Diss. ETH No. 22964

Optimization Methods for Character Animation using Rig Spaces

A thesis submitted to attain the degree of
Doctor of Sciences of ETH Zurich
(Dr. sc. ETH Zurich)

presented by
Fabian Hahn
MSc ETH in Informatik, ETH Zurich

born on
23.08.1988

citizen of
Niederhasli (ZH), Switzerland
and the **United States of America**

accepted on the recommendation of
Prof. Dr. Robert W. Sumner, examiner
Prof. Dr. Markus Gross, co-examiner
Dr. Anthony D. DeRose, co-examiner

2015

Abstract

Despite significant advances in computer graphics research targeting the field of virtual 3D character animations, the workflows and tools of professional animation studios creating content for feature films and games have barely evolved. As a result, animators often have to perform tedious and time-consuming tasks during their work cycle that are mostly unrelated to the creative process of designing a virtual character and its style of motion. This thesis attempts to bridge this gap by proposing the use of character rigs as a central component for computational methods to be built around, thus bringing state-of-the-art optimization techniques into an environment familiar to artists. Rather than discarding a rig as a mere mechanism artists build and use to facilitate their work, character rigs are treated as an artist-designed subspace of deformations a character is allowed to undergo.

To demonstrate the potential of this rig-focused optimization approach, three topics in the character animation field are investigated and new methods utilizing the rig deformation space are presented. In the field of rig-space simulation, a series of proposed optimizations allow solvers to compute secondary motion of a character's body with improved performance of up to two orders of magnitude compared to previous approaches, while achieving the same level of quality. Further, this work explores the use of rig-space optimization for the application of character posing by introducing the concept of sketch abstractions, which allows direct matching of a user drawn sketch with a rigged 3D character in 2D. As a third contribution, the thesis provides a novel adaptive subspace simulation algorithm that enables the computation of wrinkles and folds of tight-fitting clothing to be performed in a reduced space, achieving significant runtime speedups over the corresponding full-space simulations. The applicability of these presented methods to real-world data is showcased by applying them to complex production-quality character rigs.

Zusammenfassung

Trotz sigifikanten Fortschritten in der Computergraphikforschung im Gebiet der Animation von 3D-Charaktern haben sich die Arbeitsabläufe und Werkzeuge von professionellen Animationsstudios, welche Inhalte für Trickfilme und Videospiele produzieren, kaum weiterentwickelt. Dies hat zur Folge, dass Animationskünstler während ihres Arbeitsablaufs oft langwierige und zeitaufwändige Schritte durchführen müssen, welche grösstenteils keinen Zusammenhang mit dem kreativen Erstellungsprozess von virtuellen Charakteren und ihres Bewegungsstils haben. Diese Dissertation versucht, diese Defizite zu überwinden, indem sie Characterrigs as zentrale Komponente vorschlägt, auf deren Basis rechnergestützte Methoden aufgebaut werden sollten. Auf diese Weise können Künstlern moderne Optimierungsmethoden, welche dem aktuellen Stand der Technik entsprechen, in einer ihnen familiären Art näher gebracht werden. Anstatt Rigs als reine Hilfestellungen für Künstler zu betrachten, welche ihnen ihre Arbeit erleichtern, werden Charakterrigs in dieser Arbeit als von Künstlern vorgegebene Unterräume betrachtet, welche alle möglichen Deformationen eines Charakters umfassen.

Um das Potential dieses rigfokussierten Optimierungsansatzes aufzuzeigen, werden drei verschiedene Themenbereiche innerhalb der Charakteranimation untersucht und neue Methoden präsentiert, welche vom Deformationsraum eines Rigs gebrauch machen. Für Simulationen im Rigraum ermöglichen eine Reihe von Verbesserungen, dass Algorithmen die Bewegungen zweiten Grades eines Charakterkörpers mit einer verbessertern Laufzeit von bis zu zwei Grössenordnungen verglichen mit bestehenden Methoden simulieren können. Eine solche verschnellerte Simulation ist möglich, ohne dass sich dabei die Qualität der Resultate verschlechtert. Des Weiteren untersucht diese Arbeit die Verwendung von Optimierung im Rigraum im Rahmen der Charakterposierung, indem sie das Konzept von Skizzenabstraktionen einführt. Mit diesen können vom Benutzer gezeichnete Skizzen direkt in 2D auf das Rig eines 3D-Charakters abgestimmt werden. In einem dritten Beitrag stellt diese Dissertation einen neuartigen adaptiven Simulationsalgorithmus vor, welcher die Berechnung von Knitter- und Falteneffekten von eng anliegender Kleidung in Unterräumen ermöglicht. Dies hat gegenüber einer Simulation im vollen Raum eine signifikante Laufzeitverbesserung zur Folge.

Die Praxistauglichkeit der vorgestellten Methoden wird anhand der Anwendung auf komplexe Charaketerrigs mit Produktionsqualität aufgezeigt.

Acknowledgements

When I decided to study Computer Science at ETH, I did so because I thought I would learn how to program video games, and even though I could not have been more wrong, I never regretted the choice. When I later decided to pursue a PhD in Computer Science, I did so because I thought it would be like my Master thesis on a larger scale, and even though I proved to be completely wrong again, I am very happy to say now that I do not regret this choice either. The last three years have been a wild ride that probably taught me more than all the years before that, and I have a couple of people to thank for that.

First and foremost, I would like to thank my two PhD advisors who made it possible for me to write a dissertation in such an exciting field: Prof. Bob Sumner for accepting me as his first PhD student and advising me during the second half of my PhD, and Prof. Markus Gross for initially offering me the position and advising me during the first half. I am proud of having been a part of their "graphics familiy" in Zurich at the Computer Graphics Lab and at Disney Research Zurich.

Second, I would like to thank all my collaborators that I had the great pleasure of working with on all the projects that I am presenting in this thesis. I am deeply grateful to Stelian Coros and Bernhard Thomaszewski for showing me what it means to work with the very best in the field during all the long discussions and research insights we shared. I would also like to thank Forrester Cole, Mark Meyer and Tony DeRose from Pixar Research for collaborating with us on the *Subspace Clothing Simulation* project, and Tony further for also serving on my PhD committee. Many thanks to Frederik Mutzel and Maurizio Nitti for their tremendous contributions to our *Sketch Abstractions* work, without which that project simply would not exist.

Third, I would like to thank all the wonderful people that supported me during the years of my PhD. Thanks to Antoine Milliez and Pascal Bérard for being the best office mates and friends I could have ever hoped for. I am deeply grateful to Alessia Marra for all the help in creating last-minute paper results. It was a great pleasure to supervise Sabina Schellenberg, who was brave enough to do her Master thesis with us on a topic as exotic as adaptive subspace simulation. Special thanks go to all current and past members of CGL, IGL and DRZ for creating such a fantastic working and social environment.

Last but not least, I would like to thank my family and all my friends for always patiently waiting for me to return from my paper deadline related absences from real life. I have the deepest gratitude for my partner Katharina Tschanen who stood by my side even during the most difficult of times. Kathi, I could not have done this without your continuous encouragement and love.

Contents

List of Figures

List of Algorithms

List of Tables

CHAPTER 1

Introduction

Animations of three-dimensional digital characters have become ubiquitous in recent years and have reached target audiences far beyond viewers of animated movies and cartoons. Nowadays, the tools and techniques for creating character animation also see wide use in the production of live-action movies, advertisements, and computer games. Furthermore, the animation workflow also transfers to related fields like robotics and locomotion, visualization, or the design of interactive websites and mobile applications. Despite this intensive use of 3D character animation, the tools and the overall workflow to create these animations have remained largely unchanged or only evolved slowly. As a result, it remains common practice to employ processes that burden the artists with a significant amount of tedious manual labor in order to create high-quality results.

The core of the animation pipelines used by professional studios can be roughly broken down into three distinct stages, each of which are usually carried out by different specialized artists: In the *modeling* stage, a virtual character is sculpted as a static object in a neutral pose, and is stored as a set of vertices connected to a surface mesh by primitive faces such as triangles and quads. This process could now be repeated for each desired pose of an animation by either resculpting the character from scratch in that pose or by editing the vertex positions of the character's neutral pose, which would be reminiscent of traditional 2D animation where characters are fully redrawn in each frame. Instead, the neutral character mesh is passed on to the *rigging* stage, where artists augment the static surface with a series of geometric functions that transform whole parts of the character surface. These de-

formers feature freely adjustable input parameters to control the extent of the deformation, effectively providing a user with a set of high-level controls to more intuitively pose the character. The entirety of these functions and its parameters is simply called a rig, and is further passed on to the *keyframe animation* stage of the pipeline. In this third stage, the character is brought to life by letting the rig parameters follow trajectories over time that are interpolated from keyframes that artist set at certain key poses. The modeling, rigging, and keyframing stages all involve a significant deal of manual effort that involve several tedious yet crucial tasks to achieve a final animation corresponding exactly to what a director envisioned.

Meanwhile, the field of computer graphics has made many advances in surface modeling and deformation, physically-based simulation, and animation synthesis. It might sound somewhat surprising at first that only few of these state-of-the-art methods have been embraced by the animation industry. While there are a variety of different reasons for this slow adoption of research results into production and the everyday workflow of artists creating animation, one likely explanation could be that many of these methods propose workflows different from the ones commonly used. Since the established tools and processes have proven so effective and reliable, the success of these long-standing animation systems had the adverse effect of letting skepticism prevail over potential usefulness when considering a novel tool that breaks tradition with familiar approaches.

In this work, we attempt to overcome this gap between state-of-the-art computer graphics research and the toolkits of animators by purposely designing our methods from an animator's perspective. We do this by treating the character rig—being created by an artist, not a programmer or researcher—as the central concept that our framework needs to support and build upon. We will show that character rigs are not a mere mechanism used by artists to facilitate the creation of animations, but that they span a *deformation space* that can be exploited to develop novel simulation systems that fit directly into an artist's workflow. This thesis explores rig deformation spaces in three different context and proposes concrete applications in the fields of physical simulation of secondary motion, sketch-based character posing, and efficient subspace simulation of tight-fitting clothing. Our aim is to demonstrate the mathematical potential of rigs in computer graphics research, and to establish the rig deformation subspace as a valuable asset in the design of future animation systems. We believe that this concept has far-reaching consequences which could enable its application to topics beyond those treated in this work.

1.1 Overview

In this section, we will give an overview over three proposed applications that we will use to showcase the power of a character rig's deformation space. While the motivations for developing novel methods in these areas slightly differ, they all share the goal of providing novel artist-friendly animation tools.

1.1.1 Efficient Rig-Space Physics Simulation

Creating believable and compelling character motions is arguably the central challenge in animated movie productions. While manually posing a character for each animation keyframe allows artists to create very expressive animations, this process is tedious when it comes to creating secondary motion such as the bulging of muscles or jiggling of fat. Hahn and colleagues [2012] recently presented *rig-space physics*, a method to augment keyframed animations with automatically computed secondary motion. The basic idea of rig-space physics is to use physics-based simulation in rig space, the character's space of motion. As a key advantage over conventional physics-based simulation, the rig-space approach results in animation curves that can be easily edited by artists. But while rig-space physics can automatically generate secondary motion with high visual quality, it entails a significant computational burden that slows production and prohibits its use in interactive environments. In this work, we will present a method that offers a significant computational improvement over the work of Hahn and colleagues [2012], while maintaining all its benefits and the same level of quality.

1.1.2 Sketch Abstractions for Character Posing

In classic 2D animation, artists draw each pose of a character by hand using pencil and paper. This tangible connection is a powerful interface that gives artists direct control over a character's shape. In 3D animation, posing a character is a more involved endeavor, since it entails the coordinated movement of thousands of vertices. To make this process tractable, rigging artists carefully craft character rigs that define the space of meaningful deformations in terms of abstract rigging parameters. Animators determine a character's pose indirectly by choosing values for these parameters. In order to accommodate the full range of expressive deformation, a production character rig may employ hundreds or thousands of different rigging controls, varying in complexity from blend shapes to skeletal deformation to complex

procedural functions. Naturally, navigating this huge parameter space is a challenging task that can tax even the best animators. Our work attempts to bring the direct control offered by sketching into the 3D animation pipeline with a sketch-based posing system that utilizes customized character sketch abstractions.

1.1.3 Subspace Clothing Simulation

Clothing plays a central role in compelling animation by contributing to the style and personality of animated characters while evoking the impression of realism and complexity that comes from detailed folding patterns. On a practical level, a great deal of clothing used in animated films consists of close-fitting garments that move along with the body. For example, a sweater may not show noticeable dynamics under normal body motion, but it will exhibit noticeable buckling patterns at the arm bends and oblique torsional folds rising up from the waist. Frictional contacts and the overall nonlinear nature of cloth mean that these quasi-static folds depend not only on the current pose but also on the path taken in pose space to arrive at the present body configuration. This property gives clothing an infinite source of diversity and detail—and also shows that cloth simulation is an indispensable tool in feature animation, even for close-fitting garments.

Although simulation systems can compute the deformation of cloth at a remarkable level of realism and detail, they incur an extremely high computational cost. Subspace methods have proven very effective at reducing computational cost for other applications such as finite-element based solid simulation. These methods are most effective when deformations are small or predictable. In such cases, one can construct a low-dimensional linear subspace whose basis remains constant over time, thus delivering high computational efficiency. Unfortunately, due to hysteresis and its inherent nonlinear nature, cloth deformations are generally neither small nor predictable, which is seemingly at odds with subspace simulation and probably explains why so few attempts have been made so far in this direction. In this thesis, we will present a subspace simulation technique in the context of clothing simulation that attempts to overcome these challenges and that improves the performance of state-of-the-art cloth simulation codes significantly while still reproducing the rich deformations of a full-space solution.

1.2 Principal Contributions

The principal contributions of the work presented in this thesis to the field of computer graphics are as follows:

- A flexible optimization framework targeted at the simulation of physical objects and the matching of character poses while making use of character rig's deformation space. The key feature of our system is to minimize objective functions consisting of arbitrary energy terms both in full space and projected into arbitrary subspaces. This carefully designed abstraction layer allows a single solver to handle all of the numerical optimizations required for the applications presented in this work, in particular the simulation of an elastic solid in a rig's deformation space, the optimization of a character pose to match a user drawn sketch, and the reduced simulation of tight-fitting clothing worn by a virtual character. Over the course of this thesis, we will see how this core solver can be further optimized to obtain the maximum runtime performance when targeting these individual applications.
- A novel method targeting the rig-space simulation of physical secondary motion of a virtual character. Our method offers significant performance improvements over the state-of-the-art approach to the problem by Hahn and colleagues [2012], while achieving the same level of quality in simulation results. We achieve this by providing a linearized formulation of the rig-space dynamics, a physics-based volumetric skinning method, and a deferred Jacobian evaluation scheme. Taken together, these three components allow our method to achieve performance improvements of up to two orders of magnitude over the original rig-space physics method on production-quality rigs.
- The concept of sketch abstractions comprising an iconographic 2D representation of a virtual character from a particular viewpoint. Our method brings direct control to character posing by providing a sketch-based posing system that directly matches a user drawn sketch with the sketch abstraction. We achieve this by formulating a nonlinear iterative closest point energy that directly optimizes the distance between the input sketch and the sketch abstraction as deformed by the character rig. A custom regularizer is employed to address the underconstrained nature of the problem and resolve potential 3D depth ambiguities.

- An adaptive subspace simulation algorithm that enables the reduced simulation of tight-fitting clothing worn by a virtual character. We propose the use of the underlying character rig as a kinematic reference, which allows the construction of a linear subspace in an untransformed space where deformations do not exhibit rotations induced by the pose of the character. To enable fast runtime simulations with a very low number of subspace dimensions, we introduce an adaptive basis selection scheme that computes a suitable simulation space for the current pose on-the-fly. The basis vectors are precomputed and stored in a pose space database, which allows our method to both offer significant speedups over full-space simulations, as well as generalize to motions with novel poses that were not part of the training examples.

1.3 Thesis Outline

This thesis is organized as follows: After the current Chapter 1 giving an introduction to the thesis, we will discuss related work on deformable models, rigging, subspace simulation, sketch-based posing and cloth simulation methods in Chapter 2. In Chapter 3, we will lay the mathematical foundations for the thesis and describe a simulation framework that we will use throughout this work. We will start by discussing the simulation of physical objects, before we proceed to energy minimization techniques that are used to advance them in time. Further, we will describe the subspace simulation of physical objects in a reduce space, state the analytic formulas and derivatives of some important types of rigs, and will give an introduction to contact handling. Chapter 4 will address the problem of efficiently simulating secondary motion effects in rig space. After investigating the state-of-the-art solver in rig-space simulation, we will propose several new techniques to significantly improve the performance of rig-space physics simulation, while retaining the same level of quality. In Chapter 5, we will apply the concept of rig-space optimization to the problem of sketch-based posing. To this end, we introduce the novel concept of sketch abstractions, which provide a means to perform the matching optimization completely in 2D and bridges the gap between arbitrary 3D rigs and drawn 2D curves. The subsequent Chapter 6 will investigate the problem of performing reduced simulations of cloth, and introduce an algorithm that is able to simulate tight-fitting clothing using an adaptive subspace selection scheme. We will first introduce the data structure of a pose space database, which we rely on to select basis vectors from during the runtime of our adaptive subspace sim-

ulation technique, before stating and explaining the individual steps of our algorithm. Chapter 7 will conclude the thesis and discuss limitations, as well as potential avenues for future work. A description of the mathematical notation used throughout this work is provided in the Appendix.

1.4 Publications

In the context of this thesis, the following peer-reviewed publications have been accepted:

- F. HAHN, B. THOMASZEWSKI, S. COROS, R. SUMNER and M. GROSS. Efficient Simulation of Secondary Motion in Rig-Space. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 165-171, 2013.
  We present an efficient method for augmenting keyframed character animations with physically-simulated secondary motion. Our method achieves a performance improvement of one to two orders of magnitude over previous work without compromising on quality.
- F. HAHN, B. THOMASZEWSKI, S. COROS, R. SUMNER, F. COLE, M. MEYER, T. DEROSE, M. GROSS. Subspace Clothing Simulation Using Adaptive Bases. *ACM Transactions on Graphics*, vol. 33, no. 3, pp. 105:1-105:9, 2014.
  We present a new approach to clothing simulation using low-dimensional linear subspaces with adaptive bases.
- F. HAHN, F. MUTZEL, B. THOMASZEWSKI, S. COROS, M. NITTI, M. GROSS, R. SUMNER. Sketch Abstractions for Character Posing. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2015.
  We propose a sketch-based posing system for rigged 3D characters that allows artists to create custom sketch abstractions on top of a character's actual shape. A sketch abstraction is composed of rigged curves that form an iconographic 2D representation of the character from a particular viewpoint.

The contents of all these papers are included in this thesis, but the individual notations were harmonized to build upon the core simulation framework described in Chapter 3. Furthermore, additional technical details for each of the three applications are provided in this work, which should make it possible to implement the methods and reproduce the showcased results without consulting external sources.

CHAPTER 2

Related Work

This chapter lists some of the related work on rigging, physically-based simulation, and subspace optimization in the context of the three applications we are going to present in this thesis.

2.1 Efficient Rig-Space Physics Simulation

Designing and animating digital characters is a central problem in computer graphics. We refer the interested reader to the recent survey by McLaughlin and colleagues [2011] for an overview of the many challenges involved in this task. In this section, we focus on existing work related to the problem of creating secondary motions in the context of the *Efficient Rig-Space Simulation* method that we will present in Chapter 4.

2.1.1 Rigging

Before characters can be animated, they first have to be modeled and rigged. During modeling, artists define the surface mesh of a character, and the rigging stage requires them to specify how this surface mesh deforms as a function of a relatively small number of rig parameters. The map between the rig parameters and the deformation of the surface mesh can be defined using a variety of different techniques: linear blend or dual quaternion skinning [Magnenat-Thalmann et al., 1989; Kavan et al., 2008], wire

deformations [Singh and Fiume, 1998] or blend shapes [Lewis et al., 2000; Sloan et al., 2001], to name a few. Many of these techniques are complementary, and there is no single best solution for all applications. Furthermore, the choice of which of the techniques are eventually used also depends on other factors such as personal preference. In order to afford a maximum level of generality, we follow Hahn and colleagues [2012] and do not make any assumptions about the underlying rig. Consequently, we evaluate the rig and its derivatives through function calls to the modeling and animation software.

2.1.2 Deformable Models

Physics-based simulation is a natural choice for creating secondary motion effects such as flesh and fat jiggling as a character moves. Since the pioneering work of Terzopoulos and colleagues [1987], many simulation methods that can potentially be used for this purpose have been introduced. A comprehensive review of these works is outside the scope of this thesis, but the survey article of Nealen and colleagues [2006] provides more details on this topic.

Most of the methods for simulating volumetric objects require the simulation domain to be spatially discretized into tetrahedrons [Irving et al., 2004]. The characters used in animation environments are typically represented by surface meshes only, but there are many well-established tools such as *Tetgen*, *NetGen*, and *gmsh* for generating tetrahedral meshes from boundary representations. A more fundamental difference between physics-based simulation and character animation is that simulations endow each vertex of the mesh with individual degrees of freedom. To the extent that the motion must obey physics, the vertices are thus free to move independently from each other. This setup is in stark contrast to character animation, where the high-resolution surface mesh is constrained to deform only in the subspace defined by the rig. This representational mismatch typically implies that simulation must take place at a later stage of the animation pipeline and that results cannot easily be edited by animators.

The rig-space physics method of Hahn and colleagues [2012] was specifically designed to bypass this challenge, but it still needs to compute the motion of the internal vertices. This entails the solution of large systems of equations and consequently leads to high computational costs. Although there are simulation methods that do not require a volumetric representation, these are not without limitations. Shell models [Grinspun et al., 2003], for instance, can, in principle, be used to compute secondary motions on the character's

surface. However, shell models lack volume preservation by definition and thus cannot account for the natural bulging of flesh and other soft tissue. As another alternative, the boundary element method [James and Pai, 1999] condenses a volumetric problem to one with degrees of freedom only on the surface. However, this method only works well for linear problems and is therefore not attractive for modeling the highly nonlinear deformations exhibited by production-quality characters.

To alleviate these limitations, we aim to formulate an explicit, linear map that returns the position of interior vertices as a function of the configuration of the surface mesh. In effect, the deformation of the rig is automatically propagated everywhere in the interior of the simulation mesh, allowing for a very efficient implementation of subspace physics.

2.1.3 Subspace Physics

Reduced model methods for deformable objects are typically used to improve simulation speed. The underlying idea is to formulate the equations of motion in a low-dimensional subspace onto which the full-dimensional simulation model is projected. These subspaces can be defined by applying dimensionality reduction on sequences of meshes obtained through full simulations [James and Fatahalian, 2003], by embedding objects in low-resolution lattices [Faloutsos et al., 1997], by analyzing the vibration modes of an object [Barbič et al., 2009], or by using dual quaternion skinning to express the deformation of an object as a function of a small number of reference frames [Gilles et al., 2011]. While these methods are typically optimized for efficiency, rig-space physics is designed to operate in the deformation subspaces defined by arbitrary animation rigs. This generality, however, comes at a heavy computational price, which we aim to significantly lower with the method we propose.

2.1.4 Skinning

One of the key contributions of our method is that it enables the use of a deformation energy defined on a volumetric mesh, but without the need for additional internal degrees of freedom. We construct an explicit, example-based linear map that outputs the configuration of the internal vertices as a function of the surface mesh of the character. This approach is inspired by existing methods that compute skinning weights to map the motion of a set of coordinate frames onto a surface mesh [Magnenat-Thalmann et al., 1989]. There are many methods that aim to improve the quality of

skinning. Multi-linear methods [Wang and Phillips, 2002] use additional weights to improve the quality, as does the recent method of Jacobson and colleagues [2012]. Kavan and colleagues [2008] show that nonlinear skinning formulations can also lead to better quality, but our goal is to construct a linear map between surface and internal vertices. Another two groups of methods improves skinning using example shapes [Lewis et al., 2000; Sloan et al., 2001; Kry et al., 2002], or automatically compute skinning weights by optimizing for smoothness properties [Baran and Popović, 2007; Jacobson and Sorkine, 2011]. The recent method by Kavan and colleagues [2012] automatically computes optimized skinning weights by minimizing an elastic energy, which is similar in spirit to our approach. However, our skinning method does not try to alter the deformation of the surface mesh but defines the behavior of the interior.

Other research [James and Twigg, 2005; Hasler et al., 2010; Kavan et al., 2010; Le and Deng, 2012] specifically targets the problem of skinning animations, which entails finding transformations and corresponding skinning weights to best approximate a sequence of deforming meshes. Our work is closely related to these approaches. We first obtain a set of example deformations for the tetrahedral mesh by using physics-based simulation on a small number of artist-generated character poses. We then optimize for a sparse set of skinning weights that best explains the behavior of the internal vertices through the surface deformation.

Related methods to compute skinning weights create a low-resolution cage from the surface mesh and compute, for each internal vertex, the harmonic [Joshi et al., 2007], mean-value [Ju et al., 2005] or Green coordinates [Lipman et al., 2008]. While these methods lead to smooth deformation fields, they are not without drawbacks. Besides the fact that an artist needs to model and rig the cage, harmonic coordinates are expensive to compute, mean-value coordinates can lead to non-conformal interpolation, and the geometry interpolated using Green coordinates is not guaranteed to remain inside the control cage, which, for our problem setting, would result in inverted tetrahedrons. In addition, the deformation fields generated with these methods are disconnected from the elastic model used to represent the characters. We therefore resort to an example-based skinning method whose resulting deformation fields reflect the nature of the underlying material.

