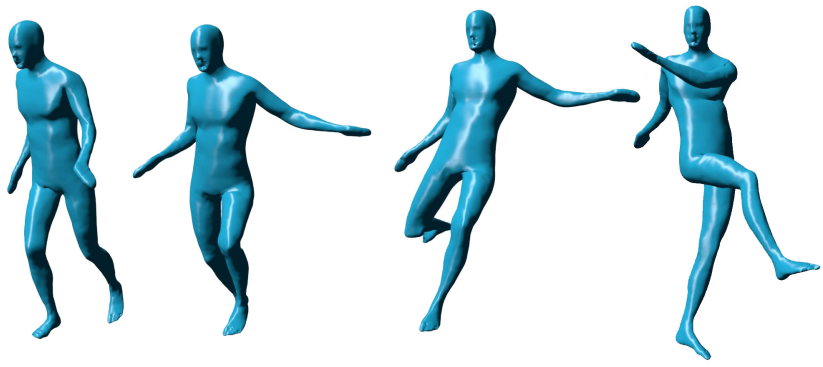


Figure 4.19: Different poses of a character kicking a ball (90.6 fps).

4.8.2.2 Second Experiment

To advance the dynamics in our second experiment, we used a 10 ms time step and 12 iterations per frame. In this experiment, the geometric constraints are solved in parallel, where the mean computation times are reported in Table 4.2. Even by using a fairly high number of constraints (e.g., the BIGBUNNYHQ test case, ~34K constraints), our system

is able to compute the animation of the tetrahedral mesh and map the deformation on the render mesh at interactive rates.

Model	# vertices	S	T	B	CT_{volume} [ms]	$CT_{stretch}$ [ms]	CT_{bind} [ms]	CT_{total} [ms]	# iterations	$k_{stretch}$	fps
Man	1654	6342	4998	1654	3.8	4.501	2.75	11.05	24	0.9	90.5
ManHQ	2645	13223	10516	2645	4.37	6.25	3.444	14.064	24	0.9	71.1
bunny	2108	9732	7827	2108	4.87	5.05	3.1	13.020	24	0.6	76.8
bunnyHQ	3190	17367	14022	3190	5.85	6.9	2.446	15.196	24	0.6	65.8
Lady	1829	9387	6032	1829	5.12	5.32	2.14	12.579	24	0.7	79.5

Table 4.1: Skinning performance. S: number of stretch constraints, T: number of volume constraints, B: number of bind constraints, fps: avg. frame rate, CT: avg. skinning computation time for running a 1 second simulation, where ($CT_{total} = CT_{volume} + CT_{stretch} + CT_{bind}$)

Model	# vertices	# tetra	# bones	S	T	B	CT_{volume} [ms]	$CT_{stretch}$ [ms]	CT_{bind} [ms]	CT_{total} [ms]	# iterations	volume loss %	$k_{stretch}$	fps
HUMAN	9528	4998	25	6342	4998	1654	2.1	3.2	1.507	6.807	12	0.09	0.9	146.9
HUMANHQ	9528	10516	25	13223	10516	2645	3.43	5.113	1.627	10.17	12	0.12	0.9	98.3
BIGBUNNY	8111	7827	29	9732	7827	2108	2.71	3.74	1.53	7.98	12	0.27	0.6	125.3
BIGBUNNYHQ	8111	14022	29	17367	14022	3190	5.41	6.56	1.95	13.92	12	0.39	0.6	71.8
OCTOPUS	4000	4339	63	7617	4339	1667	2.08	3.28	1.507	6.867	12	0.45	0.6	145.6
HORSE	3833	6126	43	9437	6126	1826	2.65	3.52	1.52	7.69	12	0.17	0.8	130
TORSO	4345	4280	25	6076	4280	1074	2.07	3.17	1.469	6.709	12	0.15	0.6	149.1

Table 4.2: Skinning performance. #vertices: number of initial vertices in the render mesh, #tetra: number of elements in the tetrahedral mesh, S: number of stretch constraints, T: number of volume constraints, B: number of bind constraints, fps: avg. frame rate, CT: avg. skinning computation time during 1 second simulation, where ($CT_{total} = CT_{volume} + CT_{stretch} + CT_{bind}$).

4.9 Comparison and Limitations

Currently, the skinning weights in our method are computed according to a simple heuristic formula (Eq. 4.1). Those weights are simple to construct, but may not be optimal for any given input mesh. Alternatively, the skinning weights can be computed using the techniques in [Dionne & de Lasa, 2013], which uses voxels to approximate the geodesic distance for computing bone influence weights. In future work, we would like to explore the possibility of applying their algorithm in the first step of our deformation model to bolster the second step and produce higher quality smooth bindings.

In contrast to other methods (e.g., [Kim & Pollard, 2011; Bender *et al.*, 2013]), our system does not model the inner structure of the human skin. However, using a combination of widely known and relatively simple techniques, linear blend skinning and position based dynamics, we achieved believable animations with a time performance suitable for interactive applications.

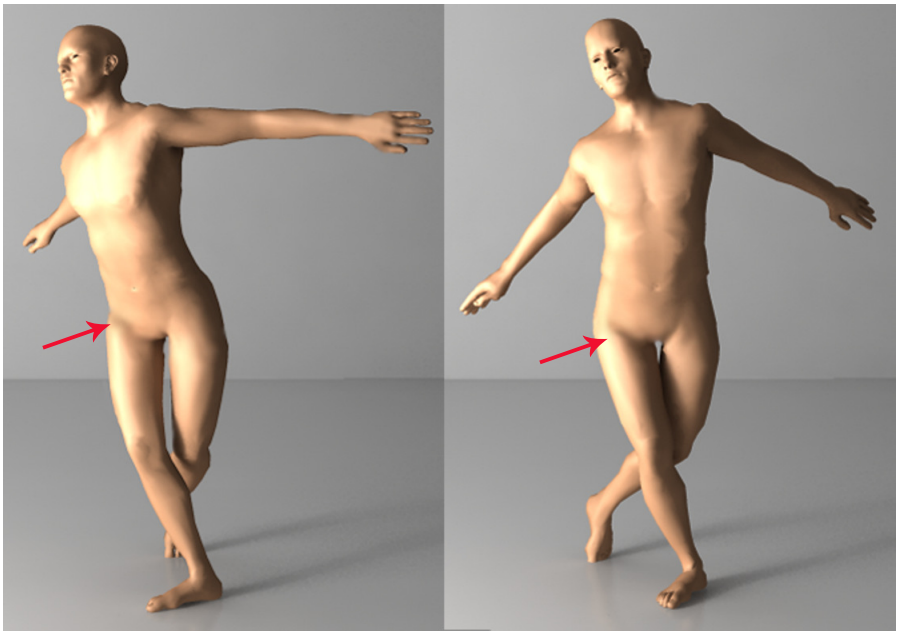


Figure 4.20: Self-collisions are not explicitly handled. This may lead to geometry overlapping in the contact regions (note the dark regions near the articulations).

Our implementation does not detect and resolve self-collisions, which may lead to geometry overlapping (Fig. 4.20), and therefore our method cannot guarantee self-intersection-free deformations. In the next chapter we extend our framework to support temporary constraints for implementing self-collisions.

4.10 Conclusion

We have presented a simple and fast skinning algorithm for skeleton-driven deformations of articulated characters [Abu Rumman & Fratarcangeli, 2015]. During the animation, the deformation model preserves the volume and allows for passive jiggling behavior. Our system initializes the blend weights and the soft constraints automatically, so it does not require a considerable set-up effort. Artists can define specific areas of the body and decide on the amount of jiggling affecting them by tuning a single scalar stiffness parameter. In the first step, the character is deformed by applying linear blend skinning, and in the second step the position of the vertices is corrected using a position based dynamics solver. The first step in our skinning method, the LBS deformer, improves the convergence speed of the PBD solver, while maintaining the elastic nature of the character body. We employ a graph coloring algorithm for parallelizing the computation of the geometrical constraints. This leads to fast performances even in the case of a fairly high number of tetrahedra. We map the tetrahedral mesh to the input skin geometry for achieving high quality renderings.

Being based on PBD, this second step is efficient, controllable and unconditionally stable, even when large time steps are employed for advancing the character's dynamics. The elastic behavior of the soft body deformer is influenced by the number of iterations employed in the parallel Gauss-Seidel solver. We employed 12 iterations during our tests, but in general the artist has to heuristically choose a value which depends on the topology and the polygonal resolution of the input mesh. In the next chapter, we address the main limitation of this method, which is the absence of self-collision handling.

CHAPTER 5

COLLISION HANDLING FOR SOFT ARTICULATED CHARACTERS

The position based skinning method presented in the previous chapter produces believable skin deformations at interactive rates. It also produces interesting effects, such as secondary motions and volume preservation. The most obvious deficiency of position based skinning is the lack of collision and contact handling. The deformations that occur in real creatures are highly dependent on contact between skin parts, especially the tissue around the joints. In this chapter, we address the problem of collision handling on articulated deformable characters. Commonly, collision handling in computer animation is composed of two steps: collision detection and collision response. These two steps are performed independently, with the result of the collision detection algorithm being used as an input to the collision response process.

Unlike highly deformable bodies such as clothes, articulated characters have limited deformation. This deformation is restricted to a function of an underlying skeletal structure, where self-collisions typically occur in very localized regions of meshes. We exploit the skeletal nature of the deformation to obtain real-time self-collision handling for models deformed by position based skinning. In this chapter, we first study the biomechanical

basis of the articulated characters (in Section 5.1). Then, we exploit the mechanical nature of the skeleton to achieve real-time collision detection (Section 5.3). In our collision detection method, we employ spatial hashing (Section 5.3.1) to detect collisions and self collisions on skeletally deformable meshes. Thereafter, to handle collisions and capture contact deformations, we formulate a constraint-based contact response method within the position based dynamics framework (Section 5.4). The resulting algorithm is simple to implement and fast enough for real-time applications (Section 5.5). We demonstrate the efficiency of our method on various animation examples (in Section 5.6).

5.1 Biomechanical Basis of Articulated Characters

Articulated characters (including humans and animals) form one of the most important subject in computer animation. Therefore, the skin deformation of such characters must "*bring the graphical characters to life*" by making their skin deform in a believable manner [Chen & Zeltzer, 1992]. One important way to achieve this is by enriching the deformations, which requires studying the biomechanics of the skeletal system and exploiting its properties into the skinning process.

The skeleton of a 3D character consists of a set of bones connected by articulated joints, arranged in a tree data structure. The joints are connected up in a hierarchical fashion to the selected root joint, where the root joint has no parent. Generally, all skeletons are kept as open trees without any closed loops; this restriction does not prevent kinematic loops. Most joints allow a certain amount of movement, where each joint has one or more degrees of freedom (DOF). Individual joints usually have one to six DOF, but all together, a detailed character may have more than a hundred DOF in the entire skeleton. Specifying values for these DOF poses the skeleton, and changing these values over time results in movement, and is the essence of the animation process. In this section, we discuss the mechanics of the skeletal system, and we provide a structural classification of joints.

5.1.1 Classification of Joints

This kind of classification is often available in games, for the purpose of ragdolling. Most movements in real life, such as walking, running and jumping, involve simultaneous or sequential movement in two or more joints (Fig. 5.1). In such multi-joint movements, the number of degrees of freedom in that part of the skeletal system responsible for the

movement is the sum of the number of degrees of freedom of the individual joints involved. Consequently, there is an almost infinite number of combinations of joint movements that could be employed in all multi-joint movements. Furthermore, temporary or permanent

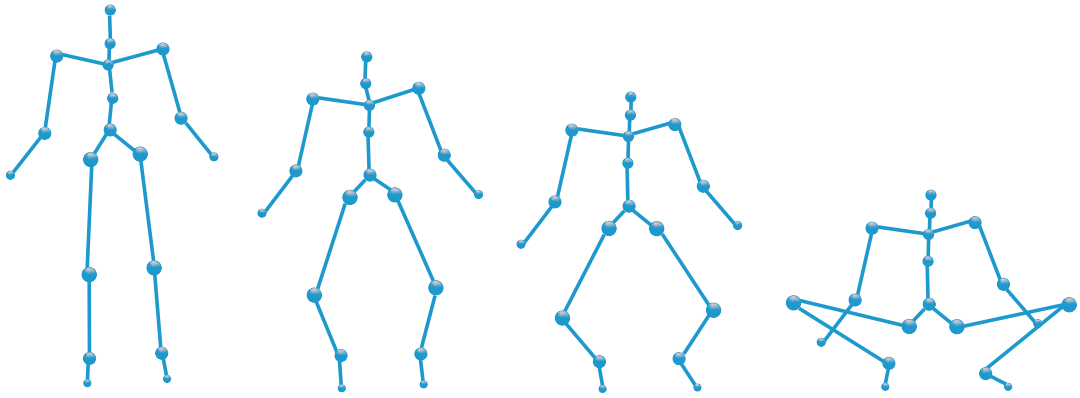


Figure 5.1: More than ten joints in the human skeleton are simultaneously rotated to perform a jumping movement.

impairment in one joint can usually be compensated for by changing in the movement of other joints. In realistic characters, all joints are rotational. Both 1-DOF and 3-DOF rotational joints are common, and 2-DOF joints are used occasionally as well. Therefore joints can be classified into three main types (Fig. 5.2):

Uniaxial (1-DOF): in uniaxial joints, movement takes place mainly about a single axis. This type of joint allows movement in only one plane, such as the elbow, knee and ankle joint. There are two types of uniaxial joint: hinge joints and pivot joints. Hinge joints allow rotation around one axis perpendicular to the length of the bones involved. In a hinge joint, usually the extremity of one bone is concave while the other is convex. In contrast, pivot joints present angular motion around an axis parallel to the length of the bones involved, allowing one bone to turn around the other.

Biaxial (2-DOF): in biaxial joints, movement mainly takes place about two axes at right angles to each other. There are three types of biaxial joint: condyloid, ellipsoid and saddle. A condyloid joint is a joint where an ovoid head of one bone moves in an elliptical cavity of another, permitting all movements except axial rotation; this joint type is found at the wrist. An ellipsoid joint has an egg-shaped head that allows opposition movement only to a small degree. Its movement, just as the diameter and curvature of an ellipse, varies in directions at right angles to each other (hence the name). A saddle joint moves like an ellipsoidal one. The difference is in the joint shape. Each articulating extremity

in a saddle joint has a double curvature, and when put together, the convex curve of one fits on the concave curve of the other; this type is found at the carpometacarpal joint of the thumb.