2.2 Sketch Abstractions for Character Posing

Researchers have long recognized the great potential of sketch-based user interfaces for a wide range of tasks in computer graphics. In this section, we

give an overview of this diverse field and analyze how different techniques relate to our setting of *Sketch Abstractions for Character Posing* that we will present in Chapter 5.

2.2.1 Sketch-Based Modeling

Sketch-based modeling is a challenging task in which a user's sketch is used to create a 3D shape. Depth ambiguities make the problem inherently intractable, since infinitely many different 3D objects can project to the same 2D camera image [Olsen et al., 2009]. To address this challenge, Igarashi and colleagues [1999] propose the use of drawn contours together with smoothness assumptions for 3D freeform sketch design, which can be further extended to support hidden segments and cusps [Karpenko and Hughes, 2006] or the automatic incorporation of rigging elements during the modeling process [Borosán et al., 2012]. Wyvill and colleagues [2005] follow a similar approach, but model the target object as an implicit surface, while Kraevoy, Sheffer, and van de Panne [2009] employ a 3D template to resolve ambiguities in drawn contours and recover a global deformation. Other work in this area uses additional annotations to mark features such as symmetries, cross-sections or alignment cues to restrict the space of possible solutions [Gingold et al., 2009].

Existing sketch-based modeling methods focus on the direct mapping of features in the input sketch to the 3D geometry, linking the quality of the results to the sketching ability of the user. Our method targets a different problem. We focus on sketch abstractions on top of artist-designed rigs, allowing character designers to expose pose and shape variability to the sketch-based posing system at the level of detail that matches their design intent for the character.

2.2.2 Mesh Deformation

Mesh deformation using sketch-based interfaces is an appealing alternative to traditional shape editing methods. SilSketch [Nealen et al., 2005; Zimmermann et al., 2007] allows the user to redraw screen-space silhouette strokes of a model and adjusts the 3D vertex positions so that the silhouette matches the input stroke. Curves derived from 2D cartoons can serve as constraints in the volumetric graph Laplacian [Zhou et al., 2005] to apply stylized deformations to 3D meshes. Kho and Garland's method [2005] provides an intuitive interface for deforming unstructured polygon meshes in which screen-space curves define a region of interest over the mesh. Redrawing or

manipulating the curve induces a mesh deformation. The FiberMesh system of Naelen and colleagues [2007] represents a hybrid approach between editing and freeform modeling. While the initial model is created using a sketch-based modeling paradigm, the strokes used to draw the shape remain on the 3D mesh and allow intuitive and interactive sketch-based editing of the geometry.

These mesh deformation methods operate in a global space and focus on deformations applied directly to vertices. However, in professional animation production pipelines, the animation workflow revolves around setting keyframes on rig parameters. A character's rig is carefully constructed to express the space of desired and allowable deformations. Moving vertices arbitrarily, as is done with mesh deformation systems, breaks the consistency that the rig affords. Our system instead targets an artist-designed subspace in the form of a character rig, allowing the artist to determine what type of variability is appropriate and how that variability is exposed via the sketch abstraction. Thus, our generated poses always conform to the artist's original design intent, since the optimization takes place within the artist-created subspace defined by the character rig.

2.2.3 Retrieval-and-Composition

Retrieval-and-composition represents another approach for sketch-based modeling that is inspired by classical information retrieval algorithms. Funkhouser and colleagues [2004] implement a system based on a database of 3D models that can be queried via keywords or existing parts of shapes. The retrieved meshes can then be composed into a new mesh via cut and paste operations. Lee and Funkhouser [2008] extend this method with sketch-based querying in order to enable a *sketch, cut and paste* 3D modeling process. Shin and Igarashi [2007] use a similar system for the sketch-based composition of scenes.

While these systems allow quick composition of detailed, artist-created objects, general-purpose customization beyond extraction and rigid transformation is difficult. Furthermore, retrieval-and-composition algorithms often require the user to embed strokes in 3D by drawing on different planes to assist the 3D retrieval process. As we show in our results, our system can enhance retrieval-and-composition algorithms by incorporating artist-controlled shape variability via arbitrary rigging controls as well as a more iconographic 2D representation of objects via the sketch abstraction.

2.2.4 Sketch-Based Posing

Sketch-based posing uses sketching interfaces for shape changes that conform to a character's rig or other pose-based parameterizations, often in the form of a skeletal structure. Early work in this field focuses on estimating bone positions from a 2D stick-figure representation [Davis et al., 2003]. Lin and colleagues [2010] explore the stick figure sketching paradigm in the context of designing sitting poses. The method of Wei and Chai [2011] lets users sketch bone positions that are then matched with natural human body poses from a large motion capture database, which Choi and colleagues [2012] extend to support the retrieval of whole motion sequences. Motivated by techniques from hand-drawn animation, Guay and colleagues [2013] infer the principal pose of a character from a single 2D input curve, the *line of action*. In order to support extreme deformations, Öztireli and colleagues [2013] combine the sketching of curved, stretchable bones with a novel skinning method that supports extreme rotations.

While powerful, these methods target skeletal rigging formulations and use a prescribed sketching methodology irrespective of the design intent of the character's creator. In contrast, the distinguishing aspect of our work is that we offer the character designer the ability to explicitly construct the representation, in the form of a sketch abstraction, that is used to sketch new character poses. In addition, our formulation generalizes to arbitrary rigging controls and does not restrict the type of deformers used to rig the character.

Other sketch-based posing systems target facial animation. Researchers use statistical models of captured [Lau et al., 2009] or generated [Gunnarsson and Maddock, 2010] face shapes in place of traditional character rigs to recover face poses that match sketched curves. In contrast, Miranda and colleagues [2012] provide a facial sketching interface specialized for bone-based rigs that derives bone deformations from sketched curves. While these methods prescribe a particular rig type that may not match animation workflows, the work of Chang and Jenkins [2006] supports arbitrary face rigs with a black-box optimization system that aligns sketched reference and target curves. This functionality matches our on-the-fly generation of sketch abstractions. However, while the method of Chang and Jenkins [2006] is designed for non-hierarchical articulation and is limited to fewer than 20 rigging variables, our method supports arbitrary rigs with hundreds of variables. Furthermore, the core focus of our work is not on aligning individual source and target curves but on allowing artists to build more elaborate sketch representations that can be used with our efficient posing system.

Finally, in the field of mechanical design, Coros and colleagues [2013] sketch

the desired motion curves for end-effectors of mechanical assemblies, essentially defining an animation curve via sketch input. In a similar fashion, methods have been proposed that do not create entirely new geometry, but instead pose an existing character using sketched user input.

2.3 Subspace Clothing Simulation

In this section, we discuss related methods and previous work with respect to the *Subspace Clothing Simulation* method that we will present in Chapter 6.

2.3.1 Cloth Simulation

Cloth simulation is a well-explored field and existing works are far too numerous to be listed here. The work of Baraff and Witkin [1998] was a major breakthrough in terms of computational efficiency and even though the following 15 years have seen many improvements, high-resolution cloth simulation is still very time-consuming.

Nevertheless, there are many methods that aim for faster cloth simulation. One line of work combines simulation on a coarse base mesh with a fast method for adding geometric details. The method of Rohmer and colleagues [2010] adds geometrically generated wrinkles based on the strain field of the coarse simulation. Mueller and Chentanez [2010] attach a high-resolution mesh to a coarse simulation, whose deformation is determined using fast static solves.

Another stream of work exploits precomputed data to avoid run-time simulations altogether. De Aguiar and colleagues [2010] present a technique for learning a linear conditional cloth model that can be trained with data from physics-based simulations. The method achieves very fast computation times, but it primarily targets low-complexity cloth with little folding. The method of Guan and colleagues [2012] factors clothing deformations into components due to body shape and pose, and learns a linear model in order to quickly dress different characters without run-time simulations. An alternative way of exploiting precomputed data was suggested by Kim and colleagues [2013], who create an exhaustive set of secondary motion to accompany a given primary motion graph. Since no run-time simulation is required, this method is very fast. However, the character's range of motion has to be small enough to fit a motion graph, which is not the case for production-level character animations. Kim and Vendrovsky [2008] make

use of precomputed data to drive the deformation of clothing using the animated underlying model of the character wearing it. While our method also relies on this correspondence between pose and cloth deformation, we perform a physical simulation to achieve faithful results rather than merely interpolating motions from the input data.

Yet another class of methods combines coarse simulations and precomputed data. Feng and colleagues [2010] describe an approach which decomposes a high-resolution simulation into mid- and fine-scale deformations. For the mid-scale deformations, the mesh is further decomposed into a set of bone clusters for which skinning weights are fit in a way similar to [James and Twigg, 2005], while fine-scale details are added based on a PCA-analysis of residual vectors as in [Kry et al., 2002]. Both mid- and fine-scale details are then driven by a coarse scale simulation, which is fast enough to yield real-time rates. Focussing on fitted clothing, Wang and colleagues [2010] present an example-based approach that augments coarse simulations with pose-dependent detail meshes obtained from a wrinkle database. The wrinkle database stores per-joint wrinkle meshes that are precomputed from high-resolution simulations and merged together at run time. Targeting the more general case of free-flowing cloth, Kavan and colleagues [2011] describe a method for learning linear upsampling operators from high-resolution simulations. With similar goals, Zurdo and colleagues [2013] combine multi-resolution and pose-space deformation (PSD) techniques in order to augment coarse simulations with example-based wrinkles.

Similar to these works, our method uses data from high-resolution simulations, but rather than augmenting a coarse simulation, we construct a low-dimensional subspace that allows for fast simulation of detailed clothing deformations.

Finally, another option for performance improvements is to leverage the processing power of parallel architectures [Selle et al., 2009]. However, while significant acceleration factors have been reported for large data of around two million triangles [Selle et al., 2009], the improvements for typical problem sizes are rather modest.

2.3.2 Subspace Simulation

Subspace simulation is generally most attractive when high-resolution models undergo low-rank deformations. The problem of subspace integration and model reduction for the simulation of elastics was originally formulated in the field of engineering [Krysl et al., 2001], but we focus on works

from computer graphics for the sake of conciseness. In this context, the method of Barbič and James [2005] was the first to demonstrate, and unleash, the potential of model reduction for accelerating the computation of elastic deformations on high-resolution meshes. Subsequent work by An and colleagues [2008] showed that a selective evaluation of elemental contributions, also known as cubature, can improve the asymptotic complexity of subspace methods. While subspace methods can be very efficient for cases with small or predictable deformations, the generalization is made difficult by the discrepancy between a low-dimensional basis and a large range of deformations. Kim and colleagues [2009] address this problem with a hybrid solution that combines subspace and full-space simulation and, for each step, decides which one to use. One technically interesting aspect of this work is the use of an adaptive basis that is dynamically updated with results from the full-space solver, on which our method also relies. However, our updates are much more frequent (once per Newton iteration) and do not require online full-space simulation. While most subspace methods rely on a linear basis, Hahn and colleagues [2012] simulate the deformation of a character's fat and muscles in the nonlinear subspace induced by its rig.

The problem of detecting and resolving collisions in the context of subspace simulation has recently gained attention. Barbič and James [2010] showed how bounding volume hierarchies can be enhanced by certificates that allow aggressive culling of overlapping tests, and Zheng and James [2012] extended this approach to also consider deformation energy. Wong and colleagues [2013] propose a method for efficient self collision culling for skeleton-driven deforming meshes. While we do not address subspace self-collision culling in this work and simply resort to full-space collision resolution, we note that many of these ideas could also be applied to our setting of subspace simulation using adaptive bases.

Harmon and colleagues [2013] dynamically augment the subspace basis with analytical functions that model deformations due to individual contact points. Since these augmentation vectors only add very localized displacements, they are able to project their current subspace coordinate vector into the new basis whenever it changes to ensure temporal coherence. In our case, the basis regularly incurs drastic changes which motivates the use of a more sophisticated approach to obtain smooth transitions.

2.3.3 Pose-Space Deformation

The concept of making shape depend on positions in a *pose space* was originally proposed by Lewis and colleagues [Lewis et al., 2000] and further ex-

tended in the context of example-based deformations [Sloan et al., 2001], medical imaging [Kurihara and Miyata, 2004] and real-time applications [Kry et al., 2002]. The skeletal shape deformation method by Weber and colleagues [2007] is similar in spirit, but based on a differential surface representation. Meyer and Anderson [2007] proposed Key Point Subspace Acceleration and soft caching to accelerate pose-dependent deformation queries. Zurdo and colleagues [2013] use PSDs to enhance a low-resolution simulation with example-based wrinkle details. The quality of pose-space deformation methods heavily depends on the way the scattered-data interpolation problem in pose space is resolved. To this end, Lee [2009] explored the space of basis functions, while Bengio and Goldenthal [2013] propose a simplicial interpolation scheme to make the interpolation space more controllable.

Similar to these works, our method is also based on the assumption that deformations are inherently pose-dependent. However, instead of interpolating deformations in pose space, we select them automatically from nearby locations and let our solver handle the transitions between them.

CHAPTER 3

Foundations

In the previous chapters, we have given an overview over previous research related to the work in this thesis. In this chapter, we will introduce the mathematical framework that will be used throughout this work. We will touch on several topics that will later see use in the subsequent chapters on applications to *Efficient Rig-Space Physics Simulation*, *Sketch Abstractions for Character Posing*, and *Subspace Clothing Simulation*. To begin with, an overview over physical objects and their simulation over time will be provided, before we head deeper into optimization techniques in the sections on energy minimization and subspace simulation. Further, we will discuss some analytic forms of commonly used rigs, as well as give a brief description of the contact handling methods we use.

3.1 Physics

This section introduces the various layers of abstractions we will use to model physical objects for their simulation over time. While we will state some of the derivations related to the physical material models, the finite element method and the time integration used in this work, the presented model will allow us to later look at physical simulations on a much higher level—even without having to fully understand them anymore. In fact, we will arrive at the formulation of a flexible objective function that we simply have to minimize with respect to its parameters in order to advance the simulation forward in time.

3.1.1 Deformable Objects and Elements

We assume that the deformable object we are interested in simulating can be modeled as an elastic object represented by a a sequence of *n vertices* $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$. In the typical three-dimensional case, each of the vertices $\mathbf{x}_i \in \mathbb{R}^3$ can be stacked up to form a state vector of dimensions $3n$:

$$\mathbf{x} = \left[(\mathbf{x}_1)_x, (\mathbf{x}_1)_y, (\mathbf{x}_1)_z, (\mathbf{x}_2)_x, (\mathbf{x}_2)_y, (\mathbf{x}_2)_z, \ldots, (\mathbf{x}_n)_x, (\mathbf{x}_n)_y, (\mathbf{x}_n)_z\right]^T \tag{3.1}$$

While a lower-case $\mathbf{x}$ denotes *deformed* positions, we will use the capital $\mathbf{X}$ to denote the *undeformed* positions of the vertices and also refer to it as the object's *rest state*.

Depending on the type of elastic object we are simulating, the vertices are connected by primitives such as edges, triangles or tetrahedrons to which we will refer as deformable *elements*. The state of an element e is not defined in terms of the full state vector $\mathbf{x}$, but by a subset of size $3m$ of the dimensions of $\mathbf{x}$, where m is the number of vertices an element refers to. For example, an edge element would use $m = 2$, while $m = 3$ would be used for a triangle element, and $m = 4$ for a tetrahedron element. We refer to the deformed state vector of an element containing only these dimensions of $\mathbf{x}$ as $\mathbf{x}^e \in \mathbb{R}^{3m}$. Analogously, the undeformed state of an element referring to a subset of the dimensions of $\mathbf{X}$ is called $\mathbf{X}^e \in \mathbb{R}^{3m}$. Together, the vertices and the elements of a deformable object form a *simulation mesh*.

Given a user-specified mass density ρ, we apply standard mass lumping by computing the masses of each element in the object's rest state and then distributing it to the element's vertices in equal parts, resulting in a diagonal mass matrix $\mathbf{M} \in \mathbb{R}^{3n \times 3n}$. Not all of the deformable elements necessarily contribute mass: For a deformable shell, a triangular area element will typically contribute mass, while a hinge element connecting two triangles over a shared edge will not.

3.1.2 Elastic Energy

The behavior of the deformable object is governed by an *elastic energy* W_{elastic}, whose precise form is defined by a deformation measure and a material law. Following the conventions of the *finite element method* (FEM), we discretize the continuous physical energy over the surface or volume of the object as the sum of per-element energy density measures W^e_{elastic} such that

$$W_{\text{elastic}}(\mathbf{x}, \mathbf{X}) = \sum_{\text{element } e} V^e(\mathbf{X}^e) \cdot W^e_{\text{elastic}}(\mathbf{x}^e, \mathbf{X}^e), \tag{3.2}$$

where V^e denotes the *generalized volume* of the element. For edges, the generalized volume denotes their edge length, for triangles, it denotes their surface area, and for tetrahedrons their volume.

While this flexible elastic energy formulation supports a wide variety of material laws, we will focus on two specific ones for the simulation of elastic solids and cloth respectively. We will now look at both cases individually and state their specific instantiation of W^e_{elastic} in Equation (3.2).

Elastic Solids

Letting $\mathbf{x}^e = (\mathbf{x}^e_1, \dots \mathbf{x}^e_4)$ and $\mathbf{X}^e = (\mathbf{X}^e_1, \dots \mathbf{X}^e_4)$ denote the deformed and undeformed positions of a given tetrahedral element, we compute its *deformation gradient* as

$$\mathbf{F} = \mathbf{d}\mathbf{D}^{-1} \in \mathbb{R}^{3\times 3}, \tag{3.3}$$

where $\mathbf{d}, \mathbf{D} \in \mathbb{R}^{3\times 3}$ are matrices whose columns hold the deformed and undeformed edge vectors $\mathbf{x}^e_i - \mathbf{x}^e_1$ and $\mathbf{X}^e_i - \mathbf{X}^e_1$ for $2 \leq i \leq 4$, respectively.

The *Green strain* is then given as

$$\mathbf{E} = \frac{1}{2}\left(\mathbf{F}^T\mathbf{F} - \mathbf{I}_3\right) \in \mathbb{R}^{3\times 3}, \tag{3.4}$$

where $\mathbf{I}_3 \in \mathbb{R}^{3\times 3}$ denotes the identity matrix.

Using a modified St. Venant-Kirchhoff material as described by Martin and colleagues [2011], the elastic energy density per element is obtained as

$$W^e_{\text{elastic}}(\mathbf{x}^e, \mathbf{X}^e) = \mu \cdot \frac{1}{2}\|\mathbf{E}\|^2_F + \lambda \cdot \left(1 - \frac{V^e(\mathbf{x}^e)}{V^e(\mathbf{X}^e)}\right), \tag{3.5}$$

where $\|\cdot\|_F$ is the Frobenius norm, and μ, λ are material parameters. The first parameter μ controls the amount of resistance of the solid to stretching and shearing, while the second parameter λ controls the resistance to volume changes of the solid, while implicitly preventing element inversions.

Cloth

Stretching Resistance Let $\mathbf{x}^e = (\mathbf{x}^e_1, \mathbf{x}^e_2, \mathbf{x}^e_3)$ and $\mathbf{X}^e = (\mathbf{X}^e_1, \mathbf{X}^e_2, \mathbf{X}^e_3)$ denote the deformed and undeformed positions of a given triangle element, respectively. Following Thomaszewski and colleagues [2008], we use *constant strain triangles* (CST) to resist stretching. To this end, we follow the standard approach and notation from Bonet and Wood [1997] and first define *shape*

functions with respect to a two-dimensional parameterization of the cloth, which is motivated by the observation that we can look at cloth as a 2D surface embedded in 3D space. The three shape functions of a single triangle for each of its three nodes are given as:

$$\begin{aligned} N_1(\xi_1, \xi_2) &= 1 - \xi_1 - \xi_2 \\ N_2(\xi_1, \xi_2) &= \xi_1 \\ N_3(\xi_1, \xi_2) &= \xi_2 \end{aligned} \tag{3.6}$$

The scalars ξ_1 and ξ_2 represent *material coordinates* in 2D, and deriving the shape function with respect to them yields

$$\frac{\partial \mathbf{N}}{\partial \boldsymbol{\xi}} = \begin{bmatrix} -1 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{3.7}$$

where $\mathbf{N} = [N_1, N_2, N_3]^T$ and $\boldsymbol{\xi} = [\xi_1, \xi_2]^T$.

Let $\mathbf{T} \in \mathbb{R}^{3\times3}$ be the transformation matrix that rotates the undeformed triangle nodes $\mathbf{X}^e$ of an element to the xy-plane while preserving relative axis orientations, and $\check{\mathbf{T}} \in \mathbb{R}^{2\times3}$ its truncated form without the third row. The undeformed triangle nodes in 2D coordinates on the xy-plane are then given as $\check{\mathbf{X}}^e = (\check{\mathbf{X}}_1^e, \check{\mathbf{X}}_2^e, \check{\mathbf{X}}_3^e)$, where

$$\check{\mathbf{X}}_i^e = \check{\mathbf{T}}\mathbf{X}_i^e \in \mathbb{R}^2 \tag{3.8}$$

We can now express any point $\check{\mathbf{p}}$ in the xy-plane as a function of the material coordinates $\boldsymbol{\xi}$, and vice versa:

$$\check{\mathbf{p}}(\xi_1, \xi_2) = \check{\mathbf{X}}_1^e + \xi_1 \cdot (\check{\mathbf{X}}_2^e - \check{\mathbf{X}}_1^e) + \xi_2 \cdot (\check{\mathbf{X}}_3^e - \check{\mathbf{X}}_1^e) \tag{3.9}$$

Making use of Equation (3.8), the Jacobian of $\check{\mathbf{p}}$ with respect to the material coordinates $\boldsymbol{\xi}$ is then given as

$$\frac{\partial \check{\mathbf{p}}}{\partial \boldsymbol{\xi}} = [\check{\mathbf{X}}_2^e - \check{\mathbf{X}}_1^e \mid \check{\mathbf{X}}_3^e - \check{\mathbf{X}}_1^e] = \check{\mathbf{T}}\mathbf{D} \in \mathbb{R}^{2\times2}, \tag{3.10}$$

where $\mathbf{D} \in \mathbb{R}^{3\times2}$ is the matrix whose two columns hold the undeformed edge vectors $\mathbf{X}_2^e - \mathbf{X}_1^e$ and $\mathbf{X}_3^e - \mathbf{X}_1^e$ in 3D space.

Combining Equations (3.7) and (3.10), we can now compute the *shape function derivatives* of a triangular element with respect to its 2D positions $\check{\mathbf{p}}$ on the xy-plane as

$$\frac{\partial \mathbf{N}}{\partial \check{\mathbf{p}}} = \frac{\partial \mathbf{N}}{\partial \boldsymbol{\xi}} \left(\frac{\partial \check{\mathbf{p}}}{\partial \boldsymbol{\xi}} \right)^{-1} \in \mathbb{R}^{3\times2}, \tag{3.11}$$

where we made use of the chain rule. Rather than computing the inverse of $\frac{\partial \check{\mathbf{p}}}{\partial \check{\boldsymbol{\xi}}}$, we compute each row of $\frac{\partial \mathbf{N}}{\partial \check{\mathbf{p}}}$ separately by solving the linear system

$$\left(\frac{\partial \check{\mathbf{p}}}{\partial \check{\boldsymbol{\xi}}}\right)^T \left(\frac{\partial N_i}{\partial \check{\mathbf{p}}}\right)^T = \left(\frac{\partial N_i}{\partial \check{\boldsymbol{\xi}}}\right)^T \tag{3.12}$$

for $i = 1, 2, 3$ with different right-hand sides to prevent numerical instabilities.

Given the deformed nodal positions $\mathbf{x}^e$, we can now compute the *deformation gradient* as the following sum of outer produces:

$$\mathbf{F} = \left(\mathbf{x}_1^e\right)^T \frac{\partial N_1}{\partial \check{\mathbf{p}}} + \left(\mathbf{x}_2^e\right)^T \frac{\partial N_2}{\partial \check{\mathbf{p}}} + \left(\mathbf{x}_3^e\right)^T \frac{\partial N_3}{\partial \check{\mathbf{p}}} \in \mathbb{R}^{3\times 2} \tag{3.13}$$

Analogously to the solid model from Equation (3.4), the *Green strain* is obtained as

$$\mathbf{E} = \frac{1}{2}\left(\mathbf{F}^T\mathbf{F} - \mathbf{I}_2\right) \in \mathbb{R}^{2\times 2}, \tag{3.14}$$

where $\mathbf{I}_2 \in \mathbb{R}^{2\times 2}$ denotes the identity matrix.

Again using the simplified St. Venant-Kirchhoff material, we compute the elastic stretching energy density as:

$$W^e_{\text{stretch}}(\mathbf{x}^e, \mathbf{X}^e) = \mu \cdot \frac{1}{2} \|\mathbf{E}\|_F^2, \tag{3.15}$$

where $\|\cdot\|_F$ is the Frobenius norm, and μ a material parameter that controls the amount of in-plane resistance to stretching of the cloth.

Bending Resistance Since the presented stretching energy W^e_{stretch} is purely two-dimensional and thus only resists in-plane deformation of each cloth triangle, we need an additional energy term to incorporate bending resistance. We choose the *hinge element* approach by Grinspun and colleagues [2003], which resists angular deformation over each inner edge of the mesh that is shared by two triangles.