Multiaxial (3-DOF): Some joints, such as the shoulder and hip, can rotate around three axes. By combining rotations about the three reference axes, these joints can rotate about any axis in between the three. The human multiaxial joints are also called spheroidal joints or ball-and-socket joints.

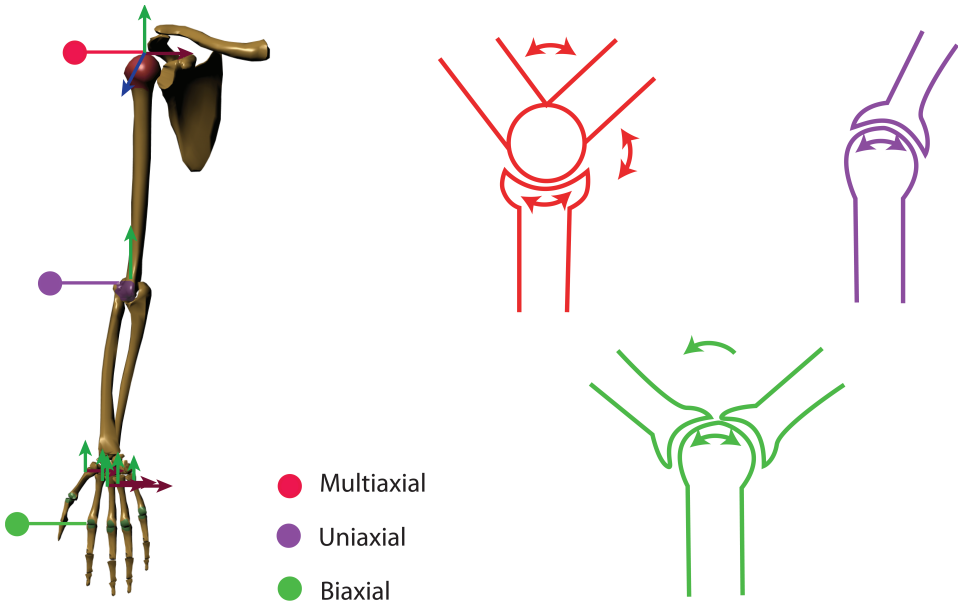


Figure 5.2: Shows an example of the classification of joints in the human skeleton, where the shoulder is a multiaxial joint (shown in red), the elbow is a uniaxial joint (shown in purple) and the carpometacarpal joint of the thumb is a biaxial joint (shown in green).

5.1.2 Axes of Rotation and Types of Movement

Motion can be defined as a continuous change in position of a character. The axis around which movement takes place and the plane through which movement occurs define specific motions or resultant positions. There are three cardinal anatomical planes that pass through the articulated body: *coronal* (frontal), *sagittal* (anteroposterior), and *transverse* (horizontal). The coronal plane divides the body front (anterior) and back (posterior) portions. The sagittal plane divides the the body into the left and right side. The transverse plane divides the body into the upper (superior) and lower (inferior) portions. Each plane is perpendicular to the other (see Fig. 5.3). In the following, we

discuss the axes and types of movements that occur in these planes.

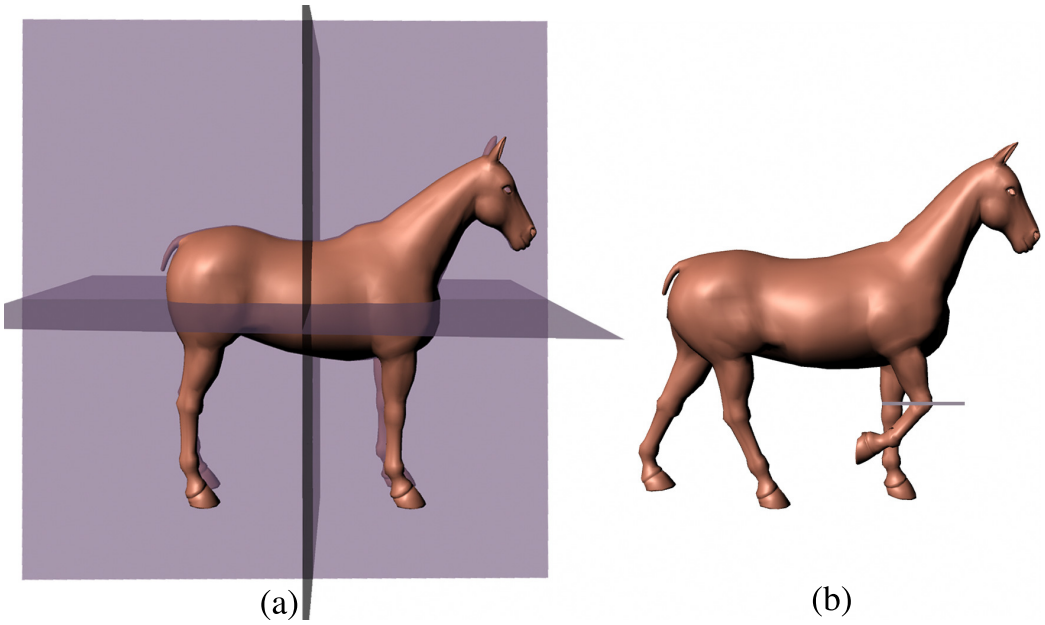


Figure 5.3: (a) The three cardinal planes all intersect at a single point known as the body's center of mass or center of gravity. (b) Flexion motion at horse's knee joint, which occurs about the coronal axis.

The coronal axis (x-axis) lies in the coronal plane and extends from one side of the body to the other. The motions of flexion and extension occur about this axis and through the sagittal plane. Extension is the motion opposite of flexion. Flexion refers to a decrease in joint angle in the sagittal plane, while extension is motion increasing joint angle. The sagittal axis (z-axis) lies in the sagittal plane and extends horizontally from anterior to posterior. Movements of abduction and adduction of the extremities, as well as lateral flexion of the spine, occur around this axis and through the coronal plane. Abduction is movement away from the body, and adduction is movement toward the body; the reference here is to the midsagittal plane of the body. Lateral flexion is a rotational movement and is used to denote lateral movements of the head, neck, and trunk in the coronal plane. The longitudinal axis (y-axis) is vertical, extending in a head-to-toe direction. Movements of medial (internal) and lateral (external) rotation in the extremities, as well as axial rotation in the spine, occur around it and through the transverse plane. It is often assumed that the motion of a joint is an angular movement (rotation about a static axis). However, the axis of rotation does not stay the same, but instead continually changes throughout the motion. Thus, it is necessary to have a set of axes of rotation in order to perform the joint motion properly.

5.2 Collision Handling for Soft Articulated Characters

In this section, we present an efficient method for detecting collisions and self-collisions on articulated models deformed by position based skinning (PBS). Additionally, to address the self-intersections problem of PBS, we propose a real-time collision response. We generate temporary constraints inside the position based skinning framework, such that these self-intersections are handled in a localized manner.

5.2.1 Method Overview

Fig. 5.4 illustrates an overview of our method. The inputs are a fine surface mesh and an animated skeleton. From the surface mesh, we generate a tetrahedral mesh using [Si, 2015], which preserves the original outer surface geometry and consists of tetrahedrons of roughly the same size.

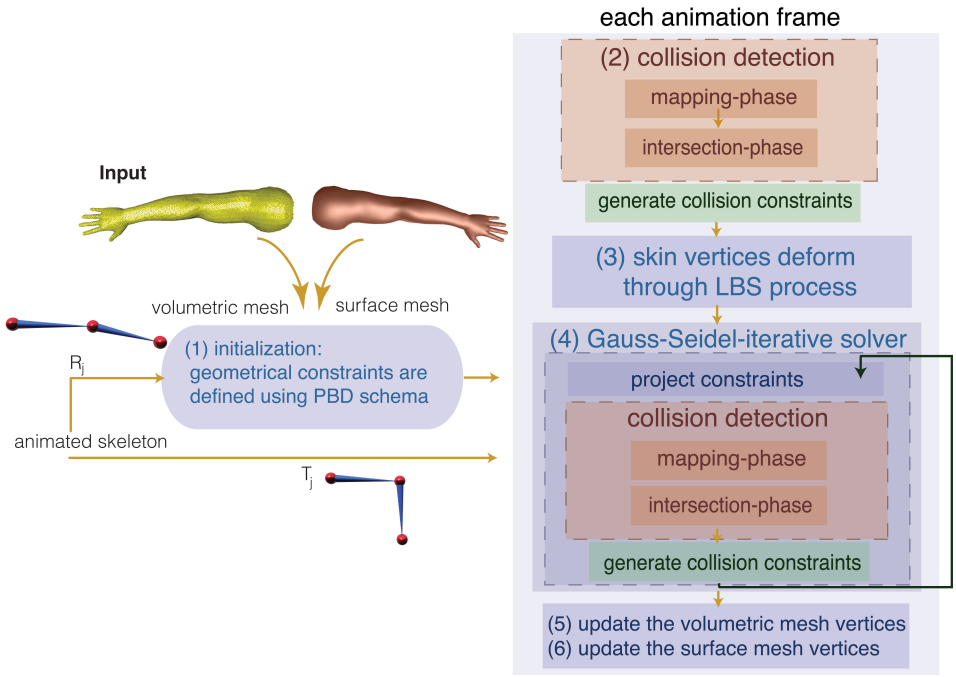


Figure 5.4: In the initialization phase, the fine surface mesh is converted to a tetrahedral mesh, which is used for computing the elastic deformation of the character skin while moving, as well as for collision detection. During the animation of the skeleton, the volumetric mesh is deformed with position based skinning (shown in purple) and the colliding vertices are computed by the collision detection algorithm (shown in orange). The collision information is used to generate collision response constraints (shown in green).

At every animation frame, in the deformation phase, the skin is deformed with position based skinning. Although position based skinning is unaffected by the artefacts of classic interactive skinning techniques, it cannot guarantee self-intersection free deformations. Therefore, the first step for handling these self-intersections is to detect the collisions. In the collision detection phase (Section 5.3), the collision detection algorithm gets the deformed vertices as input and computes the colliding vertices. The collision information is subsequently used to generate collision response constraints (Section 5.4). However, the vertices are moved during the latter constraints solving step and may encounter new collisions. Hence, we also detect the collisions inside of the solver loop, where collision constraints are generated from scratch at each iteration. In the collision response phase, we formulate a constraint-based response inside our position based skinning framework, to handle the collisions and capture contact deformations.

5.3 Collision Detection for Articulated Deformable Characters

In this section, we present a fast collision detection method for articulated deformable characters. Our method was inspired by the work of [Teschner *et al.*, 2003], but adapted for articulated characters. We exploit the skeletal nature of the deformation to obtain real time self-collision detection for models deformed by position based skinning (Chapter 4). We implicitly subdivide the space into uniform grids of axis-aligned bounding boxes (AABBs), called cells. All mesh primitives (vertices and tetrahedra) are classified with respect to these grids according to their position. Instead of using complex 3D data structures, such as kd-trees, octrees or BSP trees, a hash function is used to project the 3D cells into a finite 1D hash table. Only primitives mapped to the same hash index indicate a possible collision and need to be tested for intersections. The hash index can contain more than one primitive for the same mesh, which allows us to detect self-collisions as well (Fig. 5.5). Our method focuses on skeletal characters, where we use a lazy procedure that updates the hash table in an on-demand way. Therefore, it is not necessary to update the hash table in each time step. Instead, it is only updated when required by collision detection algorithm. This is not only memory efficient, but also leads to a significant improvement in performance. The intersection test is then carried out by computing the barycentric coordinates of a vertex with respect to a penetrated tetrahedron. Thereby, it provides the exact position of a vertex inside a penetrated tetrahedron and this information is used for collision response.

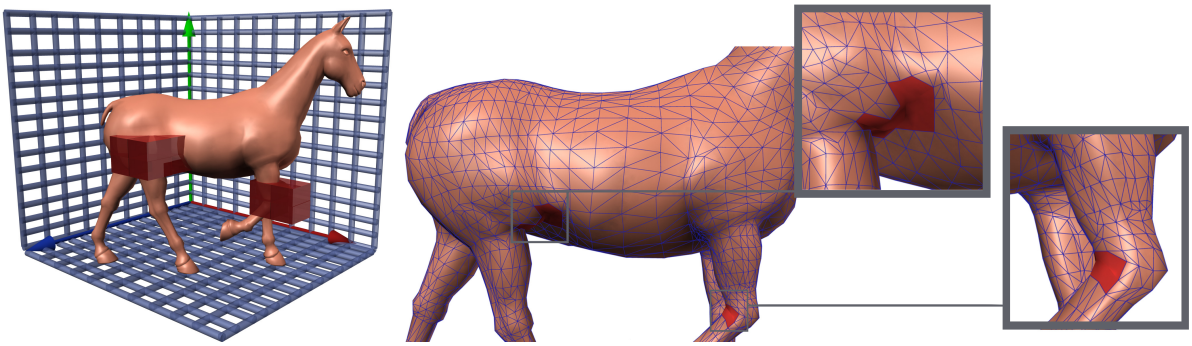


Figure 5.5: Self-collision detection. *Left.* The space is implicitly subdivided into small cells, where red cells contain fully or partially the self-colliding primitives. *Middle.* The red patches indicate self-collisions, in which intersections are quickly found with an $\mathcal{O}(1)$ cell query. *Right.* Zoomed view of self-colliding primitives (in red).

5.3.1 Optimized Spatial Partitioning

In order to detect collisions and self-collisions within the position based skinning method, we employ the *spatial hashing* procedure with temporal marks, introduced in [Teschner *et al.*, 2003]. To speed up the self-collision detection, we exploit the skeletal nature of the deformation to only update the hashing table when required. The use of a regular partition is suitable for our system, since all the tetrahedrons of the used models have about the same size. Therefore, we implicitly subdivide the space $\mathbb{R}^3$ into a uniform grid composed of small axis-aligned bounding boxes (AABBs), called cells. Each cell maintains a list of the mesh primitives (vertices and tetrahedrons) that are fully or partially contained in the cell. Rather than using complex 3D data structures, a hash function is used to map these cells to a finite number of hash table entries. The algorithm proceeds in two phases: the hashing phase (Section 5.3.1.1) and the intersection phase (Section 5.3.1.2).