Even though the angle between two triangles is a one-dimensional measure, a hinge element refers to four vertices to describe both triangles including the shared edge. Let $\mathbf{x}^e = (\mathbf{x}_1^e, \dots \mathbf{x}_4^e)$ and $\mathbf{X}^e = (\mathbf{X}_1^e, \dots \mathbf{X}_4^e)$ denote the deformed and undeformed positions of a given hinge element, respectively, where vertices 1 and 2 define the shared edge and vertices 3 and 4 define

the opposite corners of the two triangles. The two deformed normals of the triangles can then be computed as

$$\begin{aligned} \mathbf{n}_1^e &= \frac{\left(\mathbf{x}_1^e - \mathbf{x}_0^e\right) \times \left(\mathbf{x}_2^e - \mathbf{x}_0^e\right)}{\left\| \left(\mathbf{x}_1^e - \mathbf{x}_0^e\right) \times \left(\mathbf{x}_2^e - \mathbf{x}_0^e\right) \right\|_2}, \\ \mathbf{n}_2^e &= \frac{\left(\mathbf{x}_3^e - \mathbf{x}_0^e\right) \times \left(\mathbf{x}_1^e - \mathbf{x}_0^e\right)}{\left\| \left(\mathbf{x}_3^e - \mathbf{x}_0^e\right) \times \left(\mathbf{x}_1^e - \mathbf{x}_0^e\right) \right\|_2}, \end{aligned} \tag{3.16}$$

and the two undeformed normals $\mathbf{N}_1^e$ and $\mathbf{N}_2^e$ can be computed by replacing $\mathbf{x}_i^e$ with $\mathbf{X}_i^e$.

The deformed dihedral angle θ between the two adjacent triangles can then be computed as

$$\theta^e = \arccos\left(\left(\mathbf{n}_1^e\right)^T \mathbf{n}_2^e \right), \tag{3.17}$$

while the undeformed dihedral angle Θ^e is computed by replacing $\mathbf{n}_1^e$ and $\mathbf{n}_2^e$ with $\mathbf{N}_1^e$ and $\mathbf{N}_2^e$.

The elastic bending energy density is now given as the quadratic one-dimensional spring potential

$$W_{\text{bending}}^e(\mathbf{x}^e, \mathbf{X}^e) = k_{\text{bending}} \cdot \frac{1}{2} \left(\theta^e - \Theta^e\right)^2, \tag{3.18}$$

where k_{bending} is a material parameter controlling the amount of bending resistance for the cloth.

Final Energy To compute the combined elastic energy $W_{\text{elastic}}(\mathbf{x}, \mathbf{X})$ for the cloth model, we instantiate Equation (3.2) twice: Once, we use W_{stretch}^e from Equation (3.15) to get W_{stretch}, and then W_{bending}^e from Equation (3.18) to get W_{bending}. This results in a final cloth model energy of

$$W_{\text{elastic}}(\mathbf{x}, \mathbf{X}) = W_{\text{stretch}}(\mathbf{x}, \mathbf{X}) + W_{\text{bending}}(\mathbf{x}, \mathbf{X}). \tag{3.19}$$

Note that there is no need to introduce additional weighting factors between the energy terms since the material parameters μ and k already provide the necessary degrees of freedom to accomplish that. Since the stretching energy was defined as a 2D measure, the generalized volume V^e from Equation (3.2) measures the surface area for all triangular elements. In the case of the bending energy, the hinge elements measure the angular deformation over a 1D edge, so the generalized volume V^e in that case measures the edge length. We also disable mass contribution for the hinge elements to prevent double counting the vertices used by both stretching and bending elements.

3.1.3 External Energy

Similar to the elastic energy of a deformable object, we will also model the external forces acting on it as energy potentials. This will enable the use of a flexible variational energy minimization framework to be introduced in Section 3.1.4. We will specifically look at *gravitational energy*, *constraint energy* and *contact energy* potentials here, but it is worth noting that the eventual solver framework will support arbitrary potentials, which we will make use of for the *Sketch Abstractions for Character Posing* application in Section 5.2.

Gravitational Energy We will generally denote the y axis as the "up axis", resulting in a gravitational acceleration of $[0, -g, 0]^T$, where g is a parameter specifying the gravitational strength of the simulated world. Given the deformed object positions $\mathbf{x}$, the gravitational energy is then obtained as

$$W_{\text{gravity}}(\mathbf{x}) = \sum_{i=1}^{n} m_i \cdot g \cdot (\mathbf{x}_i)_y, \tag{3.20}$$

where m_i denotes the mass of vertex i of the deformable object.

Constraint Energy We will model point constraints by attaching springs with zero rest length to them. Given a set $\mathcal{C} = \{c_1, c_2, \ldots, c_{n_c}\}$ of n_c constrained vertex indices, constraint coefficients $\mathbf{k}_{\text{constraint}} \in \mathbb{R}^{n_c}$ and attachment points $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{n_c} \in \mathbb{R}^3$, the constraint energy is given as

$$W_{\text{constraint}}(\mathbf{x}) = \sum_{i=1}^{n_c} (\mathbf{k}_{\text{constraint}})_i \cdot \frac{1}{2} \left\| \mathbf{x}_{c_i} - \mathbf{c}_i \right\|^2 \tag{3.21}$$

Contact Energy Even though we will address contact handling in more detail in Section 3.5, external contact of the deformable model with other objects can be resolved with simple directional spring potentials if small penetrations between the objects are acceptable. Let $\mathcal{P} = \{p_1, p_2, \ldots, p_{n_p}\}$ denote the set of n_p penetrating vertex indices, $\mathbf{n}_1^p, \mathbf{n}_2^p, \ldots \mathbf{n}_{n_p}^p$ the contact normals for each penetrating vertex, and $\mathbf{s}_1^p, \mathbf{s}_2^p, \ldots, \mathbf{s}_{n_p}^p$ for each deformable model vertex the closest point on the surface of the external object in contact. Following McAdams and colleagues [2011], the resulting contact energy measure is

$$W_{\text{contact}}(\mathbf{x}) = \sum_{i=1}^{n_p} k_{\text{contact}} \cdot \frac{1}{2} \left(\mathbf{x}_{p_i} - \mathbf{s}_i^p \right)^T \mathbf{N}_i \left(\mathbf{x}_{p_i} - \mathbf{s}_i^p \right), \tag{3.22}$$

where $\mathbf{N}_i = (1-\alpha) \cdot \mathbf{n}_i^p \left(\mathbf{n}_i^p\right)^T + \alpha \cdot \mathbf{I}_3$, and $\mathbf{I}_3 \in \mathbb{R}^{3\times3}$ denotes the identity matrix. The spring coefficient k_{contact} controls the amount of contact resistance that should be applied, while the parameter $\alpha \in [0,1]$ controls the amount of frictional resistance that should be applied at the contact points.

3.1.4 Equations of Motion

To simulate dynamics for our deformable objects, we adopt the variational energy approach from Martin and colleagues [2011]. We will start with the equations of motion (EOM) corresponding to Newton's second law for deformable materials

$$\rho \cdot \ddot{x} = f(x), \tag{3.23}$$

where ρ denotes mass density, f the sum of all continuous force densities acting on the object for the continuous solution $x(t)$ over time t.

Following the derivation by Liu and colleagues [2013], we apply the finite element method and discretize over space and time using the implicit Euler method, where the state at time t is denoted by the superscript ^{t}. Using a time step of size h, we arrive at the following coupled system of discrete equations of motion:

$$\begin{aligned} \mathbf{x}^{t+1} &= \mathbf{x}^t + h \cdot \mathbf{v}^{t+1} \\ \mathbf{v}^{t+1} &= \mathbf{v}^t + h \cdot \mathbf{M}^{-1}\mathbf{f}\left(\mathbf{x}^{t+1}\right) \end{aligned} \tag{3.24}$$

Solving the first line of the system in Equation (3.24) for the current and next velocities $\mathbf{v}^t$ and $\mathbf{v}^{t+1}$ yields:

$$\begin{aligned} \mathbf{v}^t &= \frac{\mathbf{x}^t - \mathbf{x}^{t-1}}{h} \\ \mathbf{v}^{t+1} &= \frac{\mathbf{x}^{t+1} - \mathbf{x}^t}{h} \end{aligned} \tag{3.25}$$

We can now decouple the system in Equation (3.24) by inserting the velocities from Equation (3.25) into its second line to obtain

$$\mathbf{M}\left(\frac{\mathbf{x}^{t+1} - 2\mathbf{x}^t + \mathbf{x}^{t-1}}{h}\right) = h \cdot \mathbf{f}\left(\mathbf{x}^{t+1}\right), \tag{3.26}$$

where we have also multiplied with the discrete mass matrix $\mathbf{M}$ from the left. Because the forces $\mathbf{f}$ are typically a higher-order function of the unknown positions $\mathbf{x}^{t+1}$, Equation (3.26) is a nonlinear problem that we are aiming to solve.

Using the variational formulation by Martin and colleagues [2011], we can now express the solution of Equation (3.26) as an optimization problem. We start by expressing the forces $\mathbf{f}$ as the negative gradient of an energy potential W:

$$\mathbf{f}(\mathbf{x}) = -\frac{\partial}{\partial \mathbf{x}} W(\mathbf{x}) \tag{3.27}$$

The objective function can now be stated as:

$$H(\mathbf{x}) = \frac{1}{2h^2} \left(\mathbf{x} - 2\mathbf{x}^t + \mathbf{x}^{t-1}\right)^T \mathbf{M} \left(\mathbf{x} - 2\mathbf{x}^t + \mathbf{x}^{t-1}\right) + h \cdot W(\mathbf{x}) \tag{3.28}$$

Minimizing this objective function corresponds to solving Equation (3.26) because setting its gradient $\frac{\partial H}{\partial \mathbf{x}}$ to zero, denoting its minimizer by $\mathbf{x}^{t+1}$, and applying Equation (3.27) results in exactly the same term. This formulation is highly flexible because it supports arbitrary energy potentials to be included in W by defining it as the sum of the different potentials we have introduced before:

$$W(\mathbf{x}) = W_{\text{elastic}}(\mathbf{x}) + W_{\text{gravity}}(\mathbf{x}) + W_{\text{constraint}}(\mathbf{x}) + W_{\text{contact}}(\mathbf{x}) \tag{3.29}$$

We can further simplify the objective function (3.28) if we look at its first part as an imaginary *momentum energy* potential W_{momentum}

$$W_{\text{momentum}}(\mathbf{x}) = \frac{1}{2h^3} \left(\mathbf{x} - 2\mathbf{x}^t + \mathbf{x}^{t-1}\right)^T \mathbf{M} \left(\mathbf{x} - 2\mathbf{x}^t + \mathbf{x}^{t-1}\right). \tag{3.30}$$

Equation (3.28) then reduces to

$$H(\mathbf{x}) = h \cdot \sum_{W \in \mathcal{W}} W(\mathbf{x}), \tag{3.31}$$

where $\mathcal{W} = \{W_{\text{momentum}}, W_{\text{elastic}}, W_{\text{gravity}}, W_{\text{constraint}}, W_{\text{contact}}\}$ denotes the set of used energy potentials.

3.1.5 Static Problems

The potential energy formulation of the objective function in Equation (3.31) easily admits solving static problems as well. By excluding W_{momentum} from the potential set $\mathcal{W}$ and setting $h = 1$, the minimizer of $H(\mathbf{x})$ will be the static solution of the deformable object with respect to the boundary conditions set by the two potentials $W_{\text{constraint}}$ and W_{contact}. This is because setting the gradient $\frac{\partial H}{\partial \mathbf{x}}$ to zero is equivalent to requiring that all forces acting on the object sum to zero, providing the necessary conditions for a static equilibrium.

If neither $W_{\text{constraint}}$ nor W_{contact} provide necessary boundary conditions for the static problem, the objective function $H(\mathbf{x})$ will be unbounded below and no minimizer will exist. A practical example for this case would be a deformable object on which only elastic energy and gravity is acting, such that $\mathcal{W} = \{W_{\text{elastic}}, W_{\text{gravity}}\}$. Since no other constraints prevent the object from falling, no static solution exists since the object would simply fall to negative infinity in the limit. This problem does not occur in the dynamic case when W_{momentum} is part of the potential set, since the momentum energy potential effectively acts as a regularizer on the indefinite optimization problem: Even though the object will continue falling indefinitely with increasing velocity, the amount of distance it can travel in one time step will always be limited because W_{momentum} counteracts any motion shorter or longer than what would be expected if the current velocity was maintained.

Another interesting case with static problems arises when there is no single solution that minimizes the objective function $H(\mathbf{x})$, but a whole space of minimizers. We will refer to these minimizers as a *null space*, since it refers to all states $\mathbf{x}$ for which $\frac{\partial}{\partial \mathbf{x}} H(\mathbf{x}) = 0$. A typical example of such a situation would be a case where the constraints provided by $W_{\text{constraint}}$ admit any rotation around a certain axis. If we simulate a deformable object where only single vertex is constrained, any rotation around the axis of gravity of a state $\mathbf{x}$ will not change the objective function $H(\mathbf{x})$ because all energy potentials in $\mathcal{W}$ are invariant to such transformations. Since we will be using an iterative solver to minimize $H(\mathbf{x})$, the actual solution from the null space that we obtain will depend on the provided initial guess.

3.1.6 Energy Derivatives

In the following Section 3.2, we will describe a numerical approach to minimize the objective function from Equation (3.31). Since we will use a gradient-based approach, we need to be able to evaluate the first and second derivatives of $H(\mathbf{x})$ with respect to the state vector $\mathbf{x}$ efficiently. The simplified objective function formulation from Equation (3.31) allows us to look at the energy potentials in $\mathcal{W}$ individually since the derivative of a sum is simply the sum of the derivatives. For W_{momentum}, W_{gravity}, $W_{\text{constraint}}$ and W_{contact}, determining analytic derivatives is trivial since these potentials can be expressed as a sum over all the n vertices in $\mathbf{x}$ of the simulated object. Once again using the linearity of the derivative operator, the three gradient dimensions corresponding to the i-th vertex will only depend on the vertex $\mathbf{x}_i$. The same holds for the Hessian, which will thus be sparse except for a 3×3 block on the diagonal for each vertex.

The only remaining potential we need to consider is the elastic energy W_{elastic}. If we apply the linear derivative operator to the sum over the deformable elements of the object from Equation (3.2), we can see that we need to consider the derivative with respect to only $3m$ out of the total $3n$ dimensions, while the remaining dimensions will be zero. However, since the elastic energy terms described for the solid and cloth models in Equations (3.5), (3.15) and (3.18) are complex functions, obtaining their analytic derivatives even with respect to these $3m$ dimensions is challenging. Rather then deriving the gradients and Hessians manually, we thus choose to compute them symbolically using the Maple computer algebra software by Maplesoft. We then export the resulting expressions as ANSI C source code and compile them directly into our software, providing efficient evaluation of the derivatives at runtime. As an alternative approach, automatic differentiation techniques could be used to achieve a similar effect.

3.2 Energy Minimization

3.2.1 Solver Considerations

The variational framework introduced in Section 3.1.4 allows us to look at the physical simulation problem of a deformable object without considering any specific details of the underlying physical laws or the discretization of the differential equations of motion anymore. In fact, we can now simulate physics even without understanding it—we can simply look at the objective function $H(\mathbf{x})$ from Equation (3.31) as a vector valued scalar function $H : \mathbb{R}^{3n} \to \mathbb{R}$ that we aim to minimize using any numerical algorithm.

Our method of choice will be the Newton-Raphson method, which has several appealing properties that make it a solid approach for the minimization of the objective function $H(\mathbf{x})$:

- It has quadratic convergence properties when provided with a reasonable initial guess, which is easy to achieve at least for dynamic problems where we can simply start with the solution from the previous time step.
- It makes use of the fact that the analytic formulation of the objective function $H(\mathbf{x})$ is known, and that we can thus derive formulas for the first and second derivative.
- Since it is an iterative method, it can be aborted early even when true convergence has not been reached yet, which we will make use

of often if the difference in geometry will not be visible in order not to waste computational resources.

The Newton-Raphson method also has a few shortcomings, all of which we will attempt to address in this section:

- It is not inherently robust and may overshoot the solution or even diverge. We will address this problem by using a line search scheme that adjusts the step size taken by our solver to make sure the value of the objective function decreases at an appropriate rate.
- Every iteration requires the solution to a linear system of equations with a size equal to the dimensions of $\mathbf{x}$, which can be very large in our case. We will alleviate this issue by exploiting the sparsity of the objective function's Hessian, which enables the use of a fast sparse Cholesky factorization solver.
- Even when using a sparse solver, finding the solution to the linear system can still pose a problem when the Hessian of the objective function is not positive definite. We will make use of a regularization scheme that is able to recover from indefinite Hessians.

Any other unconstrained optimization technique that applies to the objective function $H(\mathbf{x})$ could be used in place of the Newton-Raphson method. Alternatives include other gradient-based methods such as the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method, as well as derivative-free methods such as particle swarm optimization.

3.2.2 Newton-Raphson Iterations

In order to derive the fixed-point iteration scheme used by the Newton-Raphson method, we will start by stating the optimization problem as

$$\mathbf{x}^{t+1} = \arg\min_{\mathbf{x}} H(\mathbf{x}), \tag{3.32}$$

where $H(\mathbf{x})$ is defined as in Equation (3.31). We will refer to the solution $\mathbf{x}^{t+1}$ as *unknown state* or *next state*.

The necessary condition for a solution to Equation (3.32) is obtained by setting the gradient of the objective function to zero:

$$\frac{\partial}{\partial \mathbf{x}} H(\mathbf{x}) = \mathbf{0} \tag{3.33}$$

The gradient is a nonlinear function of the positions $\mathbf{x}$ that we assume to be too complex to invert. We therefore linearize it by constructing a first-order Taylor expansion around the known *current state* $\mathbf{x}^t$:

$$\left(\frac{\partial}{\partial \mathbf{x}} H(\mathbf{x})\right)^T \approx \underbrace{\left(\frac{\partial}{\partial \mathbf{x}} H\left(\mathbf{x}^t\right)\right)^T}_{\mathbf{g}} + \underbrace{\frac{\partial^2}{\partial \mathbf{x}^2} H\left(\mathbf{x}^t\right)}_{\mathbf{H}} \underbrace{\left(\mathbf{x} - \mathbf{x}^t\right)}_{\Delta \mathbf{x}} \tag{3.34}$$

We will also refer to the gradient $\mathbf{g}$ in its transposed form as *current gradient* and to $\mathbf{H}$ as *current Hessian*.

By using the condition from Equation (3.33) for the linear approximation of the gradient in Equation (3.34), we obtain the following linear system of equations for the unknown *search direction* $\Delta \mathbf{x}$:

$$\mathbf{H} \Delta \mathbf{x} = -\mathbf{g} \tag{3.35}$$

We can now formulate the Newton-Raphson fixed-point iteration scheme, where we will denote the k-th iteration's values by appending $[k]$:

$$\mathbf{x}[k+1] = \mathbf{x}[k] + \alpha[k] \cdot \Delta \mathbf{x}[k] \tag{3.36}$$

We initialize the iteration using the current state by setting $\mathbf{x}[0] = \mathbf{x}^t$, and we employ a line search strategy that we will describe in Section 3.2.5 to choose the step size $\alpha[k] \in [0, 1]$ in every iteration. We abort the iteration when the gradient norm $\|\mathbf{g}[k]\|_2$ falls below a certain threshold σ_{gradient}, or when a certain maximum number of iterations $k_{\max}$ is reached. We set $\sigma_{\text{gradient}} = 0.001$ and $k_{\max} = 20$ for all of our examples.

3.2.3 Linear System Solving

The main challenge in using the Newton-Raphson scheme lies in solving the linear system of equations (3.35) efficiently and robustly. Since the system matrix is the current Hessian $\mathbf{H}$ of the objective functions and has dimensions of $3n \times 3n$, the number of matrix entries grows quadratically with the number of vertices of the objects, which suggests that a solve might become intractable for high mesh resolutions. However, let us recall the definition of the objective function $H(\mathbf{x})$ from Equation (3.31): The individual energy potentials in $\mathcal{W}$ only operate on small subsets of vertices of the model rather than the whole state vector $\mathbf{x}$. This means that only Hessian entries whose respective vertex pairs share such a subset will be nonzero, and we can thus represent it as a sparse matrix. Further, since the structure of the deformable

model's elements is already known at the time of its creation, the sparsity of the Hessian will remain the same for the entire simulation.

Given that the sparsity of the Hessian is known and remains constant, we can precompute a symbolic Cholesky factorization of $\frac{\partial^2 H}{\partial \mathbf{x}^2}$ and use it to efficiently evaluate a lower triangular matrix $\mathbf{L}$ at runtime such that:

$$\mathbf{H} = \mathbf{L}\mathbf{L}^T \tag{3.37}$$

Substituting $\mathbf{y} = \mathbf{L}^T \Delta\mathbf{x}$, we first solve the following linear system for $\mathbf{y}$ by forward substitution:

$$\mathbf{L}\mathbf{y} = -\mathbf{g} \tag{3.38}$$

With $\mathbf{y}$ in hand, we can now solve for $\Delta\mathbf{x}$ by back substitution as follows:

$$\mathbf{L}^T \Delta\mathbf{x} = \mathbf{y} \tag{3.39}$$

We use the CHOLMOD algorithm from the SuiteSparse library [Chen et al., 2008] to perform these steps.

3.2.4 Hessian Regularization

The previously presented sparse Cholesky factorization solver scheme will only work if the current Hessian $\mathbf{H}$ is positive definite, since otherwise no lower triangular $\mathbf{L}$ satisfying the condition from Equation (3.37) exists. Unfortunately, this is not guaranteed to be the case for our objective function $H(\mathbf{x})$ if we are evaluating its second derivative too far away from the minimizer. Even if we were able to solve the system in Equation (3.35) for a Hessian $\mathbf{H}$ that is not positive definite, the resulting search direction $\Delta\mathbf{x}$ might not be a descending direction. To see this, let us look at the directional derivative of the objective function at the current state towards the search direction $\Delta\mathbf{x}$:

$$\nabla_{\Delta\mathbf{x}} H\left(\mathbf{x}^t\right) = \left(\frac{\partial}{\partial \mathbf{x}} H\left(\mathbf{x}^t\right)\right) \Delta\mathbf{x} = \mathbf{g}^T \Delta\mathbf{x} = -\Delta\mathbf{x}^T \mathbf{H} \Delta\mathbf{x} \tag{3.40}$$

If the Hessian $\mathbf{H}$ is positive definite, we can be sure that $\Delta\mathbf{x}$ will be a descending direction because the definition of positive definiteness states that

$$\forall \mathbf{x} : \mathbf{x}^T \mathbf{H} \mathbf{x} > 0. \tag{3.41}$$

To overcome this issue and ensure that our system matrix is always positive definite, we use the Hessian regularization technique described by Nocedal and Wright [2006] and add multiples of the identity matrix $\mathbf{I}_{3n}$ to $\mathbf{H}$

until $\mathbf{H} + \tau \cdot \mathbf{I}_{3n}$ becomes positive definite by successively increasing τ. We test for positive definiteness by running the Cholesky algorithm and checking if it terminates with success. Once it does, we use the decomposed Cholesky factor $\mathbf{L}$ corresponding to $\mathbf{H} + \tau \cdot \mathbf{I}_{3n}$ to solve for $\Delta\mathbf{x}$ using Equations (3.38) and (3.39). Since the resulting search direction $\Delta\mathbf{x}$ was not using the correct system matrix and thus does not satisfy the condition from Equation (3.35), the energy decrease of the Newton-Raphson iteration will typically be smaller and lose its quadratic convergence property. In practice, however, we noticed that only a few such steps have to be taken before $\mathbf{H}$ becomes positive definite again and the algorithm continues to converge quadratically from there on.

3.2.5 Line Search

We will now discuss how to choose the step size α used for the fixed-point iteration in Equation (3.36). In an ideal setting, we would always choose $\alpha = 1$ which guarantees quadratic convergence to the minimum of $H(\mathbf{x})$ when we are close enough to it. Since this is not always the case for a nonlinear objective function such as $H(\mathbf{x})$, we need to be careful to prevent overshooting and make sure that each step we take decreases its value sufficiently.

To this end, we follow Nocedal and Wright [2006] and consider the value of the objective function as a function of the step size α:

$$\phi(\alpha) = H(\mathbf{x}[k] + \alpha \cdot \Delta\mathbf{x}[k]) \tag{3.42}$$

Using the chain rule, we can easily compute the derivative of ϕ at the current state $\mathbf{x}[k]$ as

$$\phi'(0) = \mathbf{g}^T \Delta\mathbf{x}[k] \tag{3.43}$$

To exclude step sizes $\alpha[k]$ that insufficiently decrease the objective function from being chosen, we require the *Armijo condition* to hold for a small positive constant c:

$$\phi(\alpha[k]) \leq \phi(0) + c \cdot \alpha[k] \cdot \phi'(0) \tag{3.44}$$

We choose $c = 0.0001$ for all our examples.

We then look for a suitable $\alpha[k]$ as close as possible to 1 by following these steps as described by Nocedal and Wright [2006]:

1. Attempt a full step using $\alpha = 1$ and return it if it matches the condition of Equation (3.44).
2. Since we now know $\phi(0)$, $\phi'(0)$ and $\phi(1)$, we use them to interpolate a quadratic polynomial in α and compute its minimum α_{qmin}. If α_{qmin} satisfies Equation (3.44), we return it.

3. Since we now also know $\phi\left(\alpha_{\text{qmin}}\right)$, we interpolate a cubic polynomial in α and compute its minimum α_{cmin}. If α_{cmin} satisfies Equation (3.44), we return it.
4. We continue to do cubic interpolations as in the previous step by using $\phi(0)$, $\phi'(0)$ and the last two computed minima from previous polynomial interpolations.