5.3.1.1 Hashing Phase

In the hashing phase (Fig. 5.6), all mesh primitives are classified with respect to cells and mapped into hash table entries in uniformly random fashion. Hence, all vertices are mapped into their cell (Alg. 6), and all tetrahedrons are also mapped into the cells touched by their bounding box (Alg. 7). This hashing process is dependent on the following parameters:

- Table size: the optimal size is related to the number of primitives in the scene, and must be a high prime number in order to minimize the risk of mapping different

positions to the same hash index.

- Grid cell width: influences the number of mesh primitives that are mapped to the same hash index. Thus, a reasonable choice is to employ the tetrahedron's average edge length.

- Hash function: a function that maps a cell to an arbitrary hash table address. Simple and fast to execute hash function is preferable for spatial hashing. The following function is used:

$$h = hash(i,j,k) = (i\,u \oplus j\,v \oplus k\,w) \textbf{ mod } n \tag{5.1}$$

  where $\oplus$ stands for exclusive-or operation, i,j,k are grid coordinates, u, v and w are high prime numbers and n is the hash table size.

For example, a vertex with position $\mathbf{p} = (x,y,z)$ is mapped into a hash table of size n by computing its table index h as follows: $h = \left[\left(\lfloor\frac{x}{d}\rfloor \cdot u\right) \oplus \left(\lfloor\frac{y}{d}\rfloor \cdot v\right) \oplus \left(\lfloor\frac{z}{d}\rfloor \cdot w\right)\right]$ **mod** n, where for u,v,w we use the prime numbers 73856093, 19349663, 83492791, respectively. The value d is the cell size.

Algorithm 6: Simple pseudo-code that hashes the vertices position into the hash table

Input : *Vector* inputVertices, *Integer* cellSize, *Integer* tableSize
Output : *Vector* hashIndices

```
1: for each v in inputVertices do
2:    // map the vertex's position into a cell of the grid
3:    Integer discreteGrid_x = (Integer)floor((double) v.x/cellSize)
4:    Integer discreteGrid_y = (Integer)floor((double) v.y/cellSize)
5:    Integer discreteGrid_z = (Integer)floor((double) v.z/cellSize)
6:    // Obtain a hash value for the vertex's position
7:    hashIndices = (73856093 * discreteGrid_x ⊕ 19349663 *
8:    discreteGrid_y ⊕ 83492791 * discreteGrid_z) mod tableSize
9: end for
```

In order to avoid cleaning up the grid and perform the hashing phase from scratch in each frame. Temporal marks or so called *timestamps* are used to label each cell, where these marks are associated with the moment each cell was last updated. Thus, an intersection with primitives inside a given cell is considered only if it was updated in the current iteration.

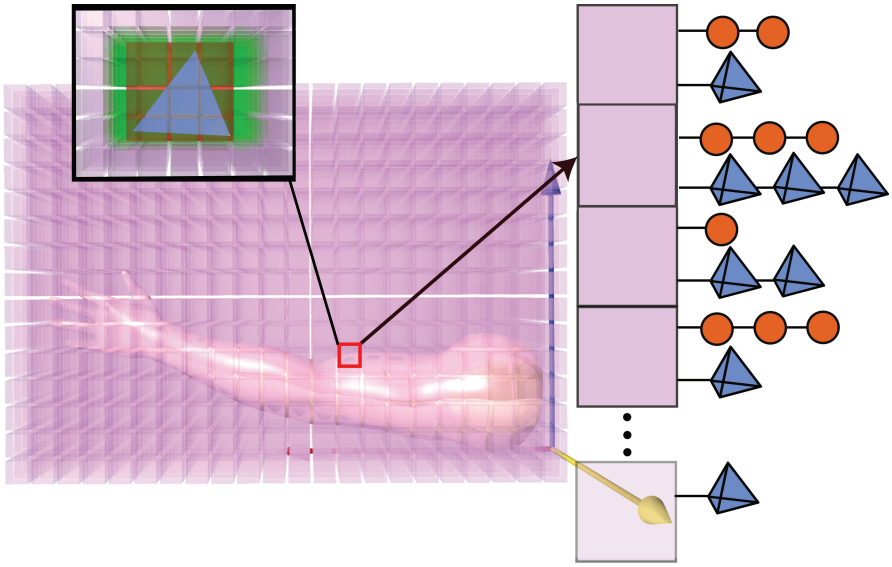


Figure 5.6: An example of 3D spatial hashing for an arm. *Left.* The arm mesh is embedded in a spatial partitioning. The zoomed view shows a tetrahedron (in blue), its bounding box (in red) and all cells affected by the tetrahedron's bounding box (in green). *Right.* In the hashing phase, all vertices of the arm mesh are mapped into their cell and the hash table indices are computed for all cells covered by the tetrahedron's bounding box. Therefore, in the intersection phase, the tetrahedron is checked for intersection with all vertices found at these hash indices.

Algorithm 7: Simple pseudo-code that hash tetrahedrons into the hash table

Input : *Vector* inputTetrahedra, *Grid*∗ gridIndices

```
1:  for each t in inputTetrahedra do
2:      // Compute the bounding box for a given tetrahedron
3:      BoundingBox BBox_tetrahedron = findBoundingBox(t)
4:      for each g_i in gridIndices do
5:          // Check if cell is affected by the tetrahedron's bounding box
6:          if intersection (BBox_tetrahedron, g_i) then
7:              gridIndices.insert(t.Index)
8:          end if
9:      end for
10: end for
```

5.3.1.2 Intersection Phase:

In a second phase, if a tetrahedron interferes with a cell, all associated vertices of that cell are checked for collision with the tetrahedron. To speed up the intersection test: we first test vertex **v** against the bounding box of tetrahedron **t**. If **v** is inside the bounding box of **t**, then an actual vertex/tetrahedron intersection test has to be performed. If an

intersection of a vertex **v** with a tetrahedron is detected and **v** is part of the same mesh, a self-intersection has been detected, but only if **v** is not part of the penetrated tetrahedron itself. The actual intersection test computes barycentric coordinates of a vertex with respect to the tetrahedron in order to detect, whether a vertex collides with the tetrahedron or not. This phase detects all colliding vertices in the scene and provides the exact position of a vertex inside a penetrated tetrahedron. This information can be employed to handle collisions by adding inequality constraints to the system of constraints within the position based skinning method.

The exact intersection test between a vertex position $\mathbf{p} = (x, y, z)$ and a tetrahedron **t** spanned by four vertices with the following positions:

$$\begin{aligned}\mathbf{p}_1 &= (x_1, y_1, z_1)\\ \mathbf{p}_2 &= (x_2, y_2, z_2)\\ \mathbf{p}_3 &= (x_3, y_3, z_3)\\ \mathbf{p}_4 &= (x_4, y_4, z_4)\end{aligned}$$

is performed by calculating barycentric coordinates of **p** with respect to **t**. These barycentric coordinates $\mathbf{b} = (b_1, b_2, b_3, b_4)$ are:

$$(b_1, b_2, b_3, b_4) = \left(\frac{D_1}{D}, \frac{D_2}{D}, \frac{D_3}{D}, \frac{D_4}{D}\right) \tag{5.2}$$

where the determinant D is computed as:

$$D = \begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix}$$

If $D = 0$, the tetrahedron is degenerated and the collision test is aborted. Otherwise, If $D > 0$ or $D < 0$ then D_1, D_2, D_3 and D_4 are computed as:

$$D1 = \begin{vmatrix} x & y & z & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix} \quad D_2 = \begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x & y & z & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix}$$

$$D_3 = \begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x & y & z & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix} \quad D_4 = \begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x & y & z & 1 \end{vmatrix}$$

Finally, the vertex position $\mathbf{p}$ is inside the tetrahedron $\mathbf{t}$ if the barycentric coordinates $\mathbf{b} = (b_1, b_2, b_3, b_4)$ of the vertex satisfies the following conditions:

$$b_1 \geq 0$$
$$b_2 \geq 0$$
$$b_3 \geq 0$$
$$b_4 \geq 0$$
$$b_1 + b_2 + b_3 + b_4 = 1$$

5.3.2 On-demand Hashing

Construction of the hash structure is performed in the hashing phase. This phase maps all primitives into a hash table in $\mathcal{O}(n)$ time. While the intersection phase takes $\mathcal{O}(n \cdot p \cdot q)$ where p is the average number of cells intersected by a tetrahedron and q is the average number of vertices per cell. In order to avoid constructing the hash table in each simulation step (which would reduce the efficiency in case of large tables), we used a *timestamp* to construct the table incrementally, inserted vertices are not removed from the table. Instead vertices are relocated whenever they move to another hash index. Our goal is to detect self-collisions on models deformed by position based skinning in an arbitrary posture. Self-collision is the most time consuming part of the collision detection phase. Hence, to speed up the self-collision detection process, we exploit the skeletal nature of the deformation. In particular, we exploit the fact that the only thing that changes the shape of the deformed skin during the animation are the bone rotations $\mathbf{T}_j$ (all other data are constant). It is therefore possible to base the hashing procedure solely on the actual bone rotations. Moreover, most movement modes of skeletal models require rotation of a body part around an axis that passes through the center of a joint, and such movements are called angular movements (Section 5.1).

The common angular movements involve either an increase or a decrease in the angle between the articulating bones. The principal angular movements are *flexion*, *extension*, *abduction* and *adduction*. Flexion motion refers to a decrease in the angle between articulating bones, while extension is the motion of increasing the joint angle between articulating bones (Fig. 5.7). Abduction is a movement of the limbs/extremities away

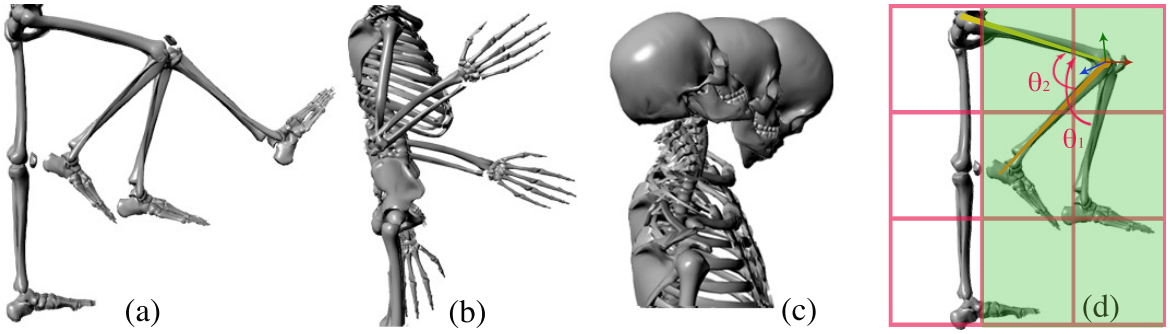


Figure 5.7: Examples of flexion and extension motions. Flexion brings two adjoining long bones closer to each other. While extension denotes rotation in the opposite direction of flexion. (a), (b) and (c) Show flexion and extension movements of the knee, elbow and neck joint, (d) shows the angle of the joint of the knee during flexion, where the angle θ_2 is indicating a possible self-collision. The hash table is partially reconstructed by considering only the cells that are affected by the bounding box of the part that indicates collisions (in green).

from the body, and adduction is movement toward the body (Fig. 5.8). Self-collisions for skeletal meshes occur in very localized regions, which are often found near joints. In addition to that, we observed that self-collisions usually happen during the flexion and adduction movements. Therefore, the main idea behind our on-demand hashing operation is to update the data structure (incrementally reconstruct the hash table) only during the flexion and adduction motions. Accordingly, we consider the angles θ_i between articulating bones during the animation, and we check if such angles would allow self-collisions. Since the number of bones is usually orders of magnitude smaller than the number of primitives, we perform a quick test based on the angles between the articulating bones (angular displacement of the bones during the animation). Thus we are able to discard reconstruction the hash table when such angles would not allow self-collisions. This is of course much more efficient than performing the hashing operation each time step. This quick test in case of flexion motion, relies on the relative angles between the articulating bones. In case of adduction, it relies on the angles between the bone segment and the midline of the character. The midline is estimated based on the bounding box of the body.

The relative angle is computed automatically as the arc cosine of the dot product of the two vectors: $\theta = arccos\frac{a \cdot b}{\|a\|\|b\|}$. If the angle is decreasing (i.e. $\theta_2 < \theta_1$ in Fig. 5.7 (d)) and less than a pre-specified tolerance angle α, then hash table is reconstructed and all primitives in same AABBs are tested for collisions. The tolerance angle α is in the range

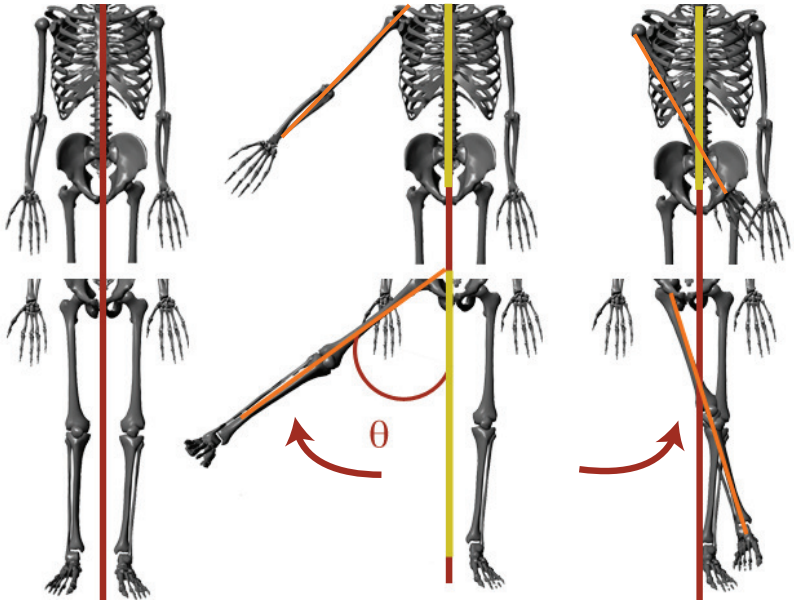


Figure 5.8: Examples of abduction and adduction motions. Abduction is the movement of a limb away from the midline. While adduction is the movement toward the midline. We compute the angle between the bone segment and the midline, in order to check whether the angle indicates a possible collision.