An example implementation of this algorithm can be found in the work of Press and colleagues [2007].

3.3 Subspace Simulation

In the previous section, we described an efficient and practical approach to finding the minimizer of an objective function $H(\mathbf{x})$ that stems from an underlying physics problem. In this section, we will vary the problem slightly by assuming that the vertex positions $\mathbf{x}$ of the deformable model are not actually the unknown quantity, but rather the resulting deformation of a *rig* controlling the vertices of the object as specified by a vector of *rig parameters*. When minimizing the objective function in the rig's subspace with respect its parameters, we obtain a method that directly allows the generation of physical rig configurations. This approach—called *Rig-Space Physics*—was first described by Hahn and colleagues [2012] and enables interesting applications that allow artists to add physical secondary motions to rigged characters directly on top of existing keyframed animations. While we will outline the basic mathematical problem and its solution in this section, we will later see in Chapter 4 how it can be solved more efficiently for our *Efficient Rig-Space Physics Simulation* application. These optimizations will be reused to implement our interactive *Sketch Abstractions for Character Posing* system in Chapter 5. Even though this section uses the rig-space optimization problem to describe our subspace simulation framework, it also directly applies to arbitrary other subspaces, as we will later show in Chapter 6 in the context of our *Subspace Clothing Simulation* method.

3.3.1 Rig Spaces

We previously assumed that the vertices $\mathbf{x}$ of the deformable model are controlled by our simulation and thus have to be solved for. For the subspace simulation problem, we will now assume that the vertex positions $\mathbf{x}$ are a nonlinear function of r parameters $\mathbf{p} \in \mathbb{R}^r$ that we will call the *rig mapping*:

$$\mathbf{x} : \mathbb{R}^r \to \mathbb{R}^{3n} \tag{3.45}$$

We will call the domain of the rig mapping a *rig space*, as it represents a space of possible parameter configurations admitted by the rig. As before, we will use a lower-case $\mathbf{p}$ to denote *deformed* parameters, and assume that there exists an *undeformed* or *rest* parameter configuration denoted by capital $\mathbf{P}$ such that $\mathbf{x}(\mathbf{P}) = \mathbf{X}$.

We intentionally make very few assumptions about the structure of the rig mapping $\mathbf{x}(\mathbf{p})$ and only require that we have access to a black box that can evaluate it efficiently. In particular, this means that we do not necessarily have access to an analytic formula for $\mathbf{x}(\mathbf{p})$, and also might not be able to derive it symbolically. The reason for these weak requirements is that this allows the rig mapping to be provided by an external rigging system used by artists such as the ones found in Autodesk Maya or Blender. Since those software packages allow rigging artists to build rigs of virtual characters from a large selection of geometric deformers as well as to compose them into complex hierarchies, the resulting mappings $\mathbf{x}(\mathbf{p})$ vary greatly between rigs, their structure is hard to predict, and their analytic formulation is—at least in the case of proprietary systems—often not available. We will lift the black box rig assumption for the *Subspace Clothing Simulation* application in Chapter 6 where we will use a specially constructed and analytically known rig to speed up simulations.

3.3.2 Subspace Physics

Given a rig space and a matching rig mapping $\mathbf{x}(\mathbf{p})$, we will reformulate our original physical simulation problem and the objective function from Equation (3.31). Instead of looking for a minimizing deformation $\mathbf{x}^{t+1}$ for $H(\mathbf{x})$, we will now in each time step look for an optimal *next parameter configuration* $\mathbf{p}^{t+1}$ that minimizes the objective function in the rig space:

$$\mathbf{p}^{t+1} = \arg\min_{\mathbf{p}} H\left(\mathbf{x}(\mathbf{p})\right) \tag{3.46}$$

While the solution $\mathbf{p}^{t+1}$ to this minimization problem might not be optimal for $H(\mathbf{x})$ in the global sense, it attempts to find an optimal admissible pose of the rig that also minimizes the physical energy potentials as far as the rig space allows it. We will see in later chapters that this is a very powerful concept that enables many novel applications.

To solve the optimization problem from Equation (3.46), we reuse the same Newton-Raphson solver framework that we previously introduced in Section 3.2 with a few modifications. By simply reformulating the linearization

from Equation (3.34) to approximate the objective function in terms of the rig parameters $\mathbf{p}$, we obtain:

$$\left(\frac{\partial}{\partial \mathbf{p}} H\left(\mathbf{x}(\mathbf{p})\right)\right)^T \approx \underbrace{\left(\frac{\partial}{\partial \mathbf{p}} H\left(\mathbf{x}\left(\mathbf{p}^t\right)\right)\right)^T}_{\mathbf{r}} + \underbrace{\frac{\partial^2}{\partial \mathbf{p}^2} H\left(\mathbf{x}\left(\mathbf{p}^t\right)\right)}_{\mathbf{K}} \underbrace{\left(\mathbf{p} - \mathbf{p}^t\right)}_{\Delta \mathbf{p}} \tag{3.47}$$

As before, we will refer to the gradient $\mathbf{r}$ in its transposed form as the *current reduced gradient* and to $\mathbf{K}$ as *current reduced Hessian*. We can derive the reduced gradient $\mathbf{r}$ analytically using the transpose of the chain rule:

$$\mathbf{r} = \left(\frac{\partial}{\partial \mathbf{p}} H\left(\mathbf{x}\left(\mathbf{p}^t\right)\right)\right)^T = \underbrace{\left(\frac{\partial}{\partial \mathbf{p}} \mathbf{x}\left(\mathbf{p}^t\right)\right)^T}_{\mathbf{J}} \underbrace{\left(\frac{\partial}{\partial \mathbf{x}} H\left(\mathbf{x}\left(\mathbf{p}^t\right)\right)\right)^T}_{\mathbf{g}} = \mathbf{J}^T \mathbf{g} \tag{3.48}$$

We will call $\mathbf{J}$ the *rig Jacobian* as it describes the change in the deformation of the rigged vertices $\mathbf{x}$ with respect to changes in the rig parameters $\mathbf{p}$.

Analogously, the reduced Hessian $\mathbf{K}$ can be derived as

$$\begin{aligned} \mathbf{K} = \frac{\partial^2}{\partial \mathbf{p}^2} H\left(\mathbf{x}\left(\mathbf{p}^t\right)\right) = &\left(\underbrace{\frac{\partial}{\partial \mathbf{p}} \mathbf{x}\left(\mathbf{p}^t\right)}_{\mathbf{J}}\right)^T \underbrace{\frac{\partial^2}{\partial \mathbf{x}^2} H\left(\mathbf{x}\left(\mathbf{p}^t\right)\right)}_{\mathbf{H}} \underbrace{\frac{\partial}{\partial \mathbf{p}} \mathbf{x}\left(\mathbf{p}^t\right)}_{\mathbf{J}} \\ &+ \underbrace{\frac{\partial^2}{\partial \mathbf{p}^2} \mathbf{x}\left(\mathbf{p}^t\right)}_{\mathbf{T}} : \underbrace{\left(\frac{\partial}{\partial \mathbf{x}} H\left(\mathbf{x}\left(\mathbf{p}^t\right)\right)\right)^T}_{\mathbf{g}} \\ &= \mathbf{J}^T \mathbf{H} \mathbf{J} + \mathbf{T} : \mathbf{g}, \end{aligned} \tag{3.49}$$

where the operator : denotes a contraction between the rank-3 tensor $\mathbf{T} \in \mathbb{R}^{3n \times r \times r}$ and the current gradient $\mathbf{g}$ over the $\mathbf{x}$ dimension.

Setting the linear approximation from Equation (3.47) to zero again results in a linear system of equations for the unknown *reduced search direction* $\Delta \mathbf{p}$:

$$\mathbf{K} \Delta \mathbf{p} = -\mathbf{r} \tag{3.50}$$

Unlike the system in Equation (3.35) for the full-space physics problem, the system matrix in Equation (3.50) does not have any known sparsity structure: Assuming a black-box rig without a known analytical formulation, we have to estimate $\mathbf{J}$ and $\mathbf{T}$ using a dense finite differentiation scheme, resulting in a dense $\mathbf{K}$ even when projecting a sparse $\mathbf{H}$ in Equation (3.49). Using

central differences, we approximate the rig derivatives as

$$\begin{aligned}\mathbf{J}_{ij} &= \frac{1}{2\delta}\Big(\mathbf{x}\left(\mathbf{p}^t + \delta\cdot\mathbf{e}_j\right) - \mathbf{x}\left(\mathbf{p}^t - \delta\cdot\mathbf{e}_j\right)\Big)_i \\ \mathbf{T}_{ijk} &= \frac{1}{4\delta^2}\Big(\quad \mathbf{x}\left(\mathbf{p}^t + \delta\cdot\mathbf{e}_j + \delta\cdot\mathbf{e}_k\right) - \mathbf{x}\left(\mathbf{p}^t - \delta\cdot\mathbf{e}_j + \delta\cdot\mathbf{e}_k\right) \\ &\qquad - \mathbf{x}\left(\mathbf{p}^t + \delta\cdot\mathbf{e}_j - \delta\cdot\mathbf{e}_k\right) + \mathbf{x}\left(\mathbf{p}^t - \delta\cdot\mathbf{e}_j - \delta\cdot\mathbf{e}_k\right)\Big)_i,\end{aligned} \tag{3.51}$$

where $\mathbf{e}_j \in \mathbb{R}^r$ denotes a unit vector that is zero in all dimensions except the j-th in which it is one, and $\delta \in \mathbb{R}$ is the finite difference step size. We choose $\delta = 0.001$ for all of our rigs.

Our assumption is that the rig provides a high-level mechanism to deform the vertices of the deformable object, and thus that $r \ll n$. Since $\mathbf{K}$ is generally not sparse, we need to solve a small dense system. While the Cholesky decomposition approach described in Section 3.2.3 would also apply to Equation (3.50) if $\mathbf{K}$ is positive definite, we found that we can achieve better performance in the dense case when using a QR factorization approach. To this end, we decompose the reduced Hessian $\mathbf{K}$ into an orthogonal matrix $\mathbf{Q}$ and an upper triangular matrix $\mathbf{R}$ such that:

$$\mathbf{K} = \mathbf{Q}\mathbf{R} \tag{3.52}$$

Multiplying both sides of the linear system (3.50) by $\mathbf{Q}^T$ from the left and using Equation (3.52), we obtain

$$\mathbf{Q}^T\mathbf{K}\Delta\mathbf{p} = \underbrace{\mathbf{Q}^T\mathbf{Q}}_{\mathbf{I}_r}\mathbf{R}\Delta\mathbf{p} = \mathbf{R}\Delta\mathbf{p} = \mathbf{Q}^T\mathbf{r}, \tag{3.53}$$

where we made use of the orthogonality of $\mathbf{Q}$. Using this transformation, we can now easily solve for $\Delta\mathbf{p}$ by using back substitution on the triangular system matrix $\mathbf{R}$ with $\mathbf{Q}^T\mathbf{r}$ as the right-hand side. We use the LAPACK library routines to perform these operations.

As in the sparse case, we employ the Hessian regularization routine as described in Section 3.2.4 on $\mathbf{K}$ to ensure that $\Delta\mathbf{p}$ is a descending direction, and we use the linear search strategy outlined in Section 3.2.5 to prevent overshooting problems.

3.3.3 Rig-Space Physics Simulation

Given a rigged virtual character and a keyframed animation, Hahn and colleagues [2012] used the described subspace simulation method to add

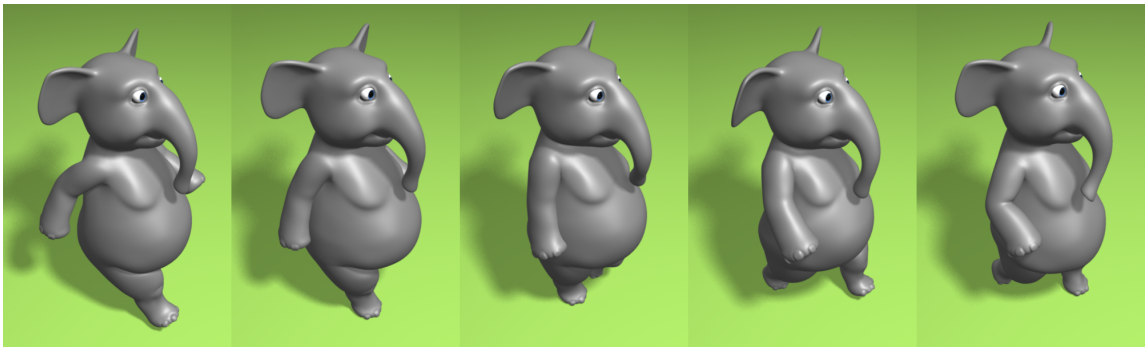

Figure 3.1: *Example result of the* Rig-Space Physics *simulation method: Given a character rig and a set of keyframes for some of its parameters, the method automatically produces animation curves for the remaining parameters.*

secondary motion effects to the animation by using the unkeyframed parameters of the character as rig mapping. Since the deformation of most artist-created rigs does not extend to the interior of the character and only deforms its surface, special care needs to be taken when modeling the character as an elastic solid using the model described in Section 3.1.2. Hahn and colleagues [2012] solve this problem by extending the rig mapping and pretending that there are $3n - 3n_{\mathbf{s}}$ additional rig parameters controlling the positions of the $n - n_{\mathbf{s}}$ interior vertices directly, where $n_{\mathbf{s}}$ denotes the number of surface vertices. An example result of this method applied to the virtual elephant character *Prof. Peanuts* can be seen in Figure 3.1.

While the additional degrees of freedom controlling the interior vertices successfully bridge the gap between the rigged surface and the volume of the deformable solid, they also introduce a large overhead to the method since they have to be solved for, thus significantly increasing the solver runtime. In Chapter 4 of this work, we will first provide a recap of the original rig-space physics system by Hahn and colleagues [2012] in Section 4.2.1, and then propose an extension to the simulation framework that obsoletes the solve for interior vertices—and thus achieves much improved performance.

3.4 Analytic Rigs

For all of the contributions and applications to be presented in this thesis, the key concepts are formed by looking at a rig as a space of potential deformations rather than a mere helper tool for artists. We specifically designed two of our methods to support *black-box* rigs for which we can only evaluate their deformation but have no access to their analytic formula. Treating a rig as a black box has the significant advantage of enabling our system to

work with professional production-grade rigs created in commercial animation packages such as Autodesk Maya. These rigging systems allow artists to select from a large variety of geometric deformers and rigging techniques, while also encouraging them to freely combine them into complex deformation hierarchies. Supporting arbitrary rigs also means that our methods are applicable to existing rigs, rather than rigs specifically designed to be used with our methods.

While the black-box rigging approach provides the user with the largest degree of flexibility, it also often comes at a significant cost when combined with simulation methods such as those presented in this work. All of our applications require some sort of derivative of the rig—either with respect to the rig parameters or the rest positions—which have to be estimated using numerical techniques. In the previous Section 3.3.2, we encountered an example of the former case in Equation (3.51) where both the Jacobian and the second-order tensor derivative of the rig had to be approximated using finite differences. Since finite difference estimations replace the exact evaluation of a derivative formula by repeated evaluations of the rig itself, their runtime performance directly depends on the evaluation time of the rig and the number of rigging controls. For complex rigs with many parameters, this causes the rig derivative approximation to quickly become the performance bottleneck of the whole system, and often impedes interactive use of our methods.

To circumvent this issue, we attempt to clone the whole character rig and its deformation behavior as created in Maya and rebuild it in our own system using an analytic formulation. We can then access the rig and its derivatives quickly during the simulation in our system, and later write the resulting rig parameters back to Maya once the solve is completed. While this is not possible for complex rigging hierarchies consisting of many different deformers—the mathematical expressions of which might even be undisclosed—there are two important exceptions where the problem is worth tackling:

- The de facto standard technique to build facial rigs is *blendshape modeling*, for which the analytic formulation is straightforward.
- At its base level, rigging artists almost always start with a skeletal hierarchy of bones when designing a character rig. The character's surface is then bound to the skeleton using *linear blend skinning* (LBS), the formulation for which is well known.

In the following subsections, we will derive the necessary derivatives for

these two techniques and describe how the implementations can be designed to exactly match Maya's deformation behavior.

3.4.1 Rigid Transformation

As an introductory example, we will look at a simplistic rig that performs only rigid transformations of an object before looking at the more complicated cases of blendshapes and linear blend skinning. The rigid transformation rig applies the same transformation to all vertices of the undeformed object, exposing six rig parameters t_x, t_y, t_z, r_x, r_y, $r_z \in \mathbb{R}$ for translation and rotation with respect to the three coordinate axes. Its parameter vector is thus given as:

$$\mathbf{p} = (t_x, t_y, t_z, r_x, r_y, r_z) \in \mathbb{R}^6 \tag{3.54}$$

Its analytic rig mapping for a vertex $\mathbf{x}_i$ is given as

$$\mathbf{x}_i(\mathbf{p}) = \mathbf{R}_z(r_z)\mathbf{R}_y(r_y)\mathbf{R}_x(r_x)\mathbf{X}_i + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}, \tag{3.55}$$

where $\mathbf{X}_i$ denotes the vertex in its undeformed state. The matrices $\mathbf{R}_x, \mathbf{R}_y, \mathbf{R}_z \in \mathbb{R}^{3\times3}$ in Equation (3.55) denote the standard 3D rotation matrices with respect to the three unit axes.

In this work, we are interested in two different derivatives with respect to a rig mapping such as the one in Equation (3.55). The first is the Jacobian of the rig's deformed vertices $\mathbf{x}$ with respect to the rig parameter vector $\mathbf{p}$, which is required to perform rig-space simulations as described in Section 3.3.2 as well as in Chapters 4 and 5. We will typically denote evaluations of this *rig Jacobian* at specific pose configurations by $\mathbf{J}$. The rig derivative of the rigid transformation rig with respect to the translational parameters is given as:

$$\frac{\partial}{\partial t_x}\mathbf{x}_i(\mathbf{p}) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad \frac{\partial}{\partial t_y}\mathbf{x}_i(\mathbf{p}) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \qquad \frac{\partial}{\partial t_z}\mathbf{x}_i(\mathbf{p}) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.56}$$

The rig derivative of the rigid transformation rig with respect to the rotational parameter r_x is given as:

$$\begin{aligned} \frac{\partial}{\partial r_x}\mathbf{x}_i(\mathbf{p}) &= \mathbf{R}_z(r_z)\mathbf{R}_y(r_y)\frac{\partial}{\partial r_x}\mathbf{R}_x(r_x)\mathbf{X}_i \\ \frac{\partial}{\partial r_y}\mathbf{x}_i(\mathbf{p}) &= \mathbf{R}_z(r_z)\frac{\partial}{\partial r_y}\mathbf{R}_y(r_y)\mathbf{R}_x(r_x)\mathbf{X}_i \\ \frac{\partial}{\partial r_z}\mathbf{x}_i(\mathbf{p}) &= \frac{\partial}{\partial r_z}\mathbf{R}_z(r_z)\mathbf{R}_y(r_y)\mathbf{R}_x(r_x)\mathbf{X}_i \end{aligned} \tag{3.57}$$

Both the translational components from Equation (3.56) and the rotational components from Equation (3.57) for all vertices can then be assembled to the full Jacobian matrix $\frac{\partial \mathbf{x}}{\partial \mathbf{p}} \in \mathbb{R}^{n \times r}$.

For the *Subspace Clothing Simulation* application we will describe in Chapter 6, we will use the rig mapping as a kinematic reference rather than a subspace deformation. To this end, we will require the derivative of its deformed vertices $\mathbf{x}$ with respect to the undeformed state $\mathbf{X}$, and will typically denote evaluations of this *rig rest state Jacobian* by $\mathbf{B}$. In the case of the rigid transformation rig, Equation (3.55) is linear in the undeformed positions $\mathbf{X}$ and only accesses the undeformed vertex with the same index as the deformed vertex for which it computes the deformation. Hence, the rest state Jacobian is a block diagonal matrix $\frac{\partial \mathbf{x}}{\partial \mathbf{X}} \in \mathbb{R}^{3n \times 3n}$ consisting of n blocks $\mathbf{B}_i \in \mathbb{R}^{3 \times 3}$ given as:

$$\mathbf{B}_i = \frac{\partial}{\partial \mathbf{X}_i} \mathbf{x}_i(\mathbf{p}) = \mathbf{R}_z(r_z)\mathbf{R}_y(r_y)\mathbf{R}_x(r_x) \tag{3.58}$$

3.4.2 Blendshapes

The blendshape technique is typically used to build facial rigs as it allows to specify poses as sparse offsets to a neutral pose, mimicking the activations of facial muscles in the human body. Given r poses $\mathbf{R}^1, \mathbf{R}^2, \ldots, \mathbf{R}^r \in \mathbb{R}^{3n}$, the parameter vector $\mathbf{p} \in \mathbb{R}^r$ of a blendshape rig comprises a real valued activation level for each pose. Its deformation is defined as:

$$\mathbf{x}(\mathbf{p}) = \mathbf{X} + \sum_{i=1}^{r} (\mathbf{p})_i \cdot \left(\mathbf{R}^i - \mathbf{X} \right) \tag{3.59}$$

The Jacobian of a blendshape rig with respect to one of the activation parameters is easily obtained as:

$$\frac{\partial}{\partial \mathbf{p}_i} \mathbf{x}(\mathbf{p}) = \mathbf{R}^i - \mathbf{X} \tag{3.60}$$

Similarly, the rest state Jacobian of the blendshape rig can be computed as

$$\frac{\partial}{\partial \mathbf{X}} \mathbf{x}(\mathbf{p}) = \left(1 - \sum_{i=1}^{r} (\mathbf{p})_i \right) \cdot \mathbf{I}_{3n}, \tag{3.61}$$

where $\mathbf{I}_{3n}$ denotes an identity matrix of size $3n \times 3n$.

3.4.3 Linear Blend Skinning

Definition

Linear blend skinning is a widely used technique for binding the surface of a character to a skeletal hierarchy of bones. While it has some shortcomings such as the "candy wrapper" and "collapsing elbow" artifacts, it remains a popular choice for characters rigs in both movie and games production because of its intuitive controls and wide availability in animation toolkits and game engines. The core idea of the technique is to express the transformation of each joint of a skeleton as a recursive function based on the transformations of its parent joint, to which it is connected to by a bone. Every point on a character's surface mesh is then expressed as a weighted combination of the character's corresponding rest position point transformed into the coordinate frames of multiple joints. Formally, the linear blend skinning transformation of a vertex $\mathbf{x}_i$ using a rig with m joints is given as

$$\mathbf{x}_i(\mathbf{p}) = \begin{bmatrix}\mathbf{I}_3 & \mathbf{0}\end{bmatrix} \sum_{j=1}^{m} (\mathbf{W})_{ij} \cdot \mathbf{C}_j(\mathbf{p}) \hat{\mathbf{X}}_i, \tag{3.62}$$

where the joint transformations $\mathbf{C}_j \in \mathbb{R}^{4\times4}$ transforming the undeformed positions $\hat{\mathbf{X}}_i \in \mathbb{R}^4$ lifted to homogeneous space are controlled by joint angles and bone scales exposed as rig parameters in $\mathbf{p}$. The matrix $\mathbf{W} \in \mathbb{R}^{n\times m}$ in Equation (3.62) stores the *skinning weights* for each vertex of the character surface with respect to all of the joints of the skeleton. Since it is desirable for skinned vertices to only have the support of a few nearby joints, $\mathbf{W}$ is typically very sparse. Rather than performing a true unprojection from homogeneous space in Equation (3.62), we simply discard the fourth dimension by multiplying with $\begin{bmatrix}\mathbf{I}_3 & \mathbf{0}\end{bmatrix} \in \mathbb{R}^{3\times4}$ after the skinning transformations. Since we only use the fourth dimension to express translations as transformation matrices, the multiplication with this matrix is equivalent to the unprojection in this case while greatly simplifying the following derivations of the rig Jacobians.

Joint Transformations

The matrices $\mathbf{C}_j$ first transform the undeformed vertices $\hat{\mathbf{X}}$ into the respective joint's undeformed coordinate frame corresponding to the neutral rig pose $\mathbf{p}^0$, which we call *forward transformation*. The undeformed vertices in local coordinates are then transformed back to world space according to the

current pose $\mathbf{p}$ of the skeleton, which we refer to as *backward transformation*. The transformation of the j-th joint is thus given as

$$\mathbf{C}_j(\mathbf{p}) = \mathbf{C}_j^{\text{back}}(\mathbf{p})\mathbf{C}_j^{\text{for}}, \tag{3.63}$$

where only the backward transformation $\mathbf{C}_j^{\text{back}}$ is a function of the rig parameters $\mathbf{p}$, and the forward transformation $\mathbf{C}_j^{\text{for}}$ only depends on $\mathbf{p}^0$.