$\left[\frac{\pi}{3}, \pi\right]$, is defined by the user, and may be dependent on the character. Instead of entirely reconstructing the hash table, we reconstruct the table only in the part where an angle is indicating a possible self-collision. Thus, the hash table is partially reconstructed by only considering the primitives that are in the cells that affected by the bounding box of the potentially colliding part (Fig. 5.7 (d)). In other words, rather than reconstructing the hash table incrementally for the entire scene in each time step [Teschner *et al.*, 2003], we partially reconstruct the hash table based on the angular displacement of bones.

5.4 Localized Self-Collision Handling for Articulated Soft Characters

In this section, we locally address the self-intersection problem in the areas around the joints. As explained in Section 5.1, all joints in realistic characters are rotational, where the rotational movement occurs in a plane and around an axis. Consequently, in an initialization phase, we generate a plane $\mathbf{q}_i$ for each joint $\mathbf{j}_i$ of the skeleton that is parallel to the axis of rotation of the joint $\mathbf{j}_i$ (Fig. 5.9). Each plane has a constant size, which is not modified during the motion. While we use transformation constraints, in

order to align each plane translation and orientation with its corresponding joint during the motion (Fig. 5.10).

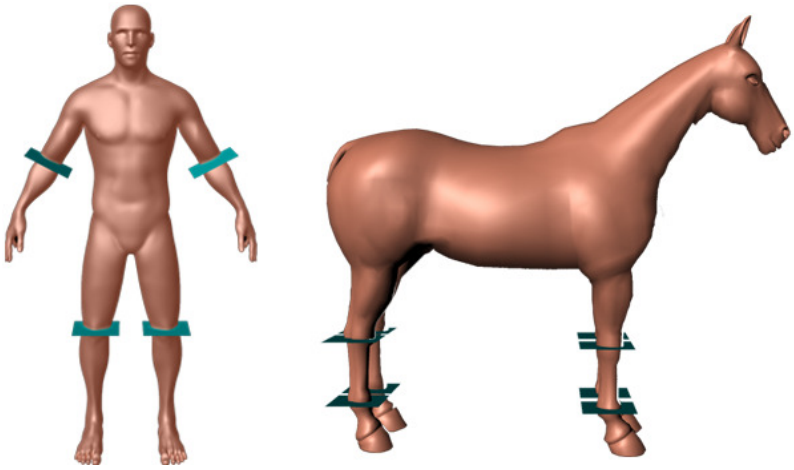


Figure 5.9: An example of the generation of planes for some joints of the skeleton for both bipedal and quadrupedal characters, in order to address the self-intersection problem between the skin parts.

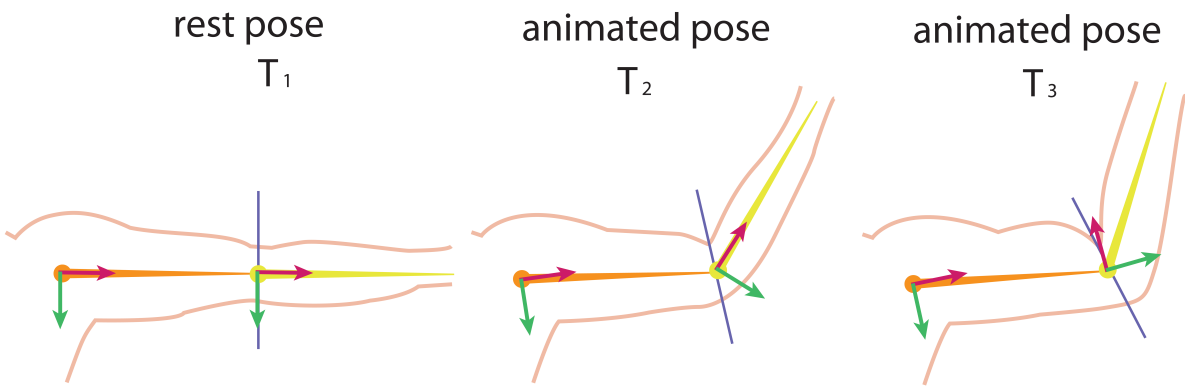


Figure 5.10: Skin is deformed by position based skinning. *Left.* The skeleton is embedded within the mesh in its initial position, where the plane is also attached to the elbow joint. *Middle.* The collisions between the skin parts and the plane are detected using the method described in Section 5.3 *Right.* Collision response constraints push the skin along the plane normal, while the tetrahedral volume constraints preserve the volume of the skin, leading to a localized muscle bulging effect.

In this configuration, planes separate the skin parts. Hence the collision between two skin parts is defined by the contact point and orientation of the plane. The intersection test described in Section 5.3 returns these contact data. Inside the position based

skinning framework, we simulate the skin as a soft body that is represented by a set of particles. In order to handle collisions on the detected candidate contact plane, for each particle an inequality constraint $C(\mathbf{p}) \geq 0$ is introduced into the position based skinning system, where

$$C(\mathbf{p}) = (\mathbf{p} - \mathbf{q}_c) \cdot \mathbf{n} \tag{5.3}$$

In this equation $\mathbf{q}_c$ is the contact point, $\mathbf{n}$ is the normal of the plane. The stiffness of this constraint is equal to 1. Responding to the collision in this case is simply a matter of pushing the particle position out to the surface of the plane along its normal, where the correction of the particle position is:

$$\Delta\mathbf{p} = -\mathbf{n}\|\mathbf{p} - \mathbf{q}_c\| \tag{5.4}$$

These additional constraints allow us to obtain localized self-intersection free deformation in the areas around the joint. The collision response constraints push the skin along the plane normal, while the tetrahedral volume constraints preserve the volume of the skin. Consequently, the system of geometric constraints within the PBS framework, including tetrahedral volume and collision response constraints is able to capture muscle bulging behavior (Fig. 5.10).

5.5 Overall Algorithm

In this section, we summarize our method (in Alg. 8).

5.6 Experiments and Results

All experiments described in this section have been performed on a mass-market laptop equipped with an Intel i5 2.50 GHz processor and 4GB RAM. We implemented the on-demand hashing and the collision response in C++, and we tested our method on a variety of articulated characters of different shapes. The animations are adapted from [Abu Rumman & Fratarcangeli, 2015].

5.6.1 Collision Detection

We have evaluated our on-demand collision detection method on a variety of examples. We also compare the performance of our method with the spatial hashing algorithm.

Algorithm 8: Collision handling within position based skinning algorithm

Input : *Mesh* surfaceMesh, *Matrices4x4* bindingweights,
// The Skeleton's data structure consists of the following:
// *Matrices4x4* currentbonesTransformations,
//*Matrices4x4* restbonesTransformations and $Vector$ Joints
Skeleton inputSkeleton,
Double Δt,
Integer solverIterations

Output : *Mesh* surfaceMesh

```
generatetetrahedralMesh(Mesh surfaceMesh)
findBarycentricCoordinateForVertex(surfaceMesh,tetrahedralMesh)
for each t_i in tetrahedralMesh do
    defineGeometricConstraints(TetrahedralVolume)
end for
for each v_i in tetrahedralMesh do
    defineGeometricConstraints(Bind)
end for
```

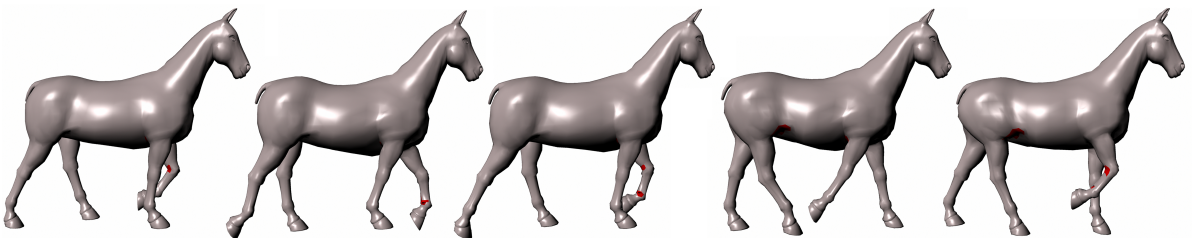


Figure 5.11: Real-time self-collision detection for an articulated character deformed by position based skinning: an animated sequence of a walking HORSE with a skeleton of 43 bones, 4K vertices and 6K tetrahedrons. All self-collisions are calculated in 2.1 ms per frame, where self-collisions shown in red.

Spatial hashing updates the whole hash table after every frame, which can be inefficient. On the other hand, our method uses a lazy procedure that updates the hash table in an on-demand way, which outperforms spatial hashing (see Table 5.1). Figs. 5.11 and 5.12 show the HORSE and HUMAN models animated with a walking cycle. All colliding vertices are detected in real-time and the mean computation times are reported in Table 5.1, where we used 10 ms time steps and 12 iterations. The grid width that we employed was slightly higher than the tetrahedron's average edge length.
Figs. 5.14 and 5.13 show ARM and LEG models while bending, where the number of iteration was 8 and all self-collisions are detected in real-time. The last scenario is a posture of a bending ARM and a RIGID BODY (Fig. 5.15). Our method successfully detected the collisions between the ARM model and the RIGID BODY, as well as the self-collisions

for each e_i in tetrahedralMesh **do**
  defineGeometricConstraints(Stretch)
end for
// For each joint position, we generate a quad
for each j_i in inputSkeleton.Joints **do**
  // Generate a quad mesh for each joint in the skeleton
  Quad q_i= generateQuad(j_i)
  // Define a parent constraint between the quad and corresponding joint
  defineParentConstraints(j_i,q_i)
end for
for each animation frame **do**
  // Loop through every vertex and compute blended positions through LBS
  for each v_i in tetrahedralMesh **do**
    $deformedVertices_i = inputVertices_i + bindingweights_{i,j} \times$
    $currentbonesTransformations \times restbonesTransformations_j^{-1}$
  end for
  // Simulating the skin as a soft body using PBD
  for each v_i in tetrahedralMesh **do**
    $\mathbf{v}_i = \mathbf{v}_i^0$
    $\mathbf{x}_i = \mathbf{x}_i^0$
    $w_i = 1/m_i$
  end for
  loop
    for each v_i in tetrahedralMesh **do**
      // We apply gravity as external force
      $\mathbf{v}_i = \mathbf{v}_i + \Delta t w_i \mathbf{F}_{ext}(x_i)$
      $\mathbf{p}_i = \mathbf{x}_i + \Delta t \mathbf{v}_i$
    end for
    detectCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
    for all collidedVertices i **do**
      generateCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
    end for
    for solverIterations **do**
      projectTetrahedralVolumeConstraints($\mathbf{C}_1,\ldots,\mathbf{C}_m,\mathbf{p}_i,\ldots,\mathbf{p}_n$)
      projectBindConstraints($\mathbf{C}_1,\ldots,\mathbf{C}_m,\mathbf{p}_i,\ldots,\mathbf{p}_n$)
      projectStretchConstraints($\mathbf{C}_1,\ldots,\mathbf{C}_m,\mathbf{p}_i,\ldots,\mathbf{p}_n$)
      projectCollisionConstraints($\mathbf{C}_1,\ldots,\mathbf{C}_m,\mathbf{p}_i,\ldots,\mathbf{p}_n$)
      detectCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
      for all collidedVertices i **do**
        generateCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
      end for
    end for
    for all each v_i in tetrahedralMesh **do**
      $\mathbf{v}_i = (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
      $\mathbf{x}_i = \mathbf{p}_i$
    end for
    velocityUpdate($\mathbf{v}_i,\ldots,\mathbf{v}_n$)
  end loop
  UpdatesurfaceMeshVertices()
end for

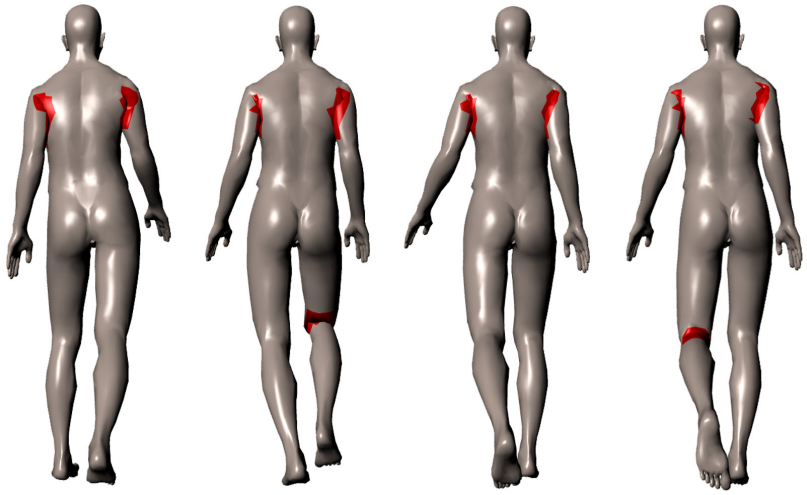


Figure 5.12: A back view of a walking HUMAN with a skeleton of 25 bones, 9K vertices and 5K tetrahedrons. All the colliding vertices are computed in 3.93 ms per frame, where the red patches indicate self-collisions.

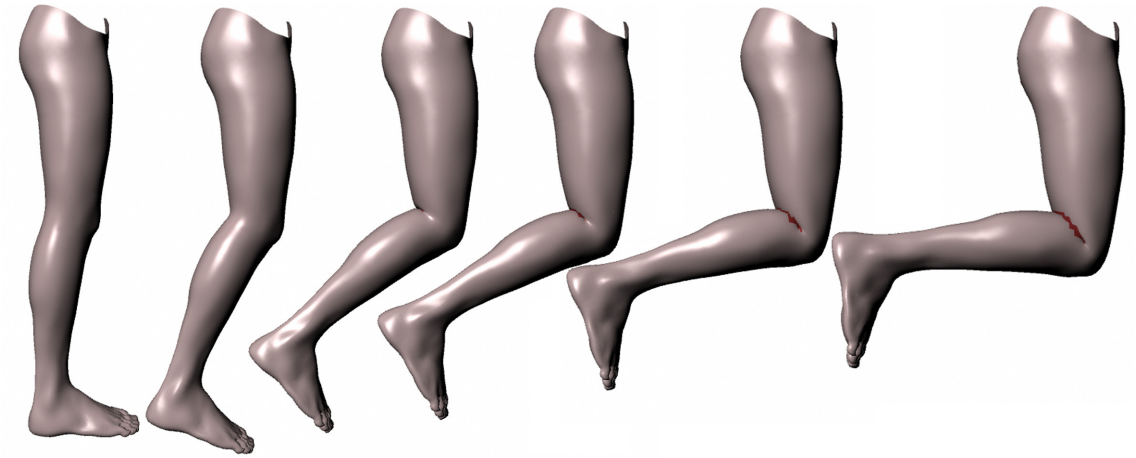


Figure 5.13: An animated sequence of a bending LEG, where all colliding vertices are computed in 1.7 ms per frame. Self-collisions shown in red.

on the ARM model. Please refer to the video for the complete animations and additional results.