Before we can state the forward and backward transformations in Equation (3.63), we need to further distinguish between local *base* and *pose* transformations. The former refer to local translation, scaling and rotation transformations to the parent joint's coordinate system already present in the rig's neutral pose $\mathbf{p}^0$, whereas the latter refer to scaling and rotation offsets caused by the deformed rig parameters $\mathbf{p}$. The forward transformation of a joint is defined by only its base transformations, and is recursively given as

$$\mathbf{C}_j^{\text{for}} = \left(\mathbf{S}_j^{\text{base}}\right)^{-1}\left(\mathbf{R}_j^{\text{base}}\right)^{-1}\left(\mathbf{T}_j^{\text{base}}\right)^{-1}\mathbf{C}_{\text{parent}(j)}^{\text{for}}, \tag{3.64}$$

where $\text{parent}(j)$ denotes the index of the parent joint of the j-th joint and we define the forward parent transformation of a root joint to be $\mathbf{C}_{\text{parent(root)}}^{\text{for}} = \mathbf{I}_4$ as the recursion's base case. The transformations $\mathbf{R}_j^{\text{base}}$, $\mathbf{T}_j^{\text{base}}$ and $\mathbf{S}_j^{\text{base}}$ in Equation (3.64) are local rotation, translation and scaling matrices of the joint in its neutral pose $\mathbf{p}^0$. The backward transformation of a joint is defined by both its base transformations and its pose transformations, and is recursively given as

$$\mathbf{C}_j^{\text{back}}(\mathbf{p}) = \mathbf{C}_{\text{parent}(j)}^{\text{back}}(\mathbf{p})\mathbf{T}_j^{\text{base}}\mathbf{R}_j^{\text{base}}\mathbf{S}_j^{\text{base}}\mathbf{R}_j^{\text{pose}}(\mathbf{p})\mathbf{S}_j^{\text{pose}}(\mathbf{p}), \tag{3.65}$$

where we define the backward parent transformation of a root joint to be $\mathbf{C}_{\text{parent(root)}}^{\text{back}}(\mathbf{p}) = \mathbf{I}_4$ as the recursion's base case. The transformations $\mathbf{R}_j^{\text{pose}}(\mathbf{p})$ and $\mathbf{S}_j^{\text{pose}}(\mathbf{p})$ in Equation (3.65) are local rotation and scaling matrices of the joint in its current pose $\mathbf{p}$.

Rig Jacobians

Let us now look at the Jacobian of the deformed positions $\mathbf{x}$ with respect to the rig parameters $\mathbf{p}$ for a linear blend skinning rig. Deriving Equation (3.62) with respect to a single parameter $\mathbf{p}_k$, we obtain:

$$\frac{\partial}{\partial \mathbf{p}_k}\mathbf{x}_i(\mathbf{p}) = \begin{bmatrix}\mathbf{I}_3 & \mathbf{0}\end{bmatrix}\sum_{j=1}^{m}(\mathbf{W})_{ij}\cdot\frac{\partial}{\partial \mathbf{p}_k}\mathbf{C}_j(\mathbf{p})\hat{\mathbf{X}}_i \tag{3.66}$$

Since the forward transformations $\mathbf{C}_j^{\text{for}}$ are not functions of $\mathbf{p}$, we only need to derive the backward transformations to compute the joint transformation derivatives in Equation (3.66):

$$\frac{\partial}{\partial \mathbf{p}_k}\mathbf{C}_j(\mathbf{p}) = \frac{\partial}{\partial \mathbf{p}_k}\mathbf{C}_j^{\text{back}}(\mathbf{p})\mathbf{C}_j^{\text{for}} \tag{3.67}$$

At first glance, computing the backward joint transformation derivative in Equation (3.67) seems intractable since expanding $\mathbf{C}_j^{\text{back}}(\mathbf{p})$ using Equation (3.65) results in a large transformation chain of many factors that individually depend on the rig parameter vector $\mathbf{p}$. Applying the chain rule to such a transformation chain would lead to a combinatorical explosion of subproducts which we would be unable to handle. However, closer inspection reveals that each rig parameter $\mathbf{p}_k$ only affects a single local transformation matrix of its respective joint in the transformation chain. Further, the derivative is zero if a rig parameter $\mathbf{p}_k$ refers to a joint that is not in the chain from the root of the skeleton to the j-th joint. For a specific rig parameter $\mathbf{p}_k$, we can thus rewrite Equation (3.65) as

$$\mathbf{C}_j^{\text{back}}(\mathbf{p}_k) = \begin{cases} \mathbf{F}^{\text{pre}}\mathbf{L}(\mathbf{p}_k)\mathbf{F}^{\text{post}} & \text{if joint of } \mathbf{p}_k \text{ is in chain of joint } j \\ \mathbf{0} & \text{else} \end{cases}, \tag{3.68}$$

for some constant prefix and postfix transformations $\mathbf{F}^{\text{pre}}$ and $\mathbf{F}^{\text{post}}$ independent of $\mathbf{p}_k$, and where $\mathbf{L}(\mathbf{p}_k)$ is either a local pose rotation or scaling matrix depending on $\mathbf{p}_k$. In the first case of Equation (3.68), the backward joint transformation derivative we need to compute for Equation (3.67) is now easily obtained as:

$$\frac{\partial}{\partial \mathbf{p}_k}\mathbf{C}_j^{\text{back}}(\mathbf{p}) = \mathbf{F}^{\text{pre}}(\mathbf{p})\frac{\partial}{\partial \mathbf{p}_k}\mathbf{L}(\mathbf{p}_k)\mathbf{F}^{\text{post}}(\mathbf{p}) \tag{3.69}$$

Luckily, the computation of the rest state derivative of a linear blend skinning rig is much more straightforward than its rig parameter derivative. Since Equation (3.62) is linear in the undeformed positions $\mathbf{X}$ and only accesses the undeformed vertex with the same index as the deformed vertex for which it computes the deformation, the structure of the rest state Jacobian $\frac{\partial \mathbf{x}}{\partial \mathbf{X}} \in \mathbb{R}^{3n\times 3n}$ of a linear blend skinning rig is the same as with the rigid transformation rig we have seen in Section 3.4.1. It is given as a block diagonal matrix with n blocks $\mathbf{B}_i \in \mathbb{R}^{3\times 3}$ defined as:

$$\mathbf{B}_i(\mathbf{p}) = \frac{\partial}{\partial \mathbf{X}_i}\mathbf{x}_i(\mathbf{p}) = \begin{bmatrix}\mathbf{I}_3 & \mathbf{0}\end{bmatrix}\sum_{j=1}^{m}(\mathbf{W})_{ij}\cdot\mathbf{C}_j(\mathbf{p}) \tag{3.70}$$

Matching Maya Behavior

The presented formulation of an analytic linear blend skinning rig differs slightly from the one present in Autodesk Maya in that Maya joints do not explicitly distinguish between base and pose transformations, and include some additional factors. Concretely, the *Maya C++ API Reference* defines the transformation of a joint as follows: [Autodesk, 2015]

```
  matrix = [S] * [RO] * [R] * [JO] * [IS] * [T]
(where '*' denotes matrix multiplication).
These matrices are defined as follows:
  [S] : scale
  [RO] : rotateOrient (attribute name is rotateAxis)
  [R] : rotate
  [JO] : jointOrient
  [IS] : parentScaleInverse
  [T] : translate
```

Since Maya multiplies points to be transformed from the left rather than from the right as in our case in Equation (3.62), we can imitate Maya's joint hierarchy behavior by assigning for each joint:

$$\begin{aligned} \mathbf{T}^{\text{base}} &\leftarrow \texttt{[T]} \\ \mathbf{R}^{\text{base}} &\leftarrow \texttt{[JO]} \\ \mathbf{R}^{\text{pose}} &\leftarrow \texttt{[R]} \\ \mathbf{S}^{\text{pose}} &\leftarrow \texttt{[S]} \end{aligned} \tag{3.71}$$

The assignment from Equation (3.71) will replicate the Maya behavior exactly as long as the following assumptions hold:

- We assume `[RO]` to be an identity matrix, which we found to be the case for all of the artist-created rigs we used.
- We assume `[IS]` to be an identity matrix, which is the case if the *Segment Scale Compensate* option for the joint is disabled. Even though this attribute is enabled by default for newly created joints in Maya, supporting it would greatly increase the complexity of the rig's deformation and derivative complexity, since it makes scaling values of a joint also affect the transformations of its parents.
- We assume `[T]` not to be part of the pose and the animation of the character. This is rarely a problem since Maya does not properly support runtime translation of joints anyway if skin is attached to them.

3.5 Contact Handling

For two of the applications presented later in this work—the *Efficient Rig-Space Physics Simulation* and the *Subspace Clothing Simulation* methods to be described in Chapters 4 and 6—we will be interested in simulating physical contact and collisions plausibly. While contact handling is not a core requirement of these systems, it is nevertheless a crucial ingredient to create compelling results that showcase the strengths and applicability of the methods. In fact, even supposedly trivial examples involving the simulation of deformable objects may result in contact. When simulating a cylindrical tube under the effects of gravity that is attached at one central point, the two sides of the tube are likely to collide with each other when falling down under the influence of the gravitational forces. Clothing represents another example where contact occurs on a frequent basis, since it typically collides with the skin of the character wearing it, as well as with itself in the case of wrinkling and folding.

We will distinguish two kinds of contact and will address them separately because of their particular effect on the simulation of a deformable object:

- *External* contact occurs between a simulated deformable object and another entity present in the virtual environment, which is typically another animated object or a ground plane. For this case, we will make the simplifying assumption that the motion of the external object is predefined and is not affected by the contact event. In other words, dropping a simulated rubber ball onto the ground will cause the ball to deform, but not the ground. We will also only consider cases where a vertex of the deformable object crosses the surface boundary of a triangle of an external object.
- *Internal* contact occurs between some part of a simulated deformable object and another part of the same object. For solid objects, we consider cases where a vertex penetrates the volume of a tetrahedral element that it is not part of. For cloth, we consider the cases where a vertex or an edge crosses the surface boundary of a triangular element that it is not part of.

3.5.1 Collision Detection

As described before, we take a primitive-based approach to the detection and resolution of contact, and break events down to individual groups of vertices or edges rather than computing an exact collision volume. Thus,

the objective of the collision detection step is to consider the current state of all objects in the simulated environment as input and produce a list of vertex-tetrahedron, vertex-triangle or edge-edge pairs that are either in close proximity or already colliding. A straightforward way to achieve this would be to enumerate all possible combinations of vertices and triangles as well as edge pairs and check whether their distance falls below a certain threshold. However, this approach is not very efficient because of the quickly exploding number of combinations especially for high-resolution objects.

Instead of a brute force search, we opt to use a specialized spatial data structure to prevent unnecessary distance checks. One special requirement is that the data structure needs to be able to be efficiently updated since we are using it to store deforming geometry. We use the well-known *bounding volume hierarchy* (BVH) approach with sphere volumes to efficiently track our geometry and test it for collisions. We start the construction of a BVH tree for an object by creating a single enclosing volume containing all of the triangles or tetrahedrons, which we will refer to as elements. The volume is then recursively split into subvolumes by using the rule described in Algorithm 1.

Algorithm 1: *Recursive volume splitting algorithm for the BVH tree construction.*

```
1:  function SplitVolume(v)
2:      compute centroids of all elements in v
3:      determine dimension with maximum distance between centroids
4:      sort elements by their centroids in determined dimension
5:      create two child volumes v_left and v_right
6:      assign first half of sorted elements to v_left
7:      resize v_left to fit its elements
8:      if number of elements in v_left > 1 then
9:          SplitVolume(v_left)
10:         add v_left as child node of v
11:     end if
12:     assign second half of sorted elements to v_right
13:     resize v_right to fit its elements
14:     if number of elements in v_right > 1 then
15:         SplitVolume(v_right)
16:         add v_right as child node of v
17:     end if
18: end function
```

At runtime, the BVH tree can be efficiently updated by recursively resizing the sphere volumes according to the updated positions of the corresponding object. Starting with the root volume, we update the data structure by

resizing the bounding spheres bottom-up using the routine outlined in Algorithm 2.

Algorithm 2: *Recursive volume update algorithm for BVH trees.*

```
1: function UPDATEVOLUME(v)
2:     if v has child nodes v_left and v_right then
3:         UPDATEVOLUME(v_left)
4:         UPDATEVOLUME(v_right)
5:         resize v to fit v_left and v_right
6:     else
7:         resize v to fit its elements using new vertex positions
8:     end if
9: end function
```

To detect proximity events between two objects, we recursively collide their bounding volume hierarchies starting with their root volumes. Whenever we encounter two colliding volumes that represent inner nodes of the tree, we split up the larger one and recursively collide the child volumes with the smaller one. Once we reach two volumes that do not enclose child volumes and thus represent leaves of the BVH tree, we report a proximity event for the volume pair. The recursive collision routine is described in Algorithm 3. In order to detect internal collisions of an object, the collision routine is called to collide the root volume of the object with itself.

Once all proximity events for volume pairs have been collected, we divide them each further into the three different *primitive proximity event* types that we handle:

- If the two volumes enclose two triangles, we combine each of the vertices with the triangle they are not part of to generate 6 vertex-triangle primitive proximity events.
- If the two volumes enclose two triangles, we combine each of the edges with each edge of the triangle they are not part of to generate 18 edge-edge primitive proximity events.
- If the two volumes enclose two tetrahedrons, we combine each of the vertices with the tetrahedron they are not part of to generate 8 vertex-tetrahedron primitive proximity events.

Algorithm 3: *Recursive collision algorithm for two BVH tree volumes.*

```
1:  function CollideVolumes(v, w)
2:      if volumes v and w collide then
3:          if v has child nodes v_left and v_right then
4:              if w has child nodes w_left and w_right then
5:                  if v is larger than w then
6:                      CollideVolumes(v_left, w)
7:                      CollideVolumes(v_right, w)
8:                  else
9:                      CollideVolumes(v, w_left)
10:                     CollideVolumes(v, w_right)
11:                 end if
12:             else
13:                 CollideVolumes(v_left, w)
14:                 CollideVolumes(v_right, w)
15:             end if
16:         else
17:             if w has child nodes w_left and w_right then
18:                 CollideVolumes(v, w_left)
19:                 CollideVolumes(v, w_right)
20:             else
21:                 report proximity event for (v, w)
22:             end if
23:         end if
24:     end if
25: end function
```

3.5.2 Penalty-Based Collision Resolution

The first approach we will describe to handle detected collisions is based on a penalty-based energy potential that we add to our physical objective function $H(\mathbf{x})$. A simple form of such a penalty potential suitable for external contact was already introduced earlier in Section 3.1.3. The general idea is to allow small penetrations between objects, but to then introduce a force proportional to the penetration depth that counteracts this contact. We model these forces as spring potentials with zero rest length, which in its simplest form is given by the W_{contact} potential from Equation (3.22). Denoting the vertices of the i-th triangle in contact by $\mathbf{t}_1, \mathbf{t}_2$ and $\mathbf{t}_3$, we compute its contact normal as:

$$\mathbf{n}_i^p = \frac{(\mathbf{t}_2 - \mathbf{t}_1) \times (\mathbf{t}_3 - \mathbf{t}_1)}{\|(\mathbf{t}_2 - \mathbf{t}_1) \times (\mathbf{t}_3 - \mathbf{t}_1)\|_2} \tag{3.72}$$

In the tetrahedral case, we instead use the three vertices of the closest surface triangle to the i-th tetrahedron in contact.

In order to install a contact spring, we need to find the closest point $\mathbf{s}_i^p$ on the surface triangle plane to the penetrating vertex $\mathbf{x}_i$. To this end, we begin by expressing $\mathbf{s}_i^p$ in terms of the triangle vertices using barycentric coordinates:

$$\mathbf{s}_i^p = \lambda_1 \cdot \mathbf{t}_1 + \lambda_2 \cdot \mathbf{t}_2 + (1 - \lambda_1 - \lambda_2) \cdot \mathbf{t}_3 \tag{3.73}$$

Ideally, we would have $\mathbf{s}_i^p = \mathbf{x}_i$, and since the barycentric coordinates λ_1 and λ_2 are unknown, we can reorder Equation (3.73) to obtain the overconstrained linear system

$$\left[\mathbf{t}_1 - \mathbf{t}_3 \;\middle|\; \mathbf{t}_2 - \mathbf{t}_3\right] \begin{bmatrix}\lambda_1 \\ \lambda_2\end{bmatrix} = \left[\mathbf{x}_i - \mathbf{t}_3\right], \tag{3.74}$$

which we solve in a least-squares sense by performing a direct solve on the corresponding normal equations. The desired closest surface point $\mathbf{s}_i^p$ is then obtained by plugging the optimized barycentric coordinates from Equation (3.74) back into Equation (3.73).

Once the barycentric coordinates λ_1 and λ_2 are known, expressing $\mathbf{s}_i^p$ in terms of the triangle vertices also provides us with a suitable way of handling internal contact where the surface triangle vertices $\mathbf{t}_1$, $\mathbf{t}_2$ and $\mathbf{t}_3$ are themselves part of the deformable object's state $\mathbf{x}$. Following Barbič and James [2008], we only compute the normal $\mathbf{n}_i$ and the barycentric coordinates λ_1 and λ_2 once as we detect the collision, but then express $\mathbf{s}_i^p$ in terms of the varying unknown vertex positions corresponding to the surface triangle corners. This approximative approach avoids degeneracies that can arise for more physically accurate penalty models while still producing plausible results in practice. The downside of this method is that the potential $W_{\text{contact}}(\mathbf{x})$ now refers to four different vertices of the deformable objects, which results in entries to Hessian locations that were previously assumed to be sparse. This forces us to recompute the sparsity structure of $\mathbf{H}$ and recompute the symbolic Cholesky factorization from Equation (3.37), incurring a small loss in performance.

3.5.3 Impulse-Based Contact Resolution

While the previously described penalty force approach works well for deformable solids, it is generally not applicable to cloth simulation. Even if a cloth surface is oriented and we can distinguish a ”left” side from a ”right”

side, cloth is able to fold over itself and thus internal contact can occur between all combinations of sides: left on left, left on right, and right on right. Since we only keep track of the current and the previous cloth state, allowing even small penetrations would cause problems in future time steps where we would be unable to distinguish between a contact about to occur and a contact that already occurred but was not fully resolved yet.

To alleviate this issue, we do not make use of the penalty-based contact resolution method and use an impulse-based scheme instead, which is based on the premise that we will never allow any collisions to occur over the course of a simulated time step. The simulation framework for a single time step is adapted to follow these steps:

1. We perform a regular simulation of the time step by minimizing our physical objective function $H(\mathbf{x})$ to obtain $\mathbf{x}^{t+1}$, potentially introducing penetrations in contact cases.
2. We detect all collisions introduced in $\mathbf{x}^{t+1}$.
3. We *filter* $\mathbf{x}^{t+1}$ by applying impulses to colliding vertices that prevent the collisions from occurring.
4. We continue with the simulation of the next time step using the filtered $\mathbf{x}^{t+1}$ as current state.

In practice, we implement the third step using the approach described by Bridson and colleagues [2002] and compute the relative velocity between each vertex-triangle and edge-edge primitive proximity event pair. If the relative velocity is large enough enough to cause a collision during the current time step, we apply a stopping impulse that is distributed equally to all four vertices participating in the contact. We only apply the full impulse magnitude in the principal direction of contact, and scale the residual tangential velocities by a factor $\beta \in [0, 1]$, which provides the user with control over the amount of exerted friction during the contact.

While this simple technique is very effective at resolving most of the contact cases arising in practice, it provides no theoretical guarantees that would allow us to always maintain the invariant that the cloth mesh should be collision-free. The reason for this is that we only resolve penetrations per primitive, but the stopping impulse we apply might very well cause additional collisions of the involved vertices with other faces or edges. Bridson and colleagues [2002] note that even iterating the filtering step does not always resolve all collisions and may become stuck in cycles. We thus recheck for collisions after the first filtering iteration, and employ the surface tracking routines from the ElTopo library by Brochu and Bridson [2009] if any

penetrations remain. While the surface tracking routine is much less efficient than the primitive-based impulse filtering, it is guaranteed to always return a collision-free mesh for arbitrarily complex cases. We noticed that our solver only invokes these routines in rare cases of heavily interlocked contact, which on average only results in a small performance penalty.

CHAPTER 4

Efficient Rig-Space Physics Simulation

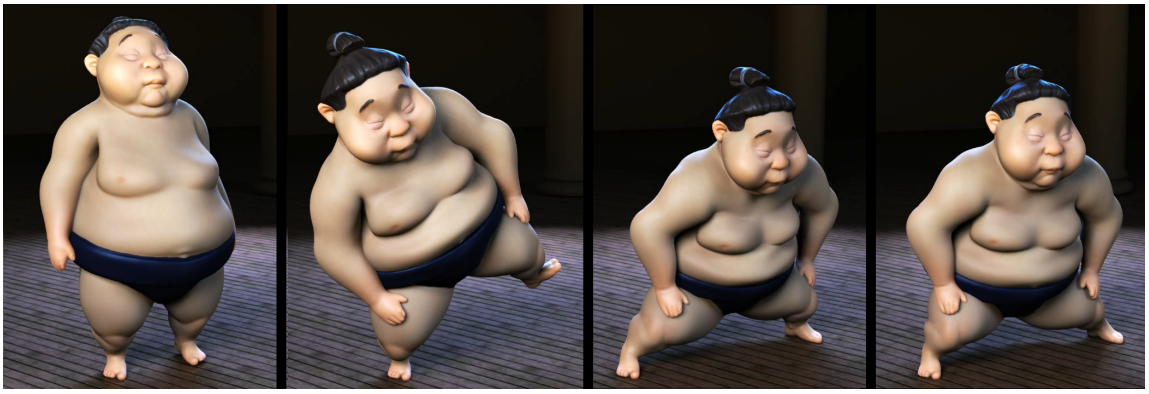

Figure 4.1: *Our method augments hand-crafted character animations such as this sumo wrestler with high-quality secondary motion, using an efficient rig-space simulation method.*

4.1 Overview

In Section 3.3.3, we described the *Rig-Space Physics* method presented by Hahn and colleagues [2012] that allows artists to add secondary motion effects to an existing rigged virtual character directly by automatically

keyframing rig parameters not used by the artist. While the method successfully bridges the gap between the traditional keyframe animation paradigm digital artists are used to and the world of physics-based simulation, its advantages come at a heavy computational price. Since all the interior vertices of the simulated elastic solid are treated as additional degrees of freedom aside from the rig parameters, the size of the linear system to compute the Newton search direction directly scales with the resolution of the tetrahedral mesh. Furthermore, the subspace system matrix is dense, which makes the problem even more difficult to solve. Even though Hahn and colleagues [2012] address this sparsity problem and propose the use of a specialized Schur complement decomposition strategy for the resulting reduced Hessian, the number of degrees of freedom remains one of the bottlenecks of their system.

In this chapter, we present a novel method that offers a significant computational improvement over the work of Hahn and colleagues [2012] while maintaining the same level of quality. This advance is made possible by three main contributions:

- a linearized formulation of rig-space dynamics using rig parameters as the only degrees of freedom,
- a physics-based volumetric skinning method that allows our algorithm to compute the position of internal vertices solely from the surface vertices, and
- a deferred Jacobian evaluation scheme that significantly reduces the number of required rig evaluations.

Taken together, these contributions allow a performance improvement of one to two orders of magnitude over the original rig-space physics method on production-quality rigs, as shown in Figure 4.1.

4.2 Method

Our method comprises three novel core contributions which we will introduce and describe in detail in this section. Before we do so, we provide a short technical recap on the original rig-space physics solver by Hahn and colleagues [2012] to highlight its shortcomings. We will then propose our extensions to the method, as well as provide the necessary derivative formulas required to reimplement our system and reproduce our results.