5.6.2 Collision Response

We demonstrate the deformation results of our collision response method with two different models. The handling of self-collisions is shown in Fig. 5.16. In our experiments, we used a 0.93 stiffness value for the collision constraints. Table 5.2 reports the mean computation times of our method for ARM and LEG bending motion, including the skinning

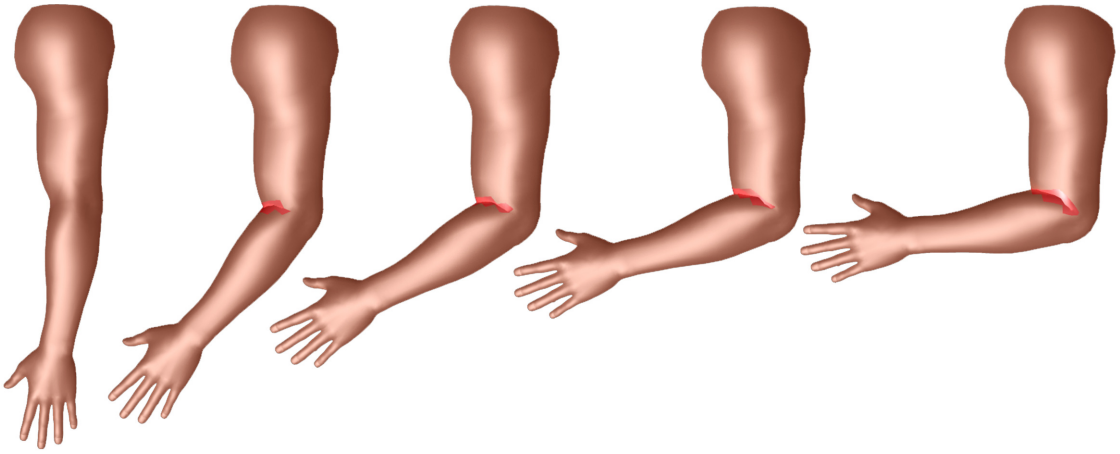


Figure 5.14: An animated sequence of a bending ARM, where all self-collisions are calculated in 1.702 ms per frame.

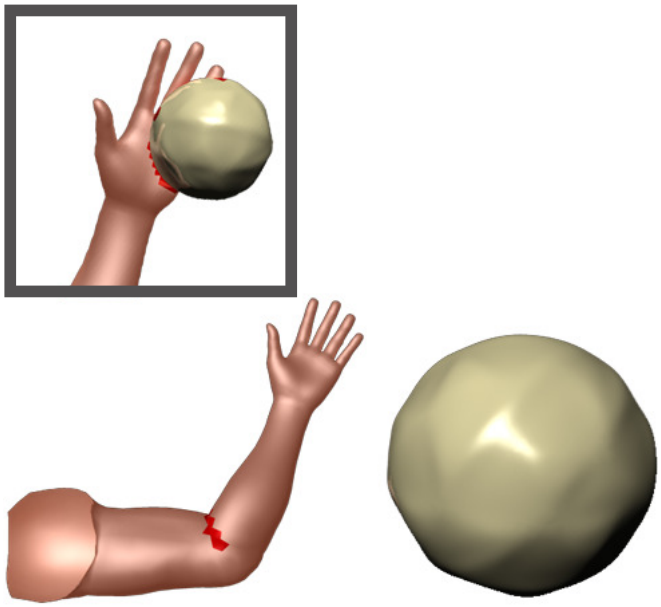


Figure 5.15: A bending ARM and a RIGID BODY, consisting of 4400 tetrahedrons and 132 tetrahedrons, respectively. Both the collisions and the self-collisions of the arm are detected in 2.52 ms.

simulation, collision detection and collision response. We did not include forces/torques in our collision response formulation. However, we present a simple and institutive method, which handles self-intersections in real-time and offers considerable benefit such as the bulging behavior.

scene	# vertices	# tetra	# bones	$CT_{skinning}$ [ms]	$CT_{on-demandHashing}$ [ms]	CT_{total} [ms]	# iterations	$fps_{on-demandHashing}$	$CT_{SpatialHashing}$	$fps_{SpatialHashing}$
HUMAN	9528	4998	25	6.807	3.93	10.737	12	93.136	5.593	80.64
ARM	4211	4400	3	4.112	1.702	5.814	8	171.998	2.53	150.557
ARM + RIGID BODY	4284	4532	3	3.792	2.52	6.312	8	158.429	3.10	145.095
LEG	3990	4339	3	4.108	1.7	5.808	8	172.176	2.47	152.021
HORSE	3833	6126	43	7.69	2.1	9.79	12	102.145	3.35	90.579

Table 5.1: Performance comparison of our on-demand collision detection and optimized spatial hashing. #vertices: number of initial vertices in the render mesh, #tetra: number of elements in the tetrahedral mesh, $fps_{on-demandHashing}$: avg. frame rate, $CT_{skinning}$: avg. skinning computation time and $CT_{on-demandHashing}$: avg. computation time of our collision detection method during 1 sec simulation, where ($CT_{total} = CT_{skinning} + CT_{on-demandHashing}$). $CT_{SpatialHashing}$: avg. computation time of optimized spatial hashing, $fps_{SpatialHashing}$: avg. frame rate of using optimized spatial hashing to detect collisions on models deform by position based skinning.

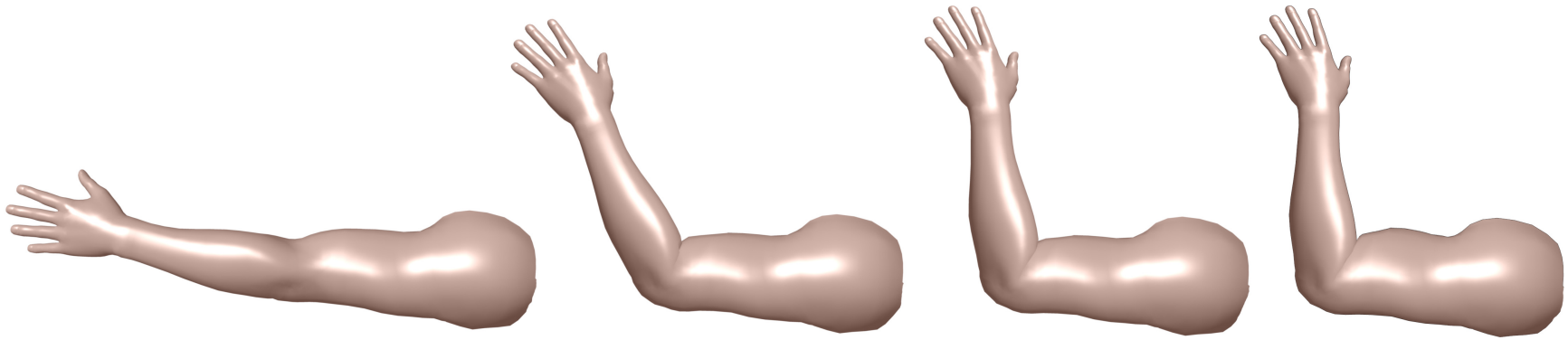


Figure 5.16: An animated sequence of a bending ARM, where our methods handles the self-collision of upper and lower arm and successfully preserves the volume.

scene	# vertices	# tetra	# bones	$CT_{skinning}$ [ms]	$CT_{on-demandHashing}$ [ms]	$CT_{collision_response}$ [ms]	CT_{total} [ms]	# iterations	fps
ARM	4211	4400	3	6.01	3.501	4.9	14.411	22	69.391
LEG	3990	4339	3	6.09	3.69	4.53	14.31	22	69.99

Table 5.2: Collision response performance. #vertices: number of initial vertices in the render mesh, #tetra: number of elements in the tetrahedral mesh, fps: avg. frame rate, $CT_{skinning}$: avg. skinning computation time, $CT_{on-demandHashing}$: avg. computation time of collision detection, $CT_{collisionResponse}$: avg. computation time of our collision response during 1 sec simulation, where ($CT_{total} = CT_{skinning} + CT_{on-demandHashing} + CT_{collisionResponse}$).

5.7 Conclusion

In the previous chapter, we presented interactive deformation model (position based skinning, Chapter 4), which provides interesting effects, such as volume preservation and jiggling. Unfortunately, PBS suffers from self-intersections, thus it fails to produce convincing organic-like deformations near joints. This chapter presents a simple and fast collision detection method for articulated deformable meshes, which used to locally handle in real-time self-intersection on models deform by PBS. During the animation, the skin is deformed using position based skinning. Then, the collision detection algorithm gets the deformed vertices as input and finds all colliding vertices. Detecting the collisions is based on spatial hashing [Teschner *et al.*, 2003], and is performed before and inside the constraint enforcement loop of position based skinning. Thus, our method does not miss collision events during the solver loop. The intersection test provides the exact position of a vertex inside a penetrated tetrahedron, and this information is employed to estimate the contact point, and to compute collision response. Being based on spatial hashing [Teschner *et al.*, 2003], our collision detection method does not rely on any preprocessing and it does not impose requirements on the characteristics of the meshes. We exploit the skeletal nature of the deformation to incrementally reconstruct the hash table only when required, using the on-demand hashing operation. This on-demand hashing operation speeds up the full collision detection considerably when compared to the original hashing operation [Rumman *et al.*, 2015].

Collision response is done by detecting collisions between skin parts and planes, where each plane is attached with its corresponding joint in the skeleton. Then, we generate temporary inequality constraints on-the-fly, and including them into the system of constraints inside PBS framework. The geometric constraints within PBS including tetrahedral volume and collision response constraints are able to capture muscles bulging behavior. However, The inconsistency between these constraints may lead to noticeable artefacts.

CHAPTER 6

CONCLUSION AND FUTURE DIRECTIONS

Believable skin deformation for soft articulated characters is essential to enrich the visual experience of the animation and for creating appealing character animation in movie productions, computer games, and virtual reality applications. The production of life-like deformations includes capturing a range of desirable effects, like secondary motions, volume preservation, and contact deformations. In character animation, the skin deformation of an articulated character is determined primarily by an underlying skeleton, which leads to purely kinematic deformations. However, it is important to simulate the secondary motion effects, as well as skin response due to collisions. But, these effects are usually challenging to achieve in real time. Various techniques have been proposed for performing skeleton-driven deformations, and others with physically based skin deformation. However, none of these approaches addresses the problem of creating believable skin deformation for soft articulated characters at interactive rates. While also efficiently handling local collision response to capture skin contact deformations.

In this dissertation, we have presented a novel two-layered deformation model for soft articulated characters, called position based skinning (PBS). For each frame, the skin is first deformed with a classic linear blend skinning (LBS) approach, which usually

leads to unsightly artefacts like the well-known candy-wrapper effect and volume loss. Then, we enforce some geometric constraints inside position based dynamics (PBD) scheme, which displace the positions of the vertices to mimic the behavior of the skin. Our system initializes the blend weights and the soft constraints automatically, so it does not require a considerable set-up effort. Linear blend skinning deformer, improves the convergence speed of the PBD solver, while PBD maintaining the elastic nature of the character body. We employ a graph coloring algorithm for parallelizing the computation of the geometric constraints. This leads to fast performances even in case of a fairly high number of tetrahedrons. We mapped the tetrahedral mesh to the input skin geometry for achieving high quality renderings. During the animation, our method preserves the volume and allows for passive jiggling behavior. Our method allows the artists to define specific areas of the body and decide on the amount of jiggling affecting them by tuning a single scalar stiffness parameter.

To produce a convincing organic-like deformation near joints. We have presented a simple and fast collision detection method for articulated deformable meshes, which used to handle locally self-intersection on models deform by position based skinning (PBS). During the animation, the skin is deformed using PBS. Then, the collision detection algorithm gets the deformed vertices as input and finds all the colliding vertices. Detecting the collisions is based on spatial hashing [Teschner *et al.*, 2003], and is performed before and inside the constraint enforcement loop of PBS. Thus, our method does not miss collision events during the solver loop. The intersection test provides the exact position of a vertex inside a penetrated tetrahedron and this information is employed to estimate the contact point, and to compute collision response. Being based on spatial hashing [Teschner *et al.*, 2003], our collision detection method does not rely on any preprocessing and it does not impose requirements on the characteristics of the meshes. We exploit the skeletal nature of the deformation to incrementally reconstruct the hash table only when required using the on demand hashing operation. The on demand hashing speeds up the collision detection algorithm considerably when compared to the optimized spatial hashing method.

Collision response is done by detecting collisions between skin parts and quads, where each quad is attached with its corresponding joint in the skeleton. Then, we generate temporary inequality constraints on-the-fly and including them into the system of constraints inside PBS framework. The geometric constraints within PBS including tetrahedral volume and collision response constraints are able to capture muscles bulging behavior.

However, the violent conflict between these constraints may lead noticeable artefacts.

6.1 Discussion and Future Work

In contrast to the existing methods, our system does not model the inner structure of the human skin. However, using a combination of widely known and relatively simple techniques, linear blend skinning and position based dynamics, we achieved believable animations with a time performance suitable for interactive applications. Being based on position based dynamics, the elastic behavior of the soft body deformer is influenced by the number of iterations employed in the parallel Gauss-Seidel solver. We employed 12 iterations during our tests, but in general the artist has to heuristically choose a value which depends on the topology and the polygonal resolution of the input mesh.

In our collision handling algorithm, we only consider the special case of self-intersection near joints. Therefore, some assumptions will not apply in general case. For example, an arm may still touch legs during an extension movement, but the method will not update the hashing for this case. Then, the algorithm will not handle the collisions.

The geometric constraints inside PBS may suffer violence, which may lead to inconsistency and noticeable artefacts. To have more stable and accurate deformations, we will accelerate our method using a GPU implementation based on [Fratarcangeli & Pellacini, 2013].