4.2.1 Rig-Space Physics Recap

Let us recall the generic rig-space energy minimization framework we presented earlier in Section 3.3. At its core, we formalized the minimization problem as a minimization of the objective function $H(\mathbf{x})$ from Equation (3.31) over the unknown rig parameters $\mathbf{p}$ as described in Equation (3.46). However, since artist-designed rigs of virtual characters in practice only deform the $n_{\mathbf{s}}$ surface points $\{\mathbf{s}_i\} \subset \{\mathbf{x}_i\}$ but not the $n_{\mathbf{n}}$ internal nodes $\{\mathbf{n}_i\} = \{\mathbf{x}_i\} \setminus \{\mathbf{s}_i\}$, the rig mapping is defined as $\mathbf{s}(\mathbf{p})$ rather than as $\mathbf{x}(\mathbf{p})$. To bridge this gap between the surface rig and the vertices of the tetrahedral simulation mesh, Hahn and colleagues [2012] propose to reformulate the optimization problem from Equation (3.46) in terms of both rig parameters $\mathbf{p}$ and internal nodes $\mathbf{n}$ to:

$$\{\mathbf{p}^{t+1}, \mathbf{n}^{t+1}\} = \arg\min_{\mathbf{p},\mathbf{n}} H\left(\{\mathbf{s}(\mathbf{p}), \mathbf{n}\}\right) \tag{4.1}$$

To minimize the objective function with respect to both $\mathbf{p}$ and $\mathbf{n}$, we need to consider the reduced derivatives with respect to those two vectors. The two reduced gradient vector strips are given as

$$\begin{aligned}
\mathbf{r}_{\mathbf{p}} &= \left(\frac{\partial}{\partial \mathbf{p}} H\left(\left\{\mathbf{s}\left(\mathbf{p}^t\right), \mathbf{n}^t\right\}\right)\right)^T \\
&= \left(\underbrace{\frac{\partial}{\partial \mathbf{p}} \mathbf{s}\left(\mathbf{p}^t\right)}_{\mathbf{J}_{\mathbf{s}}}\right)^T \underbrace{\left(\frac{\partial}{\partial \mathbf{s}} H\left(\left\{\mathbf{s}\left(\mathbf{p}^t\right), \mathbf{n}^t\right\}\right)\right)^T}_{\mathbf{g}_{\mathbf{s}}} \\
&= (\mathbf{J}_{\mathbf{s}})^T \mathbf{g}_{\mathbf{s}}, \\
\mathbf{r}_{\mathbf{n}} &= \underbrace{\left(\frac{\partial}{\partial \mathbf{n}} H\left(\left\{\mathbf{s}\left(\mathbf{p}^t\right), \mathbf{n}^t\right\}\right)\right)^T}_{\mathbf{g}_{\mathbf{n}}} \\
&= \mathbf{g}_{\mathbf{n}}.
\end{aligned} \tag{4.2}$$

The reduced second derivatives of the objective function with respect to both

$\mathbf{p}$ and $\mathbf{n}$ are given as three Hessian block matrices:

$$
\begin{aligned}
\mathbf{K_{pp}} &= \frac{\partial^2}{\partial \mathbf{p}^2} H\left(\left\{\mathbf{s}\left(\mathbf{p}^t\right), \mathbf{n}^t\right\}\right) \\
&= \left(\underbrace{\frac{\partial}{\partial \mathbf{p}} \mathbf{s}\left(\mathbf{p}^t\right)}_{\mathbf{J_s}}\right)^T \underbrace{\frac{\partial^2}{\partial \mathbf{s}^2} H\left(\left\{\mathbf{s}\left(\mathbf{p}^t\right), \mathbf{n}^t\right\}\right)}_{\mathbf{H_{ss}}} \underbrace{\frac{\partial}{\partial \mathbf{p}} \mathbf{s}\left(\mathbf{p}^t\right)}_{\mathbf{J_s}} \\
&\qquad + \underbrace{\frac{\partial^2}{\partial \mathbf{p}^2} \mathbf{s}\left(\mathbf{p}^t\right)}_{\mathbf{T_s}} : \underbrace{\left(\frac{\partial}{\partial \mathbf{s}} H\left(\left\{\mathbf{s}\left(\mathbf{p}^t\right), \mathbf{n}^t\right\}\right)\right)^T}_{\mathbf{g_s}} \\
&= \left(\mathbf{J_s}\right)^T \mathbf{H_{ss}} \mathbf{J_s} + \mathbf{T_s} : \mathbf{g_s}, \\
\mathbf{K_{pn}} &= \frac{\partial^2}{\partial \mathbf{p} \partial \mathbf{n}} H\left(\left\{\mathbf{s}\left(\mathbf{p}^t\right), \mathbf{n}^t\right\}\right) \\
&= \left(\underbrace{\frac{\partial}{\partial \mathbf{p}} \mathbf{s}\left(\mathbf{p}^t\right)}_{\mathbf{J_s}}\right)^T \underbrace{\frac{\partial^2}{\partial \mathbf{s} \partial \mathbf{n}} H\left(\left\{\mathbf{s}\left(\mathbf{p}^t\right), \mathbf{n}^t\right\}\right)}_{\mathbf{H_{sn}}} \\
&= \left(\mathbf{J_s}\right)^T \mathbf{H_{sn}}, \\
\mathbf{K_{nn}} &= \underbrace{\frac{\partial^2}{\partial \mathbf{n}^2} H\left(\left\{\mathbf{s}\left(\mathbf{p}^t\right), \mathbf{n}^t\right\}\right)}_{\mathbf{H_{nn}}} \\
&= \mathbf{H_{nn}}
\end{aligned}
\tag{4.3}
$$

With these two reduced gradient strips and three Hessian blocks in hand, we can now assemble the reduced gradient and Hessian for the linear system in Equation (3.50) as

$$
\mathbf{r} = \begin{bmatrix} \mathbf{r_p} \\ \mathbf{r_n} \end{bmatrix} \quad \text{and} \quad \mathbf{K} = \begin{bmatrix} \mathbf{K_{pp}} & \mathbf{K_{pn}} \\ \left(\mathbf{K_{pn}}\right)^T & \mathbf{K_{nn}} \end{bmatrix}, \tag{4.4}
$$

which we can then solve for the unknown step direction $\Delta\{\mathbf{p}, \mathbf{n}\}$. Since the elements of $\mathbf{K_{nn}}$ are simply a subset of the elements of the sparse full-space Hessian $\mathbf{H}$, $\mathbf{K_{nn}}$ is also sparse. However, no sparsity information is available for $\mathbf{K_{pp}}$ and $\mathbf{K_{pn}}$, leading to dense parts of the resulting reduced Hessian $\mathbf{K}$ and rendering the use of a sparse linear solver inefficient. To alleviate this issue, Hahn and colleagues [2012] propose to use a *Schur Complement Solver* scheme to speed up the linear system solve of Equation (3.50) using the subspace derivative components from Equation (4.4). The idea of the scheme is

to perform a block Gauss elimination step on Equation (4.4), resulting in two decomposed systems of equations that sequentially solve for step directions for rig parameters $\Delta\mathbf{p}$ and internal nodes $\Delta\mathbf{n}$, respectively:

$$\begin{aligned}\left(\mathbf{K_{pp}} - \mathbf{K_{pn}}\left(\mathbf{K_{nn}}\right)^{-1}\left(\mathbf{K_{pn}}\right)^{T}\right)\Delta\mathbf{p} &= \mathbf{r_p} - \mathbf{K_{pn}}\left(\mathbf{K_{nn}}\right)^{-1}\mathbf{r_n} \\ \mathbf{K_{nn}}\Delta\mathbf{n} &= \mathbf{r_n} - \left(\mathbf{K_{pn}}\right)^{T}\Delta\mathbf{p}\end{aligned} \tag{4.5}$$

Since $\mathbf{K_{nn}}$ is sparse, it can be prefactored as in Equation (3.37) and never has to be explicitly inverted. Further, the only dense linear system to be solved is the one for $\Delta\mathbf{p}$, which is assumed to be small since in practice the number of rig subspace parameters is a lot smaller than the number of vertices in the simulation mesh.

As demonstrated by Hahn and colleagues [2012], the above formulation affords high-quality simulations of secondary motion and other physics-based detail in rig space. However, the resulting algorithm is computationally intensive for two primary reasons. First, minimizing Equation (4.1) with a Newton-Raphson scheme requires the first and second derivatives $\mathbf{J_s}$ and $\mathbf{T_s}$ of the rig $\mathbf{s}$ with respect to its parameters $\mathbf{p}$. Since the rig is generally not available in analytic form, these derivatives have to be estimated using finite differences for each iteration of the solver. Second, the dimension of the resulting system is comparatively high, considering that only the free rig parameters are required. In the remainder of this section, we describe a method that greatly accelerates computations without compromising quality.

Our goal is to develop a formulation of rig-space physics that affords the same level of quality but is significantly faster. We achieve this target by establishing a linearized formulation, eliminating the internal degrees of freedom using volumetric skinning, and using a deferred Jacobian evaluation.

4.2.2 Linear Rig Approximation

We start by linearizing the rig at the beginning of every time step as:

$$\mathbf{s}(\mathbf{p}) \approx \tilde{\mathbf{s}}(\mathbf{p}) = \mathbf{s}\left(\mathbf{p}^t\right) + \underbrace{\frac{\partial}{\partial\mathbf{p}}\mathbf{s}\left(\mathbf{p}^t\right)}_{\mathbf{J_s}} \cdot \left(\mathbf{p} - \mathbf{p}^t\right), \tag{4.6}$$

This simplification is reminiscent of the semi-implicit Euler scheme described by Baraff and Witkin [1998], which relies on linearized forces. However, an important difference of our approach is that we linearize the rig but not the elastic forces and, as a result, the equations of motion remain

nonlinear. Using nonlinear as opposed to linearized elastic forces leads to improved stability and requires only a few evaluations of elastic energy gradients and Hessians, which are significantly faster to compute than evaluating $\mathbf{J_s}$. As a direct computational advantage of the rig linearization, we need to evaluate the rig Jacobian $\mathbf{J_s}$ only once per time step and, moreover, the second-order derivatives $\mathbf{T_s}$ of the rig vanish altogether and are no longer required to compute the reduced Hessian blocks in Equation (4.3).

4.2.3 Physics-based Volumetric Skinning

In the original rig-space physics method by Hahn and colleagues [2012], the problem variables consist of the rig parameters $\mathbf{p}$ and the position of the internal vertices $\mathbf{n}$. While a typical rig might feature around 100 rig parameters, the resolution of the character mesh is usually a lot larger with tens of thousands of vertices. Besides the fact that the internal vertices contribute significantly more degrees of freedom than the rig parameters, the internal vertices serve only a helper role in the computation of the internal energy for a given set of rig parameters. They are not visible in the resulting animation and of no interest to the artist. We would like to establish a formulation in which only the truly relevant variables, namely the rig parameters, are exposed as degrees of freedom.

To this end, we start by assuming that the positions of the internal vertices are always defined by the boundary vertices through static equilibrium conditions:

$$\tilde{\mathbf{n}}(\mathbf{p}) = \arg\min_{\mathbf{n}} W_{\text{elastic}}\left(\{\mathbf{s}(\mathbf{p}), \mathbf{n}\}\right). \tag{4.7}$$

While this formula defines a unique mapping from rig parameters to internal vertices, the corresponding function is implicit: it requires minimizing the elastic energy and thus solving a set of nonlinear equations. Since doing so would be computationally expensive, our goal is to approximate this implicit nonlinear map by an explicit linear function.

One option is to use cage-based deformation techniques such as Harmonic coordinates [Joshi et al., 2007] or Green coordinates [Lipman et al., 2008] that are used for deforming a high resolution embedded surface with an enveloping mesh. There are, however, two drawbacks to these approaches. First, an artist needs to design and rig a cage since there are no reliable automatic methods for this task. Second, while the deformation field inside the cage is smooth, the corresponding vertex positions will generally be far from their equilibrium positions as dictated by the underlying elastic mate-

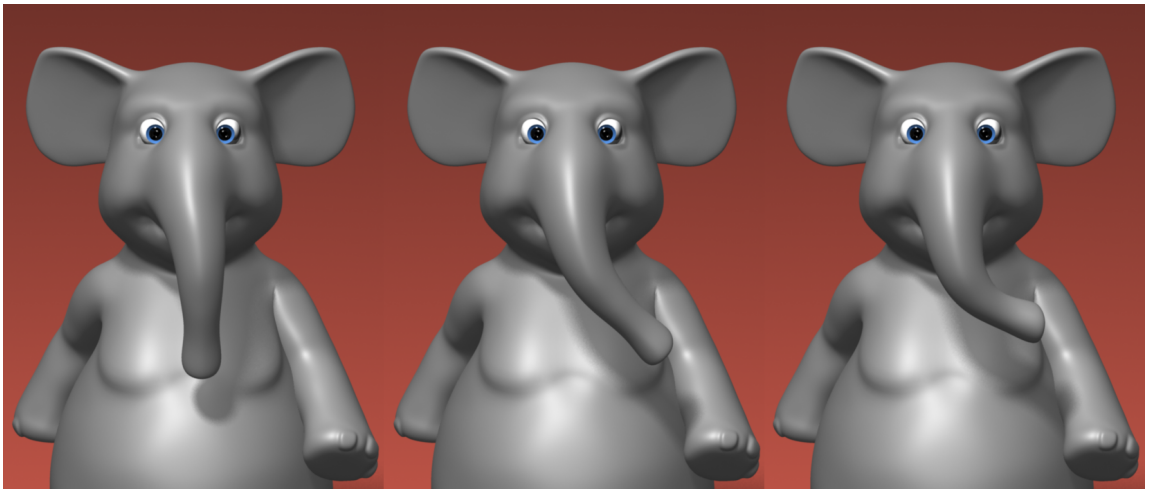

Figure 4.2: *An impulse vector applied along the horizontal image axis results in a swinging motion for the elephant's trunk, providing us with a sequence of surface deformations and corresponding internal deformations.*

rial. This disparity leads to an overestimation of the elastic energy and, in turn, severely affects the dynamics of the character.

In contrast to typical cage-based modeling problems, we have explicit knowledge about how the internal deformation field should evolve as a function of the surface mesh. Namely, for every surface configuration, we can compute the internal vertex positions by minimizing a nonlinear elastic energy. This observation motivates an example-based approach in order to compute an optimal linear approximation to the internal deformation field. We first describe how to generate a set of example poses and then explain how to compute the linear map.

Generating Example-Poses

We assume that there is a small set of about five to ten artist-generated poses that are representative of the typical range of motion during animation. In a production environment, such poses are typically created as a means of testing the character during the rigging stage and are referred to as *calisthenics*. Given these basic poses, we generate an augmented example set by applying a small number of impulse vectors to the surface of the character. Each of the impulse vectors defines initial velocities for the character that we use to perform a few steps of dynamic simulation by solving the original rig-space physics problem from Equation (4.1) for the free rig parameters $\mathbf{p}$ as well as the corresponding equilibrium positions $\tilde{\mathbf{n}}(\mathbf{p})$ of the internal vertices. The result of this process, which we refer to as *shaking*, is a sequence

of surface positions and corresponding internal deformation which we add to the example set. In this way, we can generate a wider range of poses that also reflects the influence of the simulated rig parameters $\mathbf{p}$ on the internal vertices $\mathbf{n}$. A few poses resulting from this process are shown in Figure 4.2.

Example-Based Skinning

The shaking process provides us with a set of m_{ex} deformed example configurations $\{\mathbf{s}^i, \tilde{\mathbf{n}}^i\}$ of the whole simulation mesh, comprising both surface and internal vertices that correspond to different poses of the character. For each internal vertex $\tilde{\mathbf{n}}_j \in \mathbb{R}^3$, we then seek to find weights $\tilde{\mathbf{w}}^j \in \mathbb{R}^{n_\mathbf{s}}$ with respect to the surface vertices $\mathbf{s}$ that best explain the position of the internal vertex $\tilde{\mathbf{n}}_j^i$ in all the m_{ex} different example poses:

$$\begin{aligned}\tilde{\mathbf{w}}^j &= \arg\min_{\mathbf{w}^j} \left\| \underbrace{\begin{bmatrix} \mathbf{s}_1^1 & \cdots & \mathbf{s}_n^1 \\ \vdots & \ddots & \vdots \\ \mathbf{s}_1^{m_{\text{ex}}} & \cdots & \mathbf{s}_n^{m_{\text{ex}}} \\ 1 & \cdots & 1 \end{bmatrix}}_{\mathbf{S}} \mathbf{w}^j - \underbrace{\begin{bmatrix} \tilde{\mathbf{n}}_j^1 \\ \vdots \\ \tilde{\mathbf{n}}_j^{m_{\text{ex}}} \\ 1 \end{bmatrix}}_{\tilde{\mathbf{n}}_{\text{ex}}^j} \right\|_2^2 \\ &= \arg\min_{\mathbf{w}^j} \left\| \mathbf{S}\mathbf{w}^j - \tilde{\mathbf{n}}_{\text{ex}}^j \right\|_2^2 \end{aligned} \tag{4.8}$$

In the above system, each row stands for three equations representing the x, y and z components, while the last row asks that weights sum to one, which ensures that the internal vertices transform correctly under rigid transformations of the surface. After solving the above problem for every vertex, we obtain a linear map between surface and internal vertices

$$\tilde{\mathbf{n}}(\mathbf{p}) = \tilde{\mathbf{W}}\mathbf{s}(\mathbf{p}), \tag{4.9}$$

where $\tilde{\mathbf{W}} \in \mathbb{R}^{3n_\mathbf{n} \times 3n_\mathbf{s}}$ denotes the *skinning matrix*. It is defined as

$$\tilde{\mathbf{W}} = \begin{bmatrix} \tilde{\mathbf{w}}^1 & \cdots & \tilde{\mathbf{w}}^{n_\mathbf{n}} \end{bmatrix}^T \otimes \mathbf{I}_3 = \begin{bmatrix} (\tilde{\mathbf{w}}^1)_1 \cdot \mathbf{I}_3 & \cdots & (\tilde{\mathbf{w}}^1)_{n_\mathbf{s}} \cdot \mathbf{I}_3 \\ \vdots & \ddots & \vdots \\ (\tilde{\mathbf{w}}^{n_\mathbf{n}})_1 \cdot \mathbf{I}_3 & \cdots & (\tilde{\mathbf{w}}^{n_\mathbf{n}})_{n_\mathbf{s}} \cdot \mathbf{I}_3 \end{bmatrix}, \tag{4.10}$$

where $\mathbf{I}_3 \in \mathbb{R}^{3\times3}$ denotes an identity matrix and $\otimes$ is the Kronecker product. Due to its similarity to linear blend skinning, we refer to this map $\tilde{\mathbf{n}}(\mathbf{p})$ as *physics-based volumetric skinning*.

The formulation of the optimization problem in Equation (4.8) is not yet practical. First, it allows weights to be negative, which, as reported by James and Twigg [2005], can lead to overfitting: large positive and negative weights can lead to a better fit, but outside the training data, undesirable deformations occur. Second, it assumes dense correspondences since each internal vertex can potentially be influenced by every surface point. This property deteriorates run-time performance and, as shown in Section 4.3, we found that dense correspondences again give rise to overfitting. A practical explanation for this undesirable behavior is that dense correspondences do not respect the inherent locality of the problem: the position of the internal vertices is directly influenced only by a certain neighbor set of close-by surface vertices. Despite this locality, the dense correspondence scheme can still use remote surface vertices in order to better explain the position of a given internal vertex if the training data so happens to support this prediction. However, remote correspondences do not generalize well to data outside the training set, resulting in the aforementioned overfitting.

Ideally, we would like to choose the smallest set of close-by surface vertices that yields a robust fit to the training data. Alas, automatically computing such a neighbor set is challenging. Thresholding based on Euclidean distance is difficult to implement robustly since it is unclear how to choose the cutoff value. Internal vertices in some regions, such as the elephant's belly, can be much further away from the surface than in other regions, like the arms.

Sparse Correspondences

Our solution to this problem is to ask for a sparse set of correspondences that yields a fit to the training data with a guaranteed upper bound on the error. To this end, we augment the optimization problem from Equation (4.8) with two components:

1. We add a sparse regularizer to the minimization problem that penalizes the L_1-norm $\|\mathbf{w}^j\|_1$ of the weight vector, thus favoring a sparse set of correspondences. We choose the *SmoothL1* regularizer proposed by Schmidt and colleagues [2007], which approximates the non-differentiable L_1 operator using a smooth approximation of the absolute value function parameterized by a parameter α that converges to the exact absolute value function as $\alpha \to \infty$:

$$|x| \approx \frac{1}{\alpha}\Big(\log\big(1+\exp\left(-\alpha x\right)\big) + \log\big(1+\exp\left(\alpha x\right)\big)\Big) \qquad (4.11)$$

2. We add a hard constraint on the weights $\mathbf{w}^j$ as proposed by James and Twigg [2005] that forces them all to be positive:

$$\forall k : \left(\mathbf{w}^j\right)_k \geq 0 \tag{4.12}$$

In this way, we eliminate overfitting since only those vertices are used that are actually required to explain the behavior of an internal vertex, while also preventing counter-intuitive negative weights. At the same time, we avoid the problem of having to heuristically determine the right sets of neighbors a priori. As another advantage, the significantly reduced neighbor set also speeds up computations at run-time.

Rather than applying both the L_1 regularization from Equation (4.11) and the non-negativity constraint from Equation (4.12) at the same time, we found it beneficial to alternate between the two. We will refer to a solve of the optimization problem from Equation (4.8) using the sparse regularizer as SOLVEL1, and solve it starting from some initial value using an iterative Newton-Raphson scheme alike the one previously described in Section 3.2. In each iteration, we successively increase α until the solver converges. On the other hand, solving the optimization problem using the non-negativity constraint results in a nonlinear least squares (NLLS) problem, which we solve using a quadratic programming (QP) solver. We use the Mosek library [ApS, 2015] as our QP algorithm of choice, and will refer to invocations of it as SOLVENNLS. Each call to SOLVEL1 or SOLVENNLS requires the index j of the internal node to optimize for and an initial set of correspondences $\mathcal{S}_0$ used to populate the system matrix $\mathbf{S}$. In addition to an optimized weight vector $\mathbf{w}^j$, the two routines also return the residual error *res* of the optimization problem in Equation (4.8) after minimization.

Our method for computing sparse weights is described in Algorithm 4. Starting from a conservative or even full set of correspondences $\mathcal{S}_0$, we first determine the initial error res_0 of the fit using SOLVENNLS (line 3). We then iteratively solve the L_1-regularized version of the optimization problem using SOLVEL1 (line 8) and remove the surface vertices with the smallest weights from the correspondence set (line 7). The iteration is stopped whenever removing an additional vertex would lead to a residual error larger than a given threshold value τ_{res} or a residual error higher than a threshold factor τ_{ratio} times the initial residual res_0. We set τ_{res} to 0.1% of the size of the character's bounding box and chose $\tau_{\text{ratio}} = 1.5$ for all of our examples. After the iteration is stopped, we recompute the final weights again without the L_1-regularizer using SOLVENNLS (line 10).

The results of our sparse correspondence algorithm are small sets of surface vertices and corresponding weights that explain the behavior of the internal

Algorithm 4: *Finding a sparse correspondence set for skinning*

function SPARSECORRESPONDENCES($\mathcal{S}_0$)
  for all internal vertices j **do**
    $\left(\tilde{\mathbf{w}}^j, res_0\right) \leftarrow$ SOLVENNLS(j, $\mathcal{S}_0$)
    $res \leftarrow res_0$
    $\mathcal{S} \leftarrow \mathcal{S}_0$
    while $res < \tau_{\text{res}}$ **or** $\frac{res}{res_0} < \tau_{\text{ratio}}$ **do**
      reduce $\mathcal{S}$ and $\tilde{\mathbf{w}}^j$ by smallest weight
      $\left(\tilde{\mathbf{w}}^j, res\right) \leftarrow$ SOLVEL1(j, $\mathcal{S}$, $\tilde{\mathbf{w}}^j$)
    end while
    $\left(\tilde{\mathbf{w}}^j, res\right) \leftarrow$ SOLVENNLS(j, $\mathcal{S}$)
  end for
end function

vertices in a robust and efficient way. As we show in Section 4.3, using sparse correspondences improves both the computational efficiency at runtime and eliminates overfitting.

4.2.4 Deferred Jacobian Evaluation

Even when keeping the Jacobian $\mathbf{J_s}$ constant per time step, its evaluation still constitutes a major part of the total computational cost. Yet, due to the inherent temporal coherence in animations, the Jacobian often does not change significantly from one time step to the next. Ideally, we would like to recompute the Jacobian only when necessary. While it is—to some extent—acceptable to trade accuracy for performance, we cannot compromise on stability. We therefore need a robust indicator for evaluating the error incurred by keeping the same Jacobian over multiple time steps.

In order to quantify the error of the current approximation, we compare the end-of-time-step positions $\tilde{\mathbf{s}}\left(\mathbf{p}^{t+1}\right)$ predicted by the linear approximation from Equation (4.6) to the actual positions $\mathbf{s}\left(\mathbf{p}^{t+1}\right)$ evaluated using the original rig mapping. A natural metric for this difference is the kinetic energy due to the difference in position over the time step h which is given by

$$\Delta E_{kin} = \frac{1}{2h}\left(\tilde{\mathbf{s}}\left(\mathbf{p}^{t+1}\right) - \mathbf{s}\left(\mathbf{p}^{t+1}\right)\right)^T \mathbf{M_s}\left(\tilde{\mathbf{s}}\left(\mathbf{p}^{t+1}\right) - \mathbf{s}\left(\mathbf{p}^{t+1}\right)\right), \tag{4.13}$$

where $\mathbf{M_s}$ denotes a diagonal matrix containing the parts of the mass matrix $\mathbf{M}$ that refer to the surface vertices $\mathbf{s}$.

Computing this indicator requires only one rig evaluation, but it provides us with valuable information about the *linearity* of the rig around the cur-

rent set of parameters and in the direction of the character's motion. Taking an optimistic approach, we always attempt to reuse the existing Jacobian to step the rig parameters forward in time. We then evaluate the indicator and, if it signals too high a degree of nonlinearity, we roll back to the beginning of the step, compute the Jacobian $\mathbf{J_s}$, and simulate again. While this approach is always robust and efficient in many cases, animations with rapid motion and extreme deformations can lead to an excessive number of rollbacks, effectively undoing the advantage of deferred Jacobian evaluation. In order to decrease the number of such rollbacks, we use an additional indicator that estimates the linearity of the rig in the relevant direction without requiring a full simulation step: Instead of solving for $\mathbf{p}^{t+1}$ first, we simply estimate the end-of-time-step parameters $\tilde{\mathbf{p}}$ using a linear extrapolation of the past two states as

$$\tilde{\mathbf{p}} = \mathbf{p}^t + \left(\mathbf{p}^t - \mathbf{p}^{t-1}\right) = 2 \cdot \mathbf{p}^t - \mathbf{p}^{t-1}. \tag{4.14}$$

As shown in Section 4.3, this indicator leads to significantly fewer rollbacks while limiting the number of Jacobian reevaluations.

4.2.5 Implementation

With our physics-based skinning method, the rig-space physics objective function of Equation (4.1) now only depends on the free rig parameters $\mathbf{p}$ such that

$$\begin{aligned} H(\mathbf{p}) &= H\left(\{\mathbf{s}(\mathbf{p}), \tilde{\mathbf{n}}(\mathbf{p})\}\right) \\ &= H\left(\underbrace{\left\{\mathbf{s}(\mathbf{p}), \tilde{\mathbf{W}}\mathbf{s}(\mathbf{p})\right\}}_{\mathbf{x}(\mathbf{s}(\mathbf{p}))}\right) \\ &= H(\mathbf{x}(\mathbf{s}(\mathbf{p}))), \end{aligned} \tag{4.15}$$

where we have made use of the volumetric skinning formulation from Equation (4.9) for the step on the second line.