References

Abu Rumman, Nadine, & Fratarcangeli, Marco. 2014. Position Based Skinning of Skeleton-driven Deformable Characters. *Pages 83–90 of: Proceedings of the 30th Spring Conference on Computer Graphics*. SCCG '14. New York, NY, USA: ACM.

Abu Rumman, Nadine, & Fratarcangeli, Marco. 2015. Position-Based Skinning for Soft Articulated Characters. *Computer Graphics Forum*, **34**(6), 240–250.

Alcantara, Dan A., Sharf, Andrei, Abbasinejad, Fatemeh, Sengupta, Shubhabrata, Mitzenmacher, Michael, Owens, John D., & Amenta, Nina. 2009. Real-time Parallel Hashing on the GPU. *Pages 154:1–154:9 of: ACM SIGGRAPH Asia 2009 Papers*. SIGGRAPH Asia '09. New York, NY, USA: ACM.

Alexa, Marc. 2002. Linear Combination of Transformations. *Pages 380–387 of: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '02. New York, NY, USA: ACM.

Allen, Brett, Curless, Brian, & Popović, Zoran. 2002. Articulated Body Deformation from Range Scan Data. *Pages 612–619 of: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '02. New York, NY, USA: ACM.

Angelidis, Alexis, & Singh, Karan. 2007. Kinodynamic skinning using volume-preserving deformations. *Pages 129–140 of: Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA 2007, San Diego, California, USA, August 2-4, 2007*.

Aubert, Fabrice, & Bechmann, Dominique. 1997. Volume-preserving Space Deformation. *Comput. Graph.*, **21**(5), 625–639.

Autodesk, Inc. 1990–2015. *3ds Max*. http://www.autodesk.it/products/3ds-max/overview.

Autodesk, Inc. 1998–2015. *Maya*. http://www.autodesk.com/products/maya/overview.

Baraff, David. 1996. Linear-time Dynamics Using Lagrange Multipliers. *Pages 137–146 of: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. New York, NY, USA: ACM.

Baran, Ilya, & Popović, Jovan. 2007. Automatic Rigging and Animation of 3D Characters. *In: ACM SIGGRAPH 2007 Papers*. SIGGRAPH '07. New York, NY, USA: ACM.

Bender, Jan, Müller, Matthias, Otaduy, Miguel A., & Teschner, Matthias. 2013. Position-based Methods for the Simulation of Solid Objects in Computer Graphics. *In: EUROGRAPHICS 2013 State of the Art Reports*. Eurographics Association.

Bender, Jan, M"uller, Matthias, Otaduy, Miguel A., Teschner, Matthias, & Macklin, Miles. 2014. A Survey on Position-Based Simulation Methods in Computer Graphics. *Computer Graphics Forum*, 1–25.

Bender, Jan, Müller, Matthias, & Macklin, Miles. 2015. Position-Based Simulation Methods in Computer Graphics. *In: EUROGRAPHICS 2015 Tutorials*. Eurographics Association.

Bharaj, Gaurav, Thorm"ahlen, Thorsten, Seidel, Hans-Peter, & Theobalt, Christian. 2012. Automatically Rigging Multi-component Characters. *Comp. Graph. Forum*, **31**(2pt3), 755–764.

Bloor, M. I. G., & Wilson, M. J. 1990. Using Partial Differential Equations to Generate Free-form Surfaces: 91787. *Comput. Aided Des.*, **22**(4), 202–212.

Boissonnat, Jean-Daniel, & Oudot, Steve. 2005. Provably Good Sampling and Meshing of Surfaces. *Graph. Models*, **67**(5), 405–451.

Botsch, Mario, & Kobbelt, Leif. 2003. Multiresolution Surface Representation Based on Displacement Volumes. *Computer Graphics Forum*, **22**(3), 483–491.

Bridson, Robert, Fedkiw, Ronald, & Anderson, John. 2002. Robust Treatment of Collisions, Contact and Friction for Cloth Animation. *ACM Trans. Graph.*, **21**(3), 594–603.

Bro-nielsen, Morten, & Cotin, Stephane. 1996. Real-time Volumetric Deformable Models for Surgery Simulation using Finite Elements and Condensation. *Pages 57–66 of: Computer Graphics Forum*.

Capell, Steve, Green, Seth, Curless, Brian, Duchamp, Tom, & Popović, Zoran. 2002. Interactive Skeleton-driven Dynamic Deformations. *Pages 586–593 of: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '02. New York, NY, USA: ACM.

Capell, Steve, Burkhart, Matthew, Curless, Brian, Duchamp, Tom, & Popović, Zoran. 2005. Physically Based Rigging for Deformable Characters. *Pages 301–310 of: Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '05. New York, NY, USA: ACM.

Chadwick, J. E., Haumann, D. R., & Parent, R. E. 1989. Layered Construction for Deformable Animated Characters. *SIGGRAPH Comput. Graph.*, **23**(3), 243–252.

Chen, Cheng-Hao, Lin, I-Chen, Tsai, Ming-Han, & Lu, Pin-Hua. 2011. Lattice-Based Skinning and Deformation for Real-Time Skeleton-Driven Animation. *Pages 306–312 of: Proceedings of the 2011 12th International Conference on Computer-Aided Design and Computer Graphics*. CADGRAPHICS '11. Washington, DC, USA: IEEE Computer Society.

Chen, David T., & Zeltzer, David. 1992. Pump It Up: Computer Animation of a Biomechanically Based Model of Muscle Using the Finite Element Method. *Pages 89–98 of: Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '92. New York, NY, USA: ACM.

Chen, Y., Zhu, Qing-Hong, Kaufman, A., & Muraki, S. 1998 (Jun). Physically-based animation of volumetric objects. *Pages 154–160 of: Computer Animation 98. Proceedings*.

Clifford, W. 1882 (jun). *Mathematical Papers*. Tech. rept.

Coleman, Thomas F., & More, Jorge J. 1983. Estimation of sparse Jacobian matrices and graph coloring problems. *Journal of Numerical Analasis*, **20**, 187–209.

Coquillart, Sabine. 1990. Extended Free-form Deformation: A Sculpturing Tool for 3D Geometric Modeling. *Pages 187–196 of: Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '90. New York, NY, USA: ACM.

De Aguiar, Edilson, Theobalt, Christian, Thrun, Sebastian, & Seidel, Hans-Peter. 2008. Automatic Conversion of Mesh Animations into Skeleton-based Animations. *Pages 389–397 of: Computer Graphics Forum*, vol. 27. Wiley Online Library.

Demeure, Virginie, Niewiadomski, Rados law, & Pelachaud, Catherine. 2011. How is Believability of a Virtual Agent Related to Warmth, Competence, Personification, and Embodiment? *Presence: Teleoper. Virtual Environ.*, **20**(5), 431–448.

Desbrun, Mathieu, & Gascuel, Marie-Paule. 1995. Animating Soft Substances with Implicit Surfaces. *Pages 287–290 of: Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '95. New York, NY, USA: ACM.

Dionne, Olivier, & de Lasa, Martin. 2013. Geodesic Voxel Binding for Production Character Meshes. *Pages 173–180 of: Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '13. New York, NY, USA: ACM.

Dong, Feng, Clapworthy, G.J., Krokos, M.A., & Yao, Jialiang. 2002. An anatomy-based approach to human muscle modeling and deformation. *Visualization and Computer Graphics, IEEE Transactions on*, **8**(2), 154–170.

Eitz, Mathias, & Lixu, Gu. 2007. Hierarchical Spatial Hashing for Real-time Collision Detection. *Pages 61–70 of: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2007*. SMI '07. Washington, DC, USA: IEEE Computer Society.

Ericson, Christer. 2004. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology) (The Morgan Kaufmann Series in Interactive 3D Technology)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Erleben, Kenny, Sporring, Jon, Henriksen, Knud, & Dohlman, Kenrik. 2005. *Physics-based Animation (Graphics Series)*. Rockland, MA, USA: Charles River Media, Inc.

Feng, Wei-Wen, Kim, Byung-Uck, & Yu, Yizhou. 2008. Real-time Data Driven Deformation Using Kernel Canonical Correlation Analysis. *Pages 91:1–91:9 of: ACM SIGGRAPH 2008 Papers*. SIGGRAPH '08. New York, NY, USA: ACM.

Forstmann, Sven, & Ohya, Jun. 2006. Fast skeletal animation by skinned arc-spline based deformation. *EG 2006 Short Papers*, 1–4.

Forstmann, Sven, Ohya, Jun, Krohn-Grimberghe, Artus, & McDougall, Ryan. 2007. Deformation Styles for Spline-based Skeletal Animation. *Pages 141–150 of: Proceedings*

of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. SCA '07. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association.

Fratarcangeli, M., & Pellacini, F. 2015. Scalable Partitioning for Parallel Position Based Dynamics. *Computer Graphics Forum*, **34**(2), 405–413.

Fratarcangeli, Marco. 2012. Position-based facial animation synthesis. *Computer Animation and Virtual Worlds*, **23**(3-4), 457–466.

Fratarcangeli, Marco, & Pellacini, Fabio. 2013. A GPU-Based Implementation of Position Based Dynamics for Interactive Deformable Bodies. *Journal of Graphics Tools*, **17**(3), 59–66.

Gain, James, & Bechmann, Dominique. 2008. A Survey of Spatial Deformation from a User-centered Perspective. *ACM Trans. Graph.*, **27**(4), 107:1–107:21.

Galoppo, Nico, Otaduy, Miguel A., Tekin, Serhat, Gross, Markus H., & Lin, Ming C. 2007. Soft Articulated Characters with Fast Contact Handling. *Comput. Graph. Forum*, **26**(3), 243–253.

Gao, Ming, Mitchell, Nathan, & Sifakis, Eftychios. 2014. Steklov-Poincaré Skinning. *Pages 139–148 of: The Eurographics / ACM SIGGRAPH Symposium on Computer Animation, SCA '14, Copenhagen, Denmark, 2014.*

Geijtenbeek, T., & Pronost, N. 2012. Interactive Character Animation Using Simulated Physics: A State-of-the-Art Review. *Computer Graphics Forum*, **31**(8), 2492–2515.

Gibson, Sarah F. F., & Mirtich, Brian. 1997. *A survey of deformable modeling in computer graphics*. Tech. rept. TR-97-19, MERL, Cambridge, MA, 1997.

Gilles, Benjamin, Bousquet, Guillaume, Faure, Francois, & Pai, Dinesh K. 2011a. Frame-based Elastic Models. *ACM Trans. Graph.*, **30**(2), 15:1–15:12.

Gilles, Benjamin, Bousquet, Guillaume, Faure, Francois, & Pai, Dinesh K. 2011b. Frame-based Elastic Models. *ACM Trans. Graph.*, **30**(2), 15:1–15:12.

Girard, Michael, & Maciejewski, A. A. 1985. Computational Modeling for the Computer Animation of Legged Figures. *Pages 263–270 of: Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '85. New York, NY, USA: ACM.

Goktekin, Tolga G., Reisch, Jon, Peachey, Darwyn, & Shah, Apurva. 2007. An Effects Recipe for Rolling a Dough, Cracking an Egg and Pouring a Sauce. *In: ACM SIGGRAPH 2007 Sketches*. SIGGRAPH '07. New York, NY, USA: ACM.

Gottschalk, S., Lin, M. C., & Manocha, D. 1996. OBBTree: A Hierarchical Structure for Rapid Interference Detection. *Pages 171–180 of: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. New York, NY, USA: ACM.

Gottschalk, Stefan Aric. 2000. *Collision Queries Using Oriented Bounding Boxes*. Ph.D. thesis. AAI9993311.

Gourret, J.-P., Thalmann, N. M., & Thalmann, D. 1989. Simulation of Object and Human Skin Formations in a Grasping Task. *Pages 21–30 of: Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '89. New York, NY, USA: ACM.

Govindaraju, Naga K., Knott, David, Jain, Nitin, Kabul, Ilknur, Tamstorf, Rasmus, Gayle, Russell, Lin, Ming C., & Manocha, Dinesh. 2005. Interactive collision detection between deformable models using chromatic decomposition. *ACM Trans. Graph*, **24**, 991–999.

Gross, Ralph, & Shi, Jianbo. 2001. *The CMU Motion of Body (MoBo) Database*. Tech. rept. CMU-RI-TR-01-18. Robotics Institute, Carnegie Mellon University.

Guskov, Igor, Sweldens, Wim, & Schröder, Peter. 1999. Multiresolution Signal Processing for Meshes. *Pages 325–334 of: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '99. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

Hahn, Fabian, Martin, Sebastian, Thomaszewski, Bernhard, Sumner, Robert, Coros, Stelian, & Gross, Markus. 2012. Rig-space Physics. *ACM Trans. Graph.*, **31**(4), 72:1–72:8.

Hahn, Fabian, Thomaszewski, Bernhard, Coros, Stelian, Sumner, Robert, & Gross, Markus. 2013. Efficient simulation of secondary motion in rig-space. *In: Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA '13.

He, Liang, Ortiz, Ricardo, Enquobahrie, Andinet, & Manocha, Dinesh. 2015. Interactive continuous collision detection for topology changing models using dynamic clustering.

Pages 47–54 of: Proceedings of the 19th Symposium on Interactive 3D Graphics and Games, San Francisco, CA, USA, February 27 - March 01, 2015.

Heim, Oliver, Marshall, Carl, & Lake, Adam. 2004. Fast Collision Detection for 3D Bones-Based Articulated Characters. *Pages 503–514 of:* Kirmse, Andrew (ed), *Game Programming Gems 4*. Charles River Media.

Hejl, Jim. 2004. Hardware Skinning with Quaternions. *Pages 487–495 of:* Kirmse, Andrew (ed), *Game Programming Gems 4*. Charles River Media.

Hong, Min, Jung, S., Choi, Min-Hyung, & Welch, S.W.J. 2006. Fast Volume Preservation for a Mass-Spring System. *Computer Graphics and Applications, IEEE*, **26**(5), 83–91.

Hsu, William M., Hughes, John F., & Kaufman, Henry. 1992. Direct Manipulation of Free-form Deformations. *Pages 177–184 of: Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '92. New York, NY, USA: ACM.

Huang, Haoda, Zhao, Ling, Yin, KangKang, Qi, Yue, Yu, Yizhou, & Tong, Xin. 2011. Controllable Hand Deformation from Sparse Examples with Rich Details. *Pages 73–82 of: Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '11. New York, NY, USA: ACM.

Huang, Jin, Shi, Xiaohan, Liu, Xinguo, Zhou, Kun, Wei, Li-Yi, Teng, Shang-Hua, Bao, Hujun, Guo, Baining, & Shum, Heung-Yeung. 2006. Subspace Gradient Domain Mesh Deformation. *ACM Trans. Graph.*, **25**(3), 1126–1134.

Jacka, David, Reid, Ashley, Merry, Bruce, & Gain, James. 2007. A Comparison of Linear Skinning Techniques for Character Animation. *Pages 177–186 of: Proceedings of the 5th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*. AFRIGRAPH '07. New York, NY, USA: ACM.

Jacobson, Alec, & Sorkine, Olga. 2011. Stretchable and Twistable Bones for Skeletal Shape Deformation. *Pages 165:1–165:8 of: Proceedings of the 2011 SIGGRAPH Asia Conference*. SA '11. New York, NY, USA: ACM.

Jacobson, Alec, Baran, Ilya, Popović, Jovan, & Sorkine, Olga. 2011. Bounded Biharmonic Weights for Real-time Deformation. *ACM Trans. Graph.*, **30**(4), 78:1–78:8.

Jacobson, Alec, Deng, Zhigang, Kavan, Ladislav, & Lewis, JP. 2014a. Skinning: Real-time Shape Deformation. *In: ACM SIGGRAPH 2014 Courses*.

Jacobson, Alec, Panozzo, Daniele, Glauser, Oliver, Pradalier, Cédric, Hilliges, Otmar, & Sorkine-Hornung, Olga. 2014b. Tangible and Modular Input Device for Character Articulation. *ACM Trans. Graph.*, **33**(4), 82:1–82:12.

Jain, Sumit, & Liu, C. Karen. 2011. Controlling Physics-Based Characters Using Soft Contacts. *ACM Trans. Graph. (SIGGRAPH Asia)*, **30**(Dec.), 163:1–163:10.

Jakobsen, Thomas. 2001. 2001) Advanced character physics. *In: In Proceedings of the Game Developers Conference 2001. CMP media*.

James, Doug L., & Pai, Dinesh K. 1999. ArtDefo: Accurate Real Time Deformable Objects. *Pages 65–72 of: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '99. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

James, Doug L., & Pai, Dinesh K. 2004. BD-tree: Output-sensitive Collision Detection for Reduced Deformable Models. *Pages 393–398 of: ACM SIGGRAPH 2004 Papers*. SIGGRAPH '04. New York, NY, USA: ACM.

James, Doug L., & Twigg, Christopher D. 2005. Skinning Mesh Animations. *Pages 399–407 of: ACM SIGGRAPH 2005 Papers*. SIGGRAPH '05. New York, NY, USA: ACM.

Jiménez, P., Thomas, F., & Torras, C. 2000. 3D Collision Detection: A Survey. *Computers and Graphics*, **25**, 269–285.

Joshi, Pushkar, Meyer, Mark, DeRose, Tony, Green, Brian, & Sanocki, Tom. 2007. Harmonic Coordinates for Character Articulation. *ACM Trans. Graph.*, **26**(3).

Ju, Tao, Schaefer, Scott, & Warren, Joe. 2005. Mean Value Coordinates for Closed Triangular Meshes. *ACM Trans. Graph.*, **24**(3), 561–566.

Ju, Tao, Zhou, Qian-Yi, van de Panne, Michiel, Cohen-Or, Daniel, & Neumann, Ulrich. 2008. Reusable Skinning Templates Using Cage-based Deformations. *ACM Trans. Graph.*, **27**(5), 122:1–122:10.

Jund, Thomas, Cazier, David, & Dufourd, Jean-Francois. 2009. Particle-based forecast mechanism for continuous collision detection in deformable environments. *Pages*

147–158 of: SPM '09: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling. New York, NY, USA: ACM.

Kavan, Ladislav, & Sorkine, Olga. 2012. Elasticity-inspired Deformers for Character Articulation. *ACM Trans. Graph.*, **31**(6), 196:1–196:8.

Kavan, Ladislav, & Zara, Jiri. 2003. Real Time Skin Deformation with Bones Blending. *In: The 11-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2003, WSCG 2003, in co-operation with EUROGRAPHICS and IFIP working group 5.10 on Computer Graphics and Virtual Worlds, University of West Bohemia, Campus Bory, Plzen-Bory, Czech Republic, February 3-7, 2003*.

Kavan, Ladislav, & Zara, Jiri. 2005. Fast Collision Detection for Skeletally Deformable Models. *Computer Graphics Forum*, **24**(3), 363–372.

Kavan, Ladislav, & Zára, Jiří. 2005. Spherical Blend Skinning: A Real-time Deformation of Articulated Models. *Pages 9–16 of: Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*. I3D '05. New York, NY, USA: ACM.

Kavan, Ladislav, O'Sullivan, Carol, & Žára, Jiří. 2006. Efficient Collision Detection for Spherical Blend Skinning. *Pages 147–156 of: Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia*. GRAPHITE '06. New York, NY, USA: ACM.

Kavan, Ladislav, Collins, Steven, Zára, Jiří, & O'Sullivan, Carol. 2007. Skinning with Dual Quaternions. *Pages 39–46 of: Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*. I3D '07. New York, NY, USA: ACM.

Kavan, Ladislav, Collins, Steven, Zára, Jiří, & O'Sullivan, Carol. 2008. Geometric Skinning with Approximate Dual Quaternion Blending. *ACM Trans. Graph.*, **27**(4), 105:1–105:23.

Kavan, Ladislav, Collins, Steven, & O'Sullivan, Carol. 2009. Automatic Linearization of Nonlinear Skinning. *Pages 49–56 of: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*. I3D '09. New York, NY, USA: ACM.

Kenwright, Ben. 2012. A Beginners Guide to Dual-Quaternions: What They Are, How They Work, and How to Use Them for 3D. *In: Character Hierarchies,Äù, The 20th International Conference on Computer Graphics, Visualization and Computer Vision, WSCG 2012 Communication Proceedings,pp.1-13*.

Kim, Junggon, & Pollard, Nancy S. 2011. Fast Simulation of Skeleton-driven Deformable Body Characters. *ACM Trans. Graph.*, **30**(5), 121:1–121:19.

Kim, Theodore, & James, Doug L. 2011. Physics-based Character Skinning Using Multi-domain Subspace Deformations. *Pages 63–72 of: Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '11. New York, NY, USA: ACM.

Kim, YoungBeom, & Han, JungHyun. 2014. Bulging-free dual quaternion skinning. *Journal of Visualization and Computer Animation*, **25**(3-4), 323–331.

Klosowski, James T., Held, Martin, Mitchell, Joseph S. B., Sowizral, Henry, & Zikan, Karel. 1998. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, **4**(1), 21–36.

Komatsu, Koji. 1988. Human skin model capable of natural shape variation. *The Visual Computer*, **3**(5), 265–271.

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

Kurihara, Tsuneya, & Miyata, Natsuki. 2004. Modeling Deformable Human Hands from Medical Images. *Pages 355–363 of: Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '04. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association.

Larboulette, Caroline, Cani, Marie-Paule, & Arnaldi, Bruno. 2005. Dynamic skinning: adding real-time dynamic effects to an existing character animation. *Pages 87–93 of: SCCG*.

Larsson, Thomas, & Akenine-Möller, Tomas. 2001. Collision Detection for Continuously Deforming Bodies. *Pages 325–333 of: Proceedings of Eurographics 2001*.

Larsson, Thomas, & Akenine-Möller, Tomas. 2003. Efficient collision detection for models deformed by morphing. *The Visual Computer*, **19**(2-3), 164–174.

Larsson, Thomas, & Akenine-Möller, Tomas. 2006. A Dynamic Bounding Volume Hierarchy for Generalized Collision Detection. *Comput. Graph.*, **30**(3), 450–459.

Lasseter, John. 1987. Principles of Traditional Animation Applied to 3D Computer Animation. *SIGGRAPH Comput. Graph.*, **21**(4), 35–44.

Le, Binh Huy, & Deng, Zhigang. 2012. Smooth Skinning Decomposition with Rigid Bones. *ACM Trans. Graph.*, **31**(6), 199:1–199:10.

Le, Binh Huy, & Deng, Zhigang. 2014. Robust and Accurate Skeletal Rigging from Mesh Sequences. *ACM Trans. Graph.*, **33**(4), 84:1–84:10.

Lee, Gene S., & Hanner, Frank. 2009. Practical Experiences with Pose Space Deformation. *Pages 43:1–43:1 of: ACM SIGGRAPH ASIA 2009 Sketches*. SIGGRAPH ASIA '09. New York, NY, USA: ACM.

Lee, Gene S., Lin, Andy, Schiller, Matt, Peters, Scott, McLaughlin, Mark, & Hanner, Frank. 2013. Enhanced Dual Quaternion Skinning for Production Use. *Pages 9:1–9:1 of: ACM SIGGRAPH 2013 Talks*. SIGGRAPH '13. New York, NY, USA: ACM.

Lee, Sung-Hee, Sifakis, Eftychios, & Terzopoulos, Demetri. 2009. Comprehensive Biomechanical Modeling and Simulation of the Upper Body. *ACM Trans. Graph.*, **28**(4), 99:1–99:17.

Lee, Yuencheng, Terzopoulos, Demetri, & Waters, Keith. 1995. Realistic Modeling for Facial Animation. *Pages 55–62 of: Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '95. New York, NY, USA: ACM.

Lewis, J. P., Cordner, Matt, & Fong, Nickson. 2000. Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-driven Deformation. *Pages 165–172 of: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

Lin, Ming C., & Manocha, Dinesh. 2003. *Collision and Proximity Queries*. CRC Press LLC, Boca Raton, FL, ch. 35.

Lipman, Yaron, Kopf, Johannes, Cohen-Or, Daniel, & Levin, David. 2007a. GPU-assisted Positive Mean Value Coordinates for Mesh Deformations. *Pages 117–123 of: Proceedings of the Fifth Eurographics Symposium on Geometry Processing*. SGP '07. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association.

Lipman, Yaron, Cohen-Or, Daniel, Gal, Ran, & Levin, David. 2007b. Volume and Shape Preservation via Moving Frame Manipulation. *ACM Trans. Graph.*, **26**(1).

Liu, Libin, Yin, KangKang, Wang, Bin, & Guo, Baining. 2013a. Simulation and Control of Skeleton-driven Soft Body Characters. *ACM Trans. Graph.*, **32**(6), 215:1–215:8.

Liu, Tiantian, Bargteil, Adam W., O'Brien, James F., & Kavan, Ladislav. 2013b. Fast Simulation of Mass-spring Systems. *ACM Trans. Graph.*, **32**(6), 214:1–214:7.

Luque, Rodrigo G., Comba, João L. D., & Freitas, Carla M. D. S. 2005. Broad-phase Collision Detection Using Semi-adjusting BSP-trees. *Pages 179–186 of: Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*. I3D '05. New York, NY, USA: ACM.

Maciel, Anderson, Boulic, Ronan, & Thalmann, Daniel. 2007. Efficient Collision Detection Within Deforming Spherical Sliding Contact. *IEEE Transactions on Visualization and Computer Graphics*, **13**(3), 518–529.

Madera, F. A., Day, A. M., & Laycock, S. D. 2006. Collision Detection for Deformable Objects using Octrees. *In:* Lever, Louise M., & McDerby, Mary (eds), *Theory and Practice of Computer Graphics 2006*. The Eurographics Association.

Magnenat-Thalmann, N., Laperrière, R., & Thalmann, D. 1988. Joint-dependent Local Deformations for Hand Animation and Object Grasping. *Pages 26–33 of: Proceedings on Graphics Interface '88*. Toronto, Ont., Canada, Canada: Canadian Information Processing Society.

Magnenat-Thalmann, Nadia, Seo, Hyewon, & Cordier, Frederic. 2004. Automatic Modeling of Virtual Humans and Body Clothing. *J. Comput. Sci. Technol.*, **19**(5), 575–584.

McAdams, Aleka, Zhu, Yongning, Selle, Andrew, Empey, Mark, Tamstorf, Rasmus, Teran, Joseph, & Sifakis, Eftychios. 2011. Efficient Elasticity for Character Skinning with Contact and Collisions. *ACM Trans. Graph.*, **30**(4), 37:1–37:12.

McLaughlin, Tim, Cutler, Larry, & Coleman, David. 2011. Character Rigging, Deformations, and Simulations in Film and Game Production. *Pages 5:1–5:18 of: ACM SIGGRAPH 2011 Courses*. SIGGRAPH '11. New York, NY, USA: ACM.

Merry, Bruce, Marais, Patrick, & Gain, James. 2006. Animation Space: A Truly Linear Framework for Character Animation. *ACM Trans. Graph.*, **25**(4), 1400–1423.

Mezger, J., Kimmerle, S., & Etzmuß, O. 2003. Hierarchical Techniques in Collision Detection for Cloth Animation. *Journal of WSCG*, **11**, 322–329.

Milliron, Tim, Jensen, Robert J., Barzel, Ronen, & Finkelstein, Adam. 2002. A Framework for Geometric Warps and Deformations. *ACM Trans. Graph.*, **21**(1), 20–51.

Min, Kyung-Ha, Baek, Seung-Min, Lee, Gun A, Choi, Haeock, & Park, Chan-Mo. 2000. Anatomically-based modeling and animation of human upper limbs. *In: Proceedings of International Conference on Human Modeling and Animation*.

Mohr, Alex, & Gleicher, Michael. 2003. Building Efficient, Accurate Character Skins from Examples. *Pages 562–568 of: ACM SIGGRAPH 2003 Papers*. SIGGRAPH '03. New York, NY, USA: ACM.

Moore, P., & Molloy, D. 2007 (Sept). A Survey of Computer-Based Deformable Models. *Pages 55–66 of: Machine Vision and Image Processing Conference, 2007. IMVIP 2007. International*.

Müller, Matthias, & Gross, Markus. 2004. Interactive Virtual Materials. *Pages 239–246 of: Proceedings of Graphics Interface 2004*. GI '04. School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada: Canadian Human-Computer Communications Society.

Müller, Matthias, Dorsey, Julie, McMillan, Leonard, Jagnow, Robert, & Cutler, Barbara. 2002. Stable Real-time Deformations. *Pages 49–54 of: Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. New York, NY, USA: ACM.