We perform time stepping by minimizing $H(\mathbf{p})$ using Newton's method as described in Section 3.3, which requires the reduced gradient $\mathbf{r}$ and the reduced Hessian $\mathbf{k}$ of H with respect to the parameters $\mathbf{p}$. While each iteration reevaluates the the full-space gradient $\mathbf{g}$ and the full-space Hessian $\mathbf{H}$, the rig Jacobian $\mathbf{J}$ now stays constant.

Using the chain rule and the full-space gradient strips notation from Equa-

tion (4.2), the reduced gradient $\mathbf{r}$ simplifies to:

$$\begin{aligned}
\mathbf{r} &= \left(\frac{\partial}{\partial \mathbf{p}} H\left(\mathbf{x}\left(\mathbf{s}\left(\mathbf{p}^t\right)\right)\right)\right)^T \\
&= \left(\underbrace{\frac{\partial}{\partial \mathbf{p}} \mathbf{s}\left(\mathbf{p}^t\right)}_{\mathbf{J_s}}\right)^T \left(\frac{\partial}{\partial \mathbf{s}} \mathbf{x}\left(\mathbf{s}\left(\mathbf{p}^t\right)\right)\right)^T \underbrace{\left(\frac{\partial}{\partial \mathbf{x}} H\left(\mathbf{x}\left(\mathbf{s}\left(\mathbf{p}^t\right)\right)\right)\right)^T}_{\mathbf{g}} \\
&= \left(\mathbf{J_s}\right)^T \begin{bmatrix}\mathbf{I_s} & \tilde{\mathbf{W}}\end{bmatrix}^T \mathbf{g} \\
&= \left(\mathbf{J_s}\right)^T \left(\mathbf{g_s} + \tilde{\mathbf{W}}^T \mathbf{g_n}\right)
\end{aligned} \tag{4.16}$$

Using both the chain rule, the product rule, and the full-space Hessian blocks notation from Equation (4.3), the reduced Hessian $\mathbf{K}$ is given as:

$$\begin{aligned}
\mathbf{K} &= \frac{\partial^2}{\partial \mathbf{p}^2} H\left(\mathbf{x}\left(\mathbf{s}\left(\mathbf{p}^t\right)\right)\right) \\
&= \left(\underbrace{\frac{\partial}{\partial \mathbf{p}} \mathbf{s}\left(\mathbf{p}^t\right)}_{\mathbf{J_s}}\right)^T \left(\frac{\partial}{\partial \mathbf{s}} \mathbf{x}\left(\mathbf{s}\left(\mathbf{p}^t\right)\right)\right)^T \underbrace{\frac{\partial^2}{\partial \mathbf{x}^2} H\left(\mathbf{x}\left(\mathbf{s}\left(\mathbf{p}^t\right)\right)\right)}_{\mathbf{H}} \\
&\qquad\qquad \frac{\partial}{\partial \mathbf{s}} \mathbf{x}\left(\mathbf{s}\left(\mathbf{p}^t\right)\right) \underbrace{\frac{\partial}{\partial \mathbf{p}} \mathbf{s}\left(\mathbf{p}^t\right)}_{\mathbf{J_s}} \\
&= \left(\mathbf{J_s}\right)^T \begin{bmatrix}\mathbf{I_s} & \tilde{\mathbf{W}}\end{bmatrix}^T \mathbf{H} \begin{bmatrix}\mathbf{I_s} & \tilde{\mathbf{W}}\end{bmatrix} \mathbf{J_s} \\
&= \left(\mathbf{J_s}\right)^T \left(\mathbf{H_{ss}} + \mathbf{H_{sn}}\tilde{\mathbf{W}} + \tilde{\mathbf{W}}^T \left(\mathbf{H_{sn}}\right)^T + \tilde{\mathbf{W}}^T \mathbf{H_{nn}} \tilde{\mathbf{W}}\right) \mathbf{J_s}
\end{aligned} \tag{4.17}$$

As can be seen from Equations (4.16) and (4.17), the previously required solve for the interior vertices $\mathbf{n}$ has now successfully been replaced by a slightly more involved subspace projection involving the optimized sparse skinning matrix $\tilde{\mathbf{W}}$.

4.3 Results

In this section, we present results of our method and compare its simulation performance to previous work. We also validate the physics-based skinning method by comparing it to a state-of-the-art alternative approach.

Solver	**Trunk**		**Belly**		**Sumone**	
	t_{frame}	sp	t_{frame}	sp	t_{frame}	sp
Rig-Space Physics [Hahn et al., 2012]	7.24	×1	46.5	×1	—	—
statically solved interior, per-frame Jacobian	0.37	×19	0.53	×86	2.82	×1.0
statically solved interior, deferred Jacobian	0.39	×18	0.39	×118	2.48	×1.1
skinned interior, per-frame Jacobian	0.12	×56	0.27	×168	1.04	×2.7
skinned interior, deferred Jacobian	0.13	×53	0.13	×348	0.58	×4.9

Table 4.1: *Timings for the elephant walk cycle (*Trunk, Belly*) and the Sumone animation on an Intel Core i7-930 4 x 2.8Ghz using per-frame and deferred Jacobian evaluation, as well as static solves and skinning for the internal vertices.* t_{frame} *is computation time per frame in seconds, and* sp *is the speedup.*

4.3.1 Rig-Space Simulation

To achieve a fair comparison to the original rig-space physics system [Hahn et al., 2012], we apply our method to the elephant walk cycle animation from the original work as was seen in Figure 3.1, using the same parameter values as reported in the corresponding paper. To be consistent with the results of Hahn and colleagues [2012], we simulate the trunk and the belly of the elephant separately, which are controlled through 13 trunk and 36 belly rig parameters, respectively. Taking the animation produced by rig-space physics as the ground truth, we compare the simulation time as well as the difference in geometric deformation to our method using several different solver settings.

The results of these comparisons are summarized in Table 4.1 and a visual comparison for an exemplary frame is shown in Figure 4.3. As a first test, we only applied the rig linearization technique described in Section 4.2.2 (row 2), which already results in a significant speedup compared to the reference method (row 1). Yet, our physics-based volumetric skinning and its associated dimension reduction described in Section 4.2.3 achieves another drastic speedup (rows 4 and 5). We achieved the highest speedups of factors of up to 348 when further enabling the deferred Jacobian evaluation scheme described in Section 4.2.4 (rows 3 and 5). Despite these significant accelerations, the quality of the animation remains the same. Visually, the differences between the original method and our approach are imperceptible,

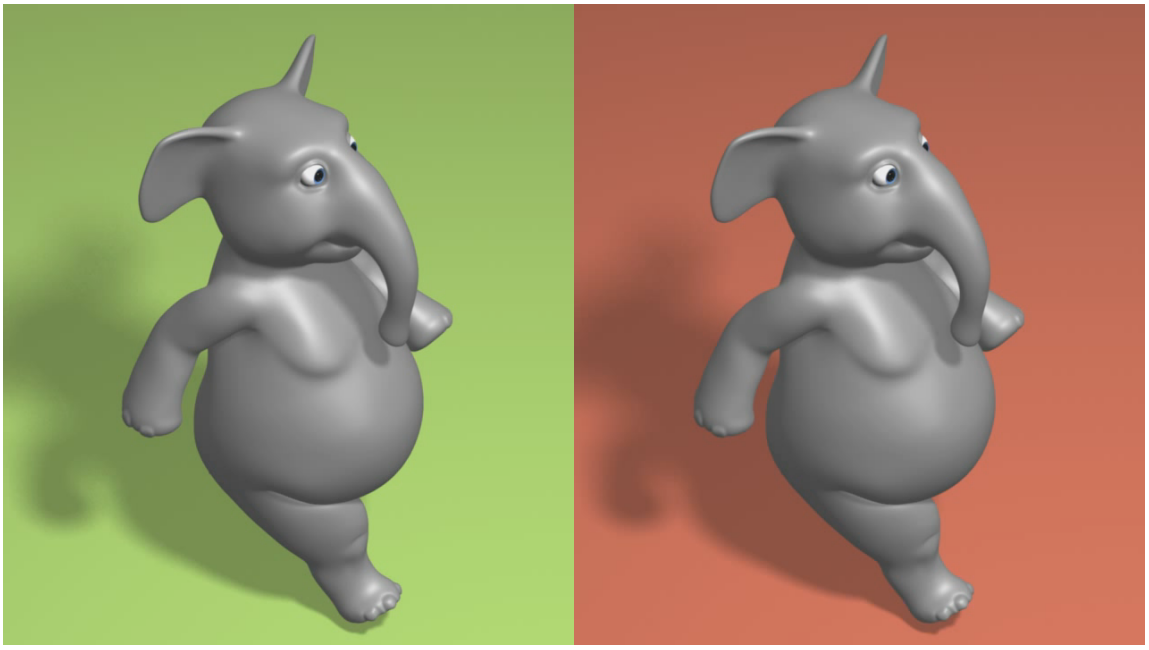

Figure 4.3: *A visual comparison between rig-space physics [Hahn et al., 2012] on the left, and our method for a selected frame on the right.*

as the maximum difference of an individual vertex position remains below 1.40% of the character's height for all frames, and the average difference is even lower with 0.08%.

Table 4.1 also indicates that our deferred Jacobian evaluation scheme (rows 3 and 5) is very effective for the belly of the elephant, whereas there is no speedup for the trunk. This result occurs because the rig is mostly linear in the region of the belly and our method can thus leverage the full potential of the deferred Jacobian evaluation. For the trunk, however, the rig is based on a nonlinear wire deformer and its Jacobian changes significantly in virtually every step of the animation. Deferred Jacobian evaluation is therefore not helpful for this example, but our linearity predictor is able to detect the nonlinearity of the rig and triggers Jacobian updates for 280 out of 360 frames. In contrast, when simulating the trunk with deferred Jacobian updates but without the predictor, 281 frames had to be resimulated, resulting in a severe performance penalty.

Our second example is an animation of a sumo wrestler, *Sumone*, performing a characteristic foot stomp followed by vigorous head shaking. In addition to the usual pose controls, this character also has a total of 174 rig parameter for secondary motion, all of which we simulate simultaneously using our method. An exemplary frame of this animation is shown in Figure 4.4, which showcases the ability of our method to achieve lively and organic-looking

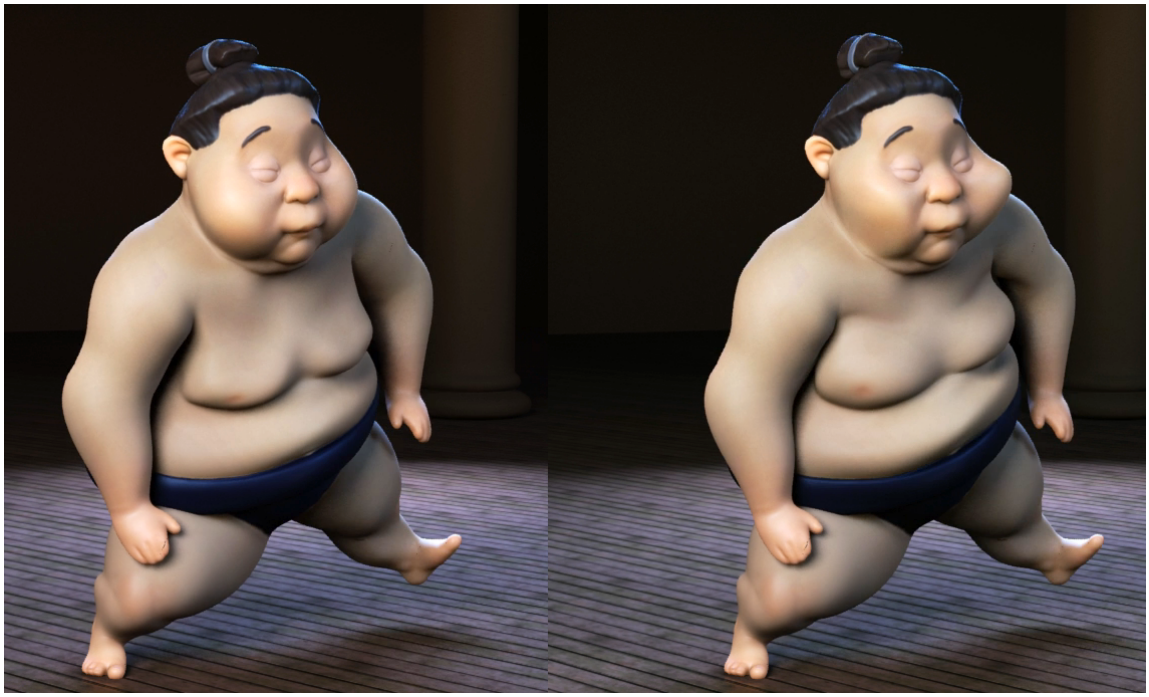

Figure 4.4: *An exemplary frame from an input animation without secondary motion* (left)*, and with secondary motion for belly, chest, hair, and cheeks simulated using our method* (right)*.*

motion for the character's belly, chest, cheeks, and hair. On the performance side, simulating this large number of rig parameters with the original rig-space physics solver is out of reach, whereas our method takes less than one second per animation frame. The timings for this animation are also listed in Table 4.1. Similar to the first example, our physics-based skinning offers a significant speedup, as does the deferred Jacobian evaluation.

4.3.2 Skinning

In order to analyze the efficiency of our skinning method, we compared it to the non-negative least squares (NNLS) solver described by James and Twigg [2005]. We quantify the performance of these approaches by plotting the resulting elastic energy for the individual frames of the elephant belly animation as shown in Figure 4.5. We start by using the NNLS scheme on a large set of 500 closest surface vertices for each internal vertex. We measure the distance between internal and surface vertices in a geodesic sense by marching along the volume mesh. In this way, we ensure that a surface vertex on the belly is not erroneously quantified as close to an internal vertex of an arm or the trunk.

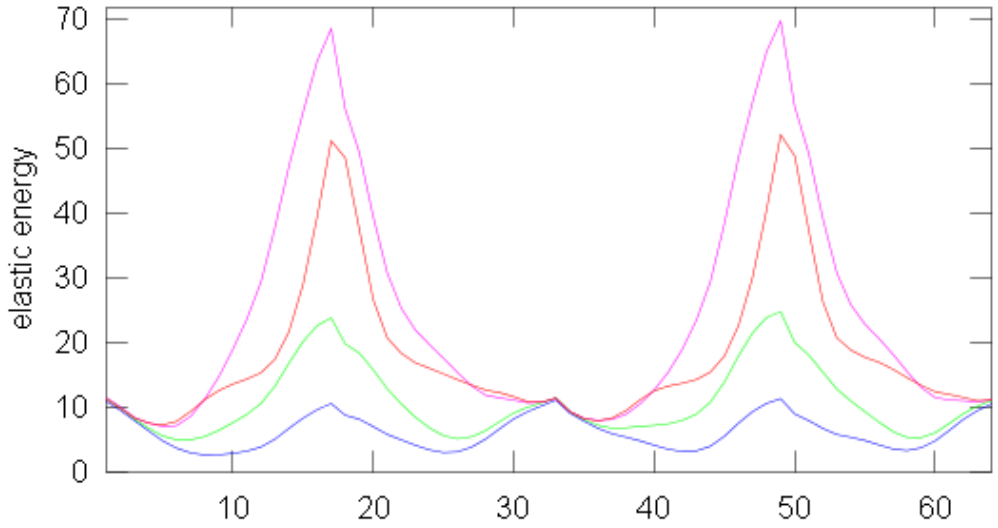

Figure 4.5: *Elastic energy plots for 64 frames of the elephant belly animation using static solve (blue), NNLS with 500 correspondences (red), NNLS with 20 correspondences (purple), and our physics-based skinning (green).*

As can be seen from Figure 4.5, the elastic energy obtained with NNLS weights on this large correspondence set (red curve) is significantly higher than the ground truth (blue curve), except for the regions around the three frames which were part of the training set. Moreover, the ground truth shows a regular saw-tooth pattern, but the NNLS solution exhibits two pronounced spikes where the elastic energy is approximately five times higher than the reference value. Reducing the per-vertex correspondence set to the 20 closest surface vertices even leads to slightly worse behavior (purple curve). However, starting from the same 20 correspondences per internal vertex, our physics-based skinning method is not only able to reduce the average number of correspondences to 5.56, it also leads to a much closer tracking of the reference data (green curve). This behavior can be explained by the fact that our sparsity-based weight computation scheme only keeps correspondences that are actually required, thus eliminating overfitting. As a final comparison, using NNLS with only the 5 closest surface vertices leads to unusable behavior due to extreme overfitting, and we were not able to generate an elastic energy curve for this setting (not shown).

In our system, the skinning weights have to be computed only once in a preprocessing step. The elephant mesh we used features 1328 interior vertices and 761 surface vertices. We trained the physics-based volumetric skinning for the belly and the trunk using 74 simulated poses. The initial correspondence sets of 20 surface vertices per internal vertex were reduced to an aver-

age of 5.52 and 5.55 in 30s and 29s per vertex for the trunk and belly, respectively. The Sumone mesh consists of 967 interior and 1302 surface vertices and we used 77 poses to train the skinning for the interior vertices. The initial correspondence sets of 20 surface vertices per internal vertex were reduced to 6.87 on average in 16.5s per vertex.

4.4 Summary and Outlook

We presented an efficient method for augmenting keyframed character animations with physically-simulated secondary motion. Our method achieves a performance improvement of one to two orders of magnitude over the reference solution by Hahn and colleagues [2012] from previous work without compromising on quality and while inheriting all of its benefits. This performance is based on a linearized formulation of rig-space dynamics that uses only rig parameters as degrees of freedom, a physics-based volumetric skinning method that allows our method to predict the motion of internal vertices solely from deformations of the surface, as well as a deferred Jacobian update scheme that drastically reduces the number of required rig evaluations. We demonstrated the performance of our method by comparing it to previous work and showcased its potential on a production-quality character rig.

In the next chapter, we will focus on a different application and a different optimization problem, while still making use of the powerful rig-space optimization concept and its runtime optimizations presented here. Further, the idea of using skinning as a tool to provide reference positions to a simulation system will be revisited in the later Chapter 6 on *Subspace Clothing Simulation*, where we will apply it to provide a kinematic reference for a simulated garment.

CHAPTER 5

Sketch Abstractions for Character Posing

Figure 5.1: *A stick-figure sketch abstraction allowed an artist to pose this 3D character using our sketch-based posing system, creating a run cycle in 3:05 minutes. The red lines show the sketched curves.*

5.1 Overview

In the previous chapter, we have seen how rig spaces can be used to enable artists to perform simulations of physical effects such as secondary motion and volume conservation, and directly add them to existing character animations. In this chapter, we will explore the power of the rigging concept and the deformation space it spans in the context of *character posing*, which

represents one of the crucial steps artists follow when creating keyframes to design new animations. While the focus for solving this problem is not on physics-based simulation, we will show that the energy minimization framework and the rig-space simulation concept introduced in Chapter 3 can be readily applied to character posing. In addition, some of the optimizations of the subspace solver proposed in Chapter 4 directly translate to the setting as well and will be reused. We will return to physics simulation in the next chapter.

Our work attempts to bring the direct control offered by sketching into the 3D animation pipeline with a sketch-based posing system that utilizes customized character sketch abstractions. A sketch abstraction is a set of rigged curves that form an iconographic 2D representation of the character from a particular viewpoint. Just as riggers currently craft the set of meaningful deformations for a character, in our system they also build this 2D abstraction to expose pose and shape variability at the level of granularity that is meaningful for the character. For example, a stick-figure representation could be used for overall skeletal posing while a torso outline could give further control over body shape. The distinguishing characteristic of our method is that it does not prescribe the sketch representation a priori, but rather empowers the rigging artist to encode the sketch representation that is most appropriate for the character in question.

Our core technical contribution focuses on enabling sketch-based posing within this setup with an optimization system that takes a new sketch as input and minimizes a nonlinear iterative closest point energy based on the 2D distance between this input sketch and the character's sketch abstraction. Solving this optimization problem with the rigging variables as unknowns aligns the character's pose to the input sketch. A custom regularizer addresses the underconstrained nature of the problem to select favorable poses. When rigging formulas and derivatives are available, our numerical method solves for poses extremely efficiently, under 400ms in our examples. Our system can also support arbitrary rigging controls offered by commercial software packages by treating the rigging system as a black box, albeit with a higher computational cost.

On the conceptual level, our primary contribution is the idea of a sketch abstraction for sketch-based posing using an artist-designed iconographic representation of the character. This technique relies on the technical contributions of our 2D nonlinear iterative closest point energy formulation, regularization, and optimization procedure that form a bridge between arbitrary 3D rigs and drawn 2D curves. Together, our system offers a more direct workflow for posing 3D characters in which the artist has the freedom to

prescribe the sketch representation that is most meaningful for the character to be animated.

We demonstrate our system's flexibility with several examples showing different artist-designed sketch abstractions used for sketch-based posing. For full-body posing, we show both a stick-figure representation that roughly follows a character's skeleton as well as an outline representation that tracks the outline of different body parts. We further demonstrate that sketch abstractions can be applied to individual components of a character authored separately. In this setting, the user can create a variety of customized creatures by sketching individual components such as a body, legs, heads, wings, and tails. The sketch abstraction for each part is much simpler than the actual 3D shape, showing that a simple iconographic sketch can result in a complex character design. Finally, we show that a sketch abstraction can also be generated on-the-fly by projecting a drawn curve onto the character's mesh. Redrawing this curve in the desired position allows the user to dynamically pose the character.

5.2 Method

We develop our sketch-based posing system in the context of existing animation production workflows where 3D character models are created and rigged using animation software packages. In this setting, the model's vertices define the space of all possible deformations, while the rig defines the subspace of meaningful ones. This rig may employ a range of controls, including skeletal kinematics, blend shapes, or arbitrary nonlinear procedural deformations to accommodate the character's range of expressive deformation. As with our efficient rig-space simulation system presented earlier in Chapter 4, our system can treat the rig as a black box, taking the character mesh and a list of arbitrary rig parameters as input. When analytic formulas for the rigging procedures are available, our method can utilize them for improved performance.

In order to provide a connection between the character's rigging controls and 2D input sketches, we enhance the classical rig parameterization with a sketch abstraction of the character from a particular viewpoint that is deformed by the same controls as the surface mesh. The character designer can author this sketch abstraction by creating and rigging a set of curves alongside the character's mesh. Or, alternatively, it can be created on-the-fly at runtime by drawing a curve onto the surface of the character which is then carried along as the character deforms. In either case, projecting the curves

into the camera's viewing plane yields a 2D representation of the character's current pose. Since the rig now simultaneously influences both the 3D character and the projected 2D sketch abstraction, our system effectively connects these two representations, allowing us to control the character's 3D shape by minimizing a 2D matching energy while using the rig parameters as unknowns. Given a new 2D sketch, we define an optimization problem in the form of a nonlinear iterative closest point (ICP) objective that attempts to align the character's 2D sketch abstraction to match the user-defined sketch. Since the optimization problem is expressed over the rig parameters, minimizing the ICP energy also deforms the 3D shape to match the sketch. A regularization term is used to resolve ambiguities in the large space of potential 3D deformations matching the 2D sketch.

5.2.1 Sketch Abstraction

As explained earlier in Section 4.2.1, typical artist-designed character rigs only deform their surface vertices $\mathbf{s}$, but additional interior vertices $\mathbf{n}$ might be required to represent them at as volume. Since we will not be simulating physics in this character posing context, we do not need such a volume representation or a distinction between surface and interior vertices. For notational simplicity, we will simply refer to the surface vertices as $\mathbf{x}$ for the remainder of this chapter.

In this case, a character rig defines a mapping from a set of rig parameters $\mathbf{p}$ to a corresponding surface deformation $\mathbf{x}(\mathbf{p})$ as in Equation (3.45). A *2D sketch abstraction* of the model enhances the rig with l deformed points $\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_l)$, giving rise to the *extended rig mapping*:

$$\mathbf{p} \rightarrow \{\mathbf{x}(\mathbf{p}), \mathbf{z}(\mathbf{p})\} \tag{5.1}$$

Our system supports arbitrary abstractions to give the rigging artist full control over the sketch representation of a character, but we also support generating them on-the-fly based on the character's surface rig $\mathbf{x}(\mathbf{p})$. To automatically extend a rig mapping by a sketch abstraction $\mathbf{z}$, we project an arbitrary source curve—2D or 3D—onto the surface mesh and compute barycentric coordinates $(\alpha_i, \beta_i, \gamma_i)$ of each curve point with respect to its three closest surface vertices $\mathbf{x}_a^i$, $\mathbf{x}_b^i$ and $\mathbf{x}_c^i$. Denoting the camera projection matrix by $\mathbf{C} \in \mathbb{R}^{2\times 3}$, we obtain the i-th 2D point of the sketch abstraction as:

$$\mathbf{z}_i(\mathbf{p}) = \mathbf{C}\left(\alpha_i \cdot \mathbf{x}_a^i(\mathbf{p}) + \beta_i \cdot \mathbf{x}_b^i(\mathbf{p}) + \gamma_i \cdot \mathbf{x}_c^i(\mathbf{p})\right) \tag{5.2}$$

Expressing the sketch abstraction $\mathbf{z}$ as a function of the rig deformation $\mathbf{x}$ allows us to easily transfer the surface deformation to the source curve based on the same rig parameters $\mathbf{p}$.