Müller, Matthias, Heidelberger, Bruno, Hennix, Marcus, & Ratcliff, John. 2007. Position Based Dynamics. *J. Vis. Comun. Image Represent.*, **18**(2), 109–118.

Nealen, Andrew, Mueller, Matthias, Keiser, Richard, Boxerman, Eddy, & Carlson, Mark. 2006. Physically Based Deformable Models in Computer Graphics. *Computer Graphics Forum*, **25**(4), 809–836.

Nieto, JesusR., & Susin, Antonio. 2013. Cage Based Deformations: A Survey. *Pages 75–99 of: Deformation Models*. Lecture Notes in Computational Vision and Biomechanics, vol. 7. Springer Netherlands.

Otaduy, Miguel A., Chassot, Olivier, Steinemann, Denis, & Gross, Markus. 2007. Balanced Hierarchies for Collision Detection between Fracturing Objects. *Proc. of the IEEE Virtual Reality Conference*, 83–90.

Park, Sang Il, & Hodgins, Jessica K. 2006. Capturing and Animating Skin Deformation in Human Motion. *Pages 881–889 of: ACM SIGGRAPH 2006 Papers*. SIGGRAPH '06. New York, NY, USA: ACM.

Park, Sang Il, & Hodgins, Jessica K. 2008. Data-driven Modeling of Skin and Muscle Deformation. *Pages 96:1–96:6 of: ACM SIGGRAPH 2008 Papers*. SIGGRAPH '08. New York, NY, USA: ACM.

Pentland, Alex, & Williams, John. 1989. Good vibrations: model dynamics for graphics and animation. *Pages 215–222 of: SIGGRAPH*.

Pons-Moll, Gerard, Romero, Javier, Mahmood, Naureen, & Black, Michael J. 2015. Dyna: A Model of Dynamic Human Shape in Motion. *ACM Trans. Graph.*, **34**(4), 120:1–120:14.

Popović, Jovan, Seitz, Steven M., & Erdmann, Michael. 2003. Motion Sketching for Control of Rigid-body Simulations. *ACM Trans. Graph.*, **22**(4), 1034–1054.

Provot, Xavier. 1997. Collision and self-collision handling in cloth model dedicated to design garments. *Pages 177–189 of:* Thalmann, Daniel, & van de Panne, Michiel (eds), *Computer Animation and Simulation ‚Äô97*. Eurographics. Springer Vienna.

Reddy, Junuthula Narasimha. 93. *An introduction to the finite element method*. McGraw-Hill series in mechanical engineering. New York, NY: McGraw-Hill Higher Education.

Rhee, Taehyun, Lewis, John P., & Neumann, Ulrich. 2006. Real-Time Weighted Pose-Space Deformation on the GPU. *Comput. Graph. Forum*, **25**(3), 439–448.

Rohmer, Damien, Hahmann, Stefanie, & Cani, Marie-Paule. 2008. Local Volume Preservation for Skinned Characters. *Comput. Graph. Forum*, **27**(7), 1919–1927.

Rohmer, Damien, Hahmann, Stefanie, & Cani, Marie-Paule. 2009. Exact Volume Preserving Skinning with Shape Control. *Pages 83–92 of: Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '09. New York, NY, USA: ACM.

Rumman, Nadine Abu, & Fratarcangeli, Marco. 2016. State of the Art in Skinning Techniques for Articulated Deformable Characters. *Pages 200–212 of: Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 1: GRAPP,*.

Rumman, Nadine Abu, Schaerf, Marco, & Bechmann, Dominique. 2015. Collision Detection for Articulated Deformable Characters. *Pages 215–220 of: Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*. MIG '15. New York, NY, USA: ACM.

Savoye, Yann, & Franco, Jean-Sébastien. 2010. CageIK: Dual-Laplacian Cage-based Inverse Kinematics. *Pages 280–289 of: Proceedings of the 6th International Conference on Articulated Motion and Deformable Objects*. AMDO'10. Berlin, Heidelberg: Springer-Verlag.

Schumacher, Christian, Thomaszewski, Bernhard, Coros, Stelian, Martin, Sebastian, Sumner, Robert, & Gross, Markus. 2012. Efficient Simulation of Example-based Materials. *Pages 1–8 of: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '12. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association.

Schvartzman, Sara C., Gascón, Jorge, & Otaduy, Miguel A. 2009. Bounded Normal Trees for Reduced Deformations of Triangulated Surfaces. *Pages 75–82 of: Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '09. New York, NY, USA: ACM.

Schvartzman, Sara C., Pérez, Álvaro G., & Otaduy, Miguel A. 2010. Star-contours for Efficient Hierarchical Self-collision Detection. *ACM Trans. Graph.*, **29**(4), 80:1–80:8.

Sederberg, Thomas W., & Parry, Scott R. 1986. Free-form Deformation of Solid Geometric Models. *Pages 151–160 of: Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '86. New York, NY, USA: ACM.

Shabana, Ahmed A. 1989. *Dynamics of multibody systems*. New York, New York: Cambridge University Press.

Sheth, Rahul, Lu, Wenlong, Yu, Yue, & Fedkiw, Ronald. 2015. Fully Momentum-conserving Reduced Deformable Bodies with Collision, Contact, Articulation, and Skinning. *Pages 45–54 of: Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA '15. New York, NY, USA: ACM.

Shi, Xiaohan, Zhou, Kun, Tong, Yiying, Desbrun, Mathieu, Bao, Hujun, & Guo, Baining. 2008. Example-based Dynamic Skinning in Real Time. *ACM Trans. Graph.*, **27**(3), 29:1–29:8.

Shinar, Tamar, Schroeder, Craig, & Fedkiw, Ronald. 2008. Two-way Coupling of Rigid and Deformable Bodies. *Pages 95–103 of: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '08. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association.

Si, Hang. 2015. TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Trans. Math. Softw.*, **41**(2), 11:1–11:36.

Sifakis, Eftychios, & Barbic, Jernej. 2012. FEM Simulation of 3D Deformable Solids: A Practitioner's Guide to Theory, Discretization and Model Reduction. *Pages 20:1–20:50 of: ACM SIGGRAPH 2012 Courses*. SIGGRAPH '12. New York, NY, USA: ACM.

Singh, Karan, & Kokkevis, Evangelos. 2000. Skinning Characters using Surface-Oriented Free-Form Deformations. *Pages 35–42 of: In Graphics Interface 2000*.

Sloan, Peter-Pike J., Rose, III, Charles F., & Cohen, Michael F. 2001. Shape by Example. *Pages 135–143 of: Proceedings of the 2001 Symposium on Interactive 3D Graphics*. I3D '01. New York, NY, USA: ACM.

Sorkine, O., Cohen-Or, D., Lipman, Y., Alexa, M., Rössl, C., & Seidel, H.-P. 2004. Laplacian Surface Editing. *Pages 175–184 of: Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*. SGP '04. New York, NY, USA: ACM.

Spillmann, J., Becker, M., & Teschner, M. 2007. Efficient Updates of Bounding Sphere Hierarchies for Geometrically Deformable Models. *Journal of Visual Communication and Image Representation*, **18**(2), 101–108.

Stam, J. 2009 (Aug). Nucleus: Towards a unified dynamics solver for computer graphics. *Pages 1–11 of: Computer-Aided Design and Computer Graphics, 2009. CAD/Graphics '09. 11th IEEE International Conference on*.

Stewart, D. E., & Trinkle, J. C. 1996. An Implicit Time-Stepping Scheme For Rigid Body Dynamics With Inelastic Collisions And Coulomb Friction. *International Journal for Numerical Methods in Engineering*, **39**(15), 2673–2691.

Sud, Avneesh, Govindaraju, Naga, Gayle, Russell, Kabul, Ilknur, & Manocha, Dinesh. 2006. Fast proximity computation among deformable models using discrete Voronoi diagrams. *ACM Trans. Graph. (Proc ACM SIGGRAPH*, **25**, 1144–1153.

Sueda, Shinjiro, Kaufman, Andrew, & Pai, Dinesh K. 2008. Musculotendon Simulation for Hand Animation. *ACM Trans. Graph.*, **27**(3), 83:1–83:8.

Teller, Seth J., & Sequin, Carlo H. 1991. Visibility Preprocessing For Interactive Walkthroughs. *Pages 61–69 of: IN: COMPUTER GRAPHICS (SIGGRAPH 91 PROCEEDINGS*.

Teng, Yun, Otaduy, Miguel A., & Kim, Theodore. 2014. Simulating Articulated Subspace Self-contact. *ACM Trans. Graph.*, **33**(4), 106:1–106:9.

Terzopoulos, Demetri, Platt, John, Barr, Alan, & Fleischer, Kurt. 1987. Elastically Deformable Models. *SIGGRAPH Comput. Graph.*, **21**(4), 205–214.

Teschner, M., Kimmerle, S., Heidelberger, B., Zachmann, G., Raghupathi, L., Fuhrmann, A., Cani, M.-P., Faure, F., Magnenat-Thalmann, N., Strasser, W., & Volino, P. 2005. Collision Detection for Deformable Objects. *Computer Graphics Forum*, **24**(1), 61–81.

Teschner, Matthias, Heidelberger, Bruno, Müller, Matthias, Pomerantes, Danat, & Gross, Markus H. 2003. Optimized Spatial Hashing for Collision Detection of Deformable Objects. *Pages 47–54 of: Proceedings of the Vision, Modeling, and Visualization Conference 2003 (VMV 2003), München, Germany, November 19-21, 2003*.

Thomas, Frank, & Johnston, Ollie. 1981. *The illusion of life : Disney animation*. New York: Disney Editions.

Turk, Greg. 1989. *Interactive collision detection for molecular graphics*. Tech. rept.

Turner, Russell, & Gobbetti, Enrico. 1998. Interactive Construction and Animation of Layered Elastically Deformable Characters. *Computer Graphics Forum*, **17**(2), 135–152.

Turner, Russell, & Thalmann, Daniel. 1993. The Elastic Surface Layer Model for Animated Character Construction. *Pages 399–412 of: Proceedings of Computer Graphics International '93*. SpringerVerlag.

Vaillant, Rodolphe, Barthe, Loïc, Guennebaud, Gaël, Cani, Marie-Paule, Rohmer, Damien, Wyvill, Brian, Gourmel, Olivier, & Paulin, Mathias. 2013. Implicit Skinning: Real-time Skin Deformation with Contact Modeling. *ACM Trans. Graph.*, **32**(4), 125:1–125:12.

Vaillant, Rodolphe, Guennebaud, Gäel, Barthe, Loïc, Wyvill, Brian, & Cani, Marie-Paule. 2014. Robust Iso-surface Tracking for Interactive Character Skinning. *ACM Trans. Graph.*, **33**(6), 189:1–189:11.

van den Bergen, Gino. 1998. Efficient Collision Detection of Complex Deformable Models Using AABB Trees. *J. Graph. Tools*, **2**(4), 1–13.

Verlet, Loup. 1968. Computer "Experiments" on Classical Fluids. II. Equilibrium Correlation Functions. *Phys. Rev.*, **165**(Jan), 201–214.

Volino, Pascal, & Thalmann, Nadia Magnenat. 1994. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum*, **13**(3), 155–166.

von Funck, Wolfram, Theisel, Holger, & Seidel, Hans-Peter. 2006. Vector Field Based Shape Deformations. *ACM Trans. Graph.*, **25**(3), 1118–1125.

von Funck, Wolfram, Theisel, Holger, & Seidel, Hans-Peter. 2008. Volume-preserving Mesh Skinning. *Pages 409–414 of: Proceedings of the Vision, Modeling, and Visualization Conference 2008, VMV 2008, Konstanz, Germany, October 8-10, 2008*.

Walter, Marcelo, & Fournier, Alain. 1997. Growing and Animating Polygonal Models of Animals. *Computer Graphics Forum*, **16**(3), 151–158.

Wang, Robert Y., Pulli, Kari, & Popović, Jovan. 2007. Real-time Enveloping with Rotational Regression. *ACM Trans. Graph.*, **26**(3).

Wang, Xiaohuan Corina, & Phillips, Cary. 2002. Multi-weight Enveloping: Least-squares Approximation Techniques for Skin Animation. *Pages 129–138 of: Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '02. New York, NY, USA: ACM.

Wareham, Rich, & Lasenby, Joan. 2008. Bone Glow: An Improved Method for the Assignment of Weights for Mesh Deformation. *Pages 63–71 of: AMDO*.

Weber, Ofir, Sorkine, Olga, Lipman, Yaron, & Gotsman, Craig. 2007. Context-Aware Skeletal Shape Deformation. *Comput. Graph. Forum*, **26**(3), 265–274.

Wilhelms, Jane. 1994. *Modeling Animals with Bones, Muscles, and Skin*. Tech. rept. University of California.

Yang, Xiaosong, Somasekharan, Arun, & Zhang, Jian J. 2006. Curve skeleton skinning for human and creature characters. *Computer Animation and Virtual Worlds*, **17**(3-4), 281–292.

Ye, Yuting, & Liu, C. Karen. 2012. Synthesis of Detailed Hand Manipulations Using Contact Sampling. *ACM Trans. Graph.*, **31**(4), 41:1–41:10.

Zachmann, Gabriel. 1995. The BoxTree: Exact and Fast Collision Detection of Arbitrary Polyhedra. *Pages 104–112 of: In SIVE Workshop*.

Zachmann, Gabriel, & Langetepe, E. 2002. Geometric Data Structures for Computer Graphics. *In: Proceedings of Eurographics 2002 Tutorials*. The Eurographics Association. Tutorial.

Zachmann, Gabriel, & Weller, René. 2006 (14–17 June). Kinetic Bounding Volume Hierarchies for Deformable Objects. *In: ACM International Conference on Virtual Reality Continuum and Its Applications (VRCIA)*.

Zhang, Dongliang, & Yuen, M.M.F. 2000. Collision detection for clothed human animation. *Pages 328–337 of: Computer Graphics and Applications, 2000. Proceedings. The Eighth Pacific Conference on*.

Zhou, Kun, Huang, Jin, Snyder, John, Liu, Xinguo, Bao, Hujun, Guo, Baining, & Shum, Heung-Yeung. 2005. Large Mesh Deformation Using the Volumetric Graph Laplacian. *Pages 496–503 of: ACM SIGGRAPH 2005 Papers*. SIGGRAPH '05. New York, NY, USA: ACM.