5.2.2 Matching Optimization

Sampling the input stroke provides us with a set of 2D points of size m given as $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m)$ that we are looking to match with the model's sketch abstraction $\mathbf{z}$. Since we would ideally like all points of the sketch abstraction $\mathbf{z}$ to coincide with some permutation of the user-drawn 2D points in $\mathbf{y}$, we define our *matching energy* as

$$W_{\text{match}}(\omega, \mathbf{z}(\mathbf{p})) = \sum_{i=1}^{l} \sum_{j=1}^{m} \omega_{ij} \cdot \left\| \mathbf{y}_j - \mathbf{z}_i(\mathbf{p}) \right\|_2^2, \tag{5.3}$$

where ω_{ij} denotes potential *correspondence* between points $\mathbf{z}_i$ and $\mathbf{y}_j$, associating them with each other. Further, we require there to be at least one correspondence entry for each $\mathbf{y}_i$ and each $\mathbf{z}_j$ to exclude trivial solutions:

$$\forall i \; \exists j : \; \omega_{ij} \geq 1 \quad \wedge \quad \forall j \; \exists i : \; \omega_{ij} \geq 1 \tag{5.4}$$

To minimize W_{match}, we follow the well known *iterative closest point* (ICP) approach and alternate between the following two stages:

- We fix the rig parameters as $\hat{\mathbf{p}}$ and thus also the 2D representation $\hat{\mathbf{z}} = \mathbf{z}(\hat{\mathbf{p}})$, and compute optimal correspondences ω that minimize W_{match} for the specific $\hat{\mathbf{p}}$.
- We fix the correspondences as $\hat{\omega}$, and minimize W_{match} with respect to the rig parameters $\mathbf{p}$.

Correspondences

We provide two procedures to optimize for the first ICP stage, where the choice of which to use is predefined for each sketch abstraction. In the first procedure, we perform an arc-length parameterized resampling using the same number of sample points for both the user sketch and the model's 2D sketch abstraction. We then establish straightforward one-to-one correspondences between $\hat{\mathbf{z}}$ and $\mathbf{y}$ for $\omega_{ij} \in \{0, 1\}$ by considering the drawing direction of both strokes. This procedure is best suited for matching line sketches as it makes sure that no direction flipping problems will occur, and it trivially satisfies the condition from Equation (5.4).

The second procedure is used for the matching of more complicated gestures such as circular shapes that do not necessarily have a clearly defined starting point or direction. After again performing an arc-length parameterized resampling with the same number of sample points for both the user sketch

and the 2D sketch abstraction of the model, we first compute *forward correspondences* $a_{ij} \in \{0, 1\}$ from $\hat{\mathbf{z}}$ to $\mathbf{y}$ by simply assigning the closest match $\mathbf{y}_j$ to every point $\hat{\mathbf{z}}_i$. Inversely, we compute *backward correspondences* $b_{ij} \in \{0, 1\}$ by assigning the closest sketch abstraction point $\hat{\mathbf{z}}_i$ to each user-drawn sketch point $\mathbf{y}_j$. We then combine a_{ij} and b_{ij} to many-to-many correspondences by summing them:

$$\omega_{ij} = a_{ij} + b_{ij} \tag{5.5}$$

By construction, the following properties hold for the forward correspondences and the backward correspondences:

$$\forall i : \sum_{j=1}^{l} a_{ij} = 1 \quad \text{and} \quad \forall j : \sum_{i=1}^{m} b_{ij} = 1 \tag{5.6}$$

Consequently, the combined weights satisfy $\omega_{ij} \in \{0, 1, 2\}$ and also the condition from Equation (5.4). We found that this second greedy matching procedure produces good results and is much faster than exact solutions to the underlying minimum bipartite matching problem such as the ones computed using variants of the Hungarian algorithm.

Subspace Optimization

In the second stage, our goal is to minimize the matching energy W_{match} by modifying the parameters $\mathbf{p}$ that control the sketch abstraction $\mathbf{z}$ of the model. Since the correspondences $\hat{\omega}$ are now fixed, we are effectively interested in the following optimization problem:

$$\mathbf{p}^t = \arg\min_{\mathbf{p}} W_{\text{match}}\left(\hat{\omega}, \mathbf{z}(\mathbf{p})\right), \tag{5.7}$$

Even though we are not performing a simulation over time, we will retain the ^{t} superscript to mark the optimized rig parameter vector after the t-th ICP iteration.

Closer inspection reveals that Equation (5.7) is in fact an instance of the rig-space optimization problem we have seen earlier in Section 3.3.2, since setting $h = 1$ and defining the energy potential set as $\mathcal{W} = \{W_{\text{match}}(\hat{\omega}, \cdot)\}$ in Equation (3.31) results in an objective function H compatible with Equation (3.46). Hence, we can make use of the subspace optimization method described in Section 3.3.2, while also employing the rig linearization technique introduced in Section 4.2.2 to achieve better performance. We will denote the required rig Jacobian of the sketch abstraction $\mathbf{z}$ with respect to the rig parameters $\mathbf{p}$ by $\mathbf{J}_{\mathbf{z}}$. If no formula for the rig mapping of the character is known, we estimate $\mathbf{J}_{\mathbf{z}}$ using finite differences as in Equation (3.51)

around the initial parameter vector $\mathbf{P}$. Otherwise, we assume that $\mathbf{J_z}$ can be computed analytically. Since W_{match} is a quadratic function in terms of $\mathbf{z}$, its derivatives with respect to $\mathbf{z}$ to compute the full-space gradient $\mathbf{g}$ and the full-space Hessian $\mathbf{H}$ are trivially obtained.

5.2.3 Regularization

Even though the matching energy W_{match} introduced before is well-defined, the optimization problem in Equation (5.7) is generally underconstrained as potentially many subspace parameter configurations $\mathbf{p}$—originally deforming the surface points $\mathbf{x}$ in 3D—map to the same 2D point set $\mathbf{z}$, resulting in an infinitely large set of candidate solutions. Further, many of these candidates correspond to surface representations $\mathbf{x}$ in 3D that are distorted even though the 2D points $\mathbf{z}$ match perfectly. To eliminate this null space of ambiguous solutions, we redefine our optimization problem as

$$\mathbf{p}^t = \arg\min_{\mathbf{p}} H(\mathbf{p}), \tag{5.8}$$

where we have added three regularization components to the objective function H:

$$H(\mathbf{p}) = W_{\text{match}}(\hat{\omega}, \mathbf{z}(\mathbf{p})) + W_{\text{pose}}(\mathbf{p}) + W_{\text{plane}}(\mathbf{x}(\mathbf{p})) + W_{\text{dist}}(\mathbf{x}(\mathbf{p})) \tag{5.9}$$

The first component is a $L2$ regularizer that favors solutions with the least amount of required *change from the initial pose*:

$$W_{\text{pose}}(\mathbf{p}) = \lambda_{\text{pose}} \cdot \left\| \mathbf{p} - \mathbf{p}^0 \right\|_2^2 \tag{5.10}$$

The second component favors deformations of the 3D surface points $\mathbf{x}$ within their *viewing plane* and is given by

$$W_{\text{plane}}(\mathbf{x}(\mathbf{p})) = \lambda_{\text{plane}} \cdot \sum_{i=1}^{n} \left(\mathbf{v}^T \left(\mathbf{x}_i(\mathbf{p}) - \mathbf{X}_i \right) \right)^2, \tag{5.11}$$

where $\mathbf{v}$ is the viewing direction of the camera and $\mathbf{X}_i = \mathbf{x}_i(\mathbf{P})$ denotes the position of vertex i prior to optimization.

The third component favors local deformations by penalizing deformations in regions far away from the user sketch. For each initial 3D vertex $\mathbf{X}_i$, we precompute its distance d_i to the closest point of the initial 2D representation

point $\mathbf{Z}_i$ projected onto the undeformed surface $\mathbf{X}$. We then regularize the *distance* using the term

$$W_{\text{dist}}(\mathbf{p}) = \lambda_{\text{dist}} \cdot \sum_{i=1}^{n} \frac{d_i}{d_{\max}} \|\mathbf{x}_i(\mathbf{p}) - \mathbf{X}_i\|_2^2, \tag{5.12}$$

where $d_{\max} = \max_i d_i$. We chose $\lambda_{\text{pose}} = 0.5$, $\lambda_{\text{plane}} = 0.01$ and $\lambda_{\text{dist}} = 0.001$ for all of our examples.

Subspace Derivatives

Since the regularized objective function H from Equation (5.9) we are minimizing now also directly depends on $\mathbf{p}$ and $\mathbf{x}(\mathbf{p})$, the derivatives $\mathbf{r}$ and $\mathbf{K}$ for the Newton-Raphson iterations from Equation (3.50) need to be adjusted:

$$\begin{aligned} \mathbf{r} &= \mathbf{r}_{\text{match}} + \mathbf{r}_{\text{pose}} + \mathbf{r}_{\text{plane}} + \mathbf{r}_{\text{dist}} \\ \mathbf{K} &= \mathbf{K}_{\text{match}} + \mathbf{K}_{\text{pose}} + \mathbf{K}_{\text{plane}} + \mathbf{K}_{\text{dist}} \end{aligned} \tag{5.13}$$

As mentioned before, the matching component of the objective function represents a rig-space optimization problem with respect to the 2D sketch abstraction rig mapping $\mathbf{z}(\mathbf{p})$. The same is true for the viewing plane and distance regularization components with respect to the 3D surface rig mapping $\mathbf{x}(\mathbf{p})$. We can thus instantiate the reduced gradient formula from Equation (3.48) and the reduced Hessian formula from Equation (3.49). Denoting the Jacobian of the 3D surface rig $\mathbf{x}$ with respect to rig parameters $\mathbf{p}$ by $\mathbf{J}_\mathbf{x}$, we obtain:

$$\begin{aligned} \mathbf{r}_{\text{match}} &= (\mathbf{J}_\mathbf{z})^T \mathbf{g}_{\text{match}} & \mathbf{K}_{\text{match}} &= (\mathbf{J}_\mathbf{z})^T \mathbf{H}_{\text{match}} \mathbf{J}_\mathbf{z} \\ \mathbf{r}_{\text{plane}} &= (\mathbf{J}_\mathbf{x})^T \mathbf{g}_{\text{plane}} & \mathbf{K}_{\text{plane}} &= (\mathbf{J}_\mathbf{x})^T \mathbf{H}_{\text{plane}} \mathbf{J}_\mathbf{x} \\ \mathbf{r}_{\text{dist}} &= (\mathbf{J}_\mathbf{x})^T \mathbf{g}_{\text{dist}} & \mathbf{K}_{\text{dist}} &= (\mathbf{J}_\mathbf{x})^T \mathbf{H}_{\text{dist}} \mathbf{J}_\mathbf{x} \end{aligned} \tag{5.14}$$

As with $\mathbf{g}_{\text{match}}$ and $\mathbf{H}_{\text{match}}$, the full-space derivatives $\mathbf{g}_{\text{plane}}$, $\mathbf{g}_{\text{dist}}$, $\mathbf{K}_{\text{plane}}$ and $\mathbf{K}_{\text{dist}}$ are easily obtained analytically since both W_{plane} and W_{dist} are quadratic functions in terms of $\mathbf{x}$.

The remaining pose regularization component W_{pose} is directly defined in terms of the rig parameters $\mathbf{p}$ and thus does not require a subspace projection. Its derivatives $\mathbf{r}_{\text{pose}}$ and $\mathbf{K}_{\text{pose}}$ are trivially obtained since W_{pose} is another quadratic function, this time in terms of $\mathbf{p}$.

Coarsening

While the parameter-based regularization term W_{pose} is inexpensive to compute, the vertex-based regularization terms W_{plane} and W_{dist} come at a higher cost. Since the objective function H from Equation (5.9) to minimize now also directly depends on $\mathbf{x}(\mathbf{p})$, the derivatives in Equation (5.13) also consider the vertex Jacobian $\mathbf{J}_\mathbf{x}$ as we have seen from Equation (5.14). Even if $\mathbf{J}_\mathbf{x}$ can be computed efficiently, its dimensions of $3n \times r$ effectively make the computation time of these subspace derivatives depend on the resolution of the 3D mesh, which impedes interactive use. To alleviate this issue, we preprocess the initial surface $\mathbf{X}$ by clustering its vertices $\mathbf{X}_i$ using the k-means algorithm with the Euclidean distance measure. By projecting the k cluster centers back to the surface $\mathbf{X}$ we obtain a vertex subset $\mathcal{R} \subset \{\mathbf{x}_i\}$ that we use as *representative elements* to compute coarsened variants of the two vertex-based regularization terms:

$$\begin{aligned} W_{\text{plane}}(\mathbf{x}(\mathbf{p})) &\approx \lambda_{\text{plane}} \cdot \frac{n}{k} \sum_{\mathbf{x}_i \in \mathcal{R}} \left(\mathbf{v}^T \left(\mathbf{x}_i(\mathbf{p}) - \mathbf{X}_i \right) \right)^2 \\ W_{\text{dist}}(\mathbf{x}(\mathbf{p})) &\approx \lambda_{\text{dist}} \cdot \frac{n}{k} \sum_{\mathbf{x}_i \in \mathcal{R}} \frac{d_i}{d_{\max}} \left\| \mathbf{x}_i(\mathbf{p}) - \mathbf{X}_i \right\|_2^2 \end{aligned} \tag{5.15}$$

Since these coarsened regularization terms only depend on the representative elements in $\mathcal{R}$, only k rows of $\mathbf{J}_\mathbf{x}$ need to be computed and multiplied. In our implementation, we never explicitly construct the full Jacobian $\mathbf{J}_\mathbf{x}$, effectively making the regularized optimization problem independent of the mesh resolution n. We use $k = 200$ clusters for all of our examples.

5.2.4 Linear Blend Skinning Rigs

Our method supports arbitrary rig mappings for both the character's surface and its sketch abstraction. In particular, we also support *black-box* rigs for which we can only evaluate their deformation but have no access to their analytic formula. This enables our method to pose existing rigs created in commercial animation packages such as Autodesk Maya, which allow artists to freely combine various rigging techniques into complex deformation hierarchies. While black-box rigs impose no construction constraints on the rigging artists, posing them using our method requires several seconds of computation time for the costly finite difference estimation of the Jacobians $\mathbf{J}_\mathbf{z}$ and $\mathbf{J}_\mathbf{x}$, making it cumbersome for interactive use. However, in case the

Example	dim **p**	dim **x**	dim **z**	**Runtime**
Elephant	132	6162	200	304ms
Cartoon Man (Stick figure)	91	6708	380	260ms
Cartoon Man (Outline)	91	6708	706	373ms
Face	44	19671	200	8922ms
Dragon	25	46380	200	100ms

Table 5.1: *Average runtimes per solver invocation until convergence is reached for the different applications and examples measured on an Intel Core i7 930 8 x 2.8GHz.*

character rig was designed using the well-known *linear blend skinning* technique, we can obtain analytic expressions of the Jacobians for efficient evaluation. The analytic form of the linear-blend skinning rig with its derivatives was described earlier in Section 3.4.3. Using Equations (3.66), (3.67) and (3.69) from that section, the Jacobians $\mathbf{J_x}$ and $\mathbf{J_z}$ are now easily obtained, only requiring derivatives of the transformation matrices $\mathbf{C}_j$ with respect to $\mathbf{p}$ in addition to the skinning weights $\mathbf{W}$ and the initial surface $\mathbf{X}$.

5.3 Results

In this section, we show how our sketch-based posing framework can be applied to various sketch abstractions designed for different applications. While the number of solver invocations varies depending on the application and the user's interaction with our system, we provide the average solver timings for all examples in Table 5.1. Our system is implemented within Autodesk Maya and utilizes Maya's rigging system and user interface to make it as intuitive as possible to the artist using it.

The *Cartoon Man*, *Elephant* and *Dragon* characters use linear-blend skinning rigs designed by a rigging artist, which expose their joint angles and bone scales as rig parameters. Since the analytic formulas for these rigs are available as instances of Equation (3.62) from Section 3.4.3, posing them using our method is achieved in under 400ms. We used the first of the two described correspondence procedures for *Elephant* and *Cartoon Man* with the *stick-figure* abstraction, and the second procedure for *Dragon* and *Cartoon Man* with the *outline* representation.

The *Face* character uses a complex combination of blendshapes, bone transformations, nonlinear bend and twist deformers, and Maya expressions. Our system can nonetheless work with this complex rig by computing the

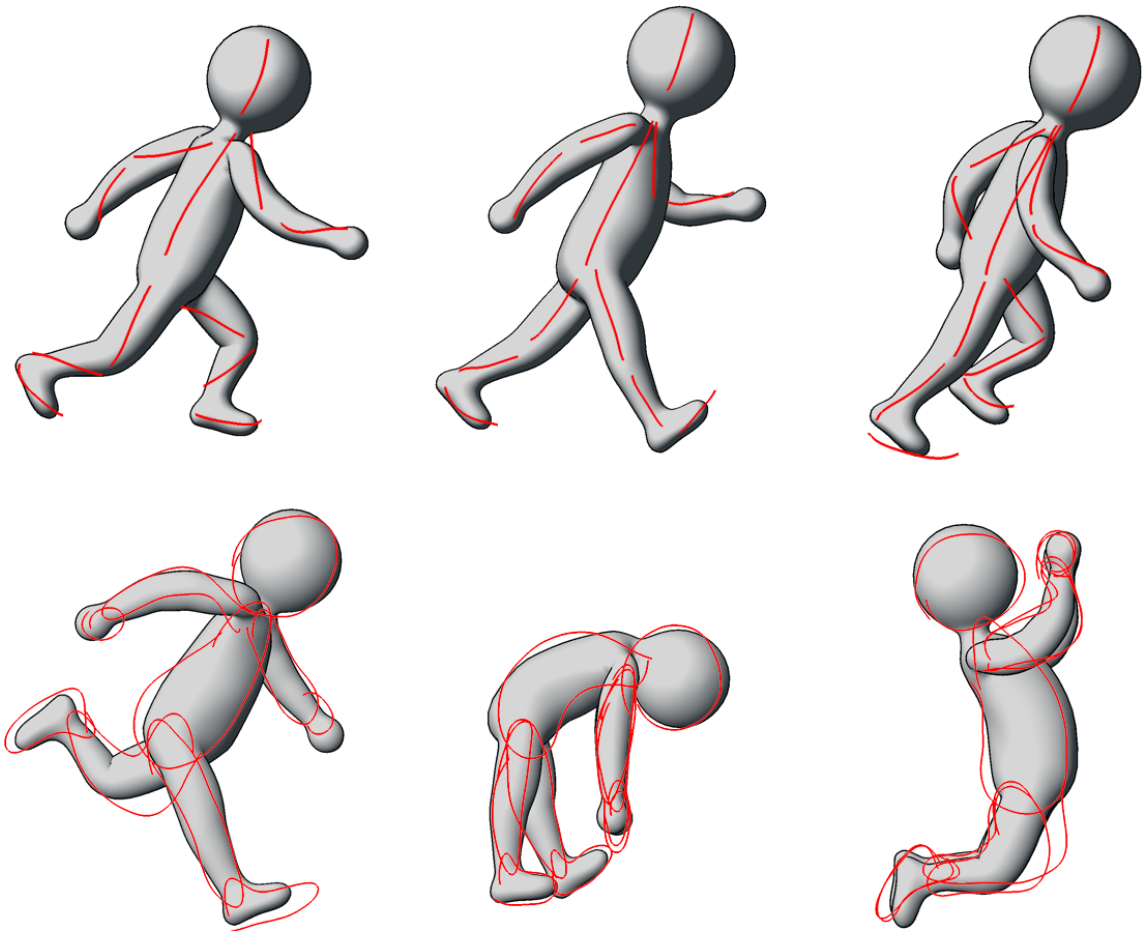

Figure 5.2: *Example poses for the* Cartoon Man *character using the* stick-figure *abstraction (top) and the* outline *abstraction (bottom).*

necessary derivatives using finite differences, albeit with a slower runtime of 9s. Profiling reveals that 95% of the computation time for the *Face* character is spent within Maya on rig evaluations for finite differences estimations, which suggests that significant speedups similar to the linear-blend skinning case would be possible if the rigging system provided analytic derivatives. We also used the first of the two described correspondence procedures for the *Face* example.

5.3.1 Redraw Posing

For this first application, we assume that the rigging artist designed one or more sketch abstractions for the character, providing us with a complete extended rig mapping. Our *Cartoon Man* character features two such representations for the user to choose from:

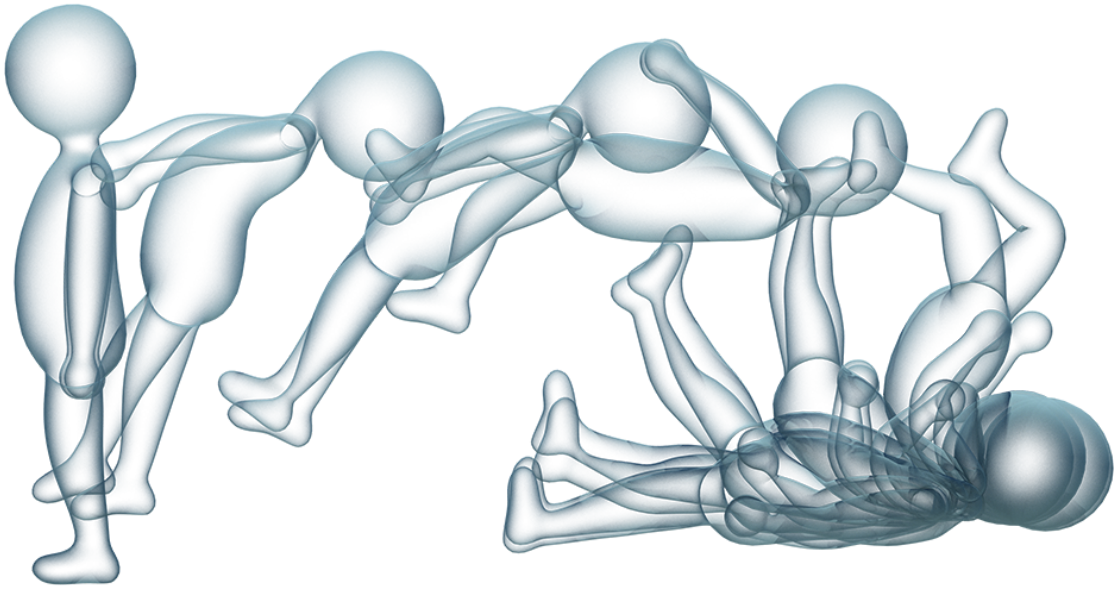

Figure 5.3: Cartoon Man *falling animation created with our method.*

- a *stick-figure* representation that closely follows the bones of the character's skeleton, and
- an *outline* representation skinned to the character's surface.

We created several expressive poses from crude stick-figure and outline sketches that can be seen in Figure 5.2, as well as a walking animation shown in Figure 5.1. Further, an animation of the *Cartoon Man* falling backwards created using our system can be found in Figure 5.3. To facilitate the correspondence computation, we require the user to draw the different curves in a prescribed order, which improves the runtime of the algorithm roughly by a factor of two because the solver converges faster. Given a few more ICP iterations, we observed results of similar quality even when the drawing order was not prescribed. Because our system works with normal character rigs within Maya, it enhances existing animation workflows while still allowing artists to adjust rig parameters directly. For the falling animation, the artist made a few manual fine-scale adjustments to the spine joint angles for two of the poses.

5.3.2 Character Individualization

In this application, we use our posing system to accommodate *sketch-based character individualization*, allowing novice users to design their own virtual character from simple sketches using predefined adaptive model parts. Our

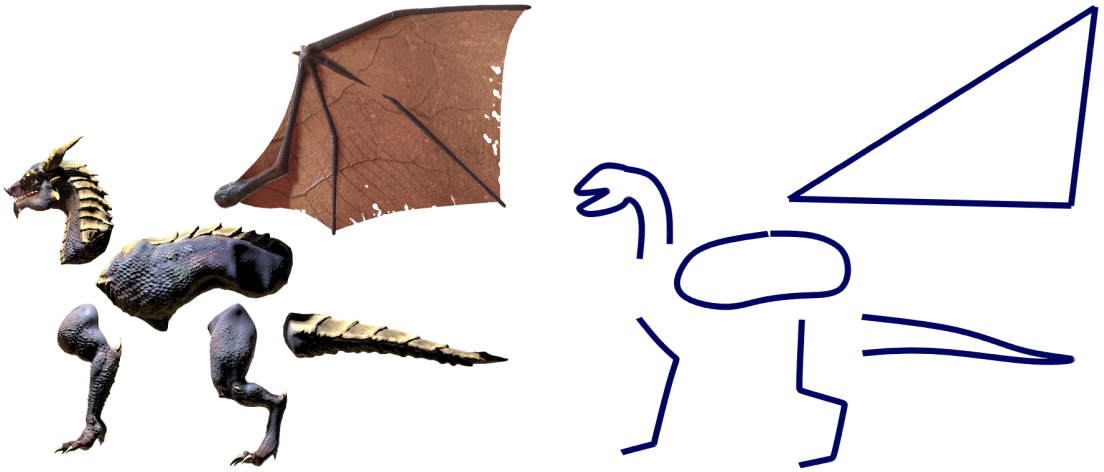

Figure 5.4: *Our database of six dragon body parts (left) and their corresponding sketch abstractions (right).*

work enhances other efforts toward character individualization [Hecker et al., 2008] by incorporating sketch-based control as well as more generalized shape customization, which is based on the underlying support of our method for sketch abstractions on top of arbitrary character rigs. To demonstrate this concept, we created a small database of six dragon body parts modeled and rigged by an expert artist: *Torso*, *Wings*, *Head*, *Tail*, *Front Legs* and *Hind Legs*. Each part also contains an embedded sketch abstraction that additionally serves as an identifier to perform the classification of the drawn part. To allow the parts to match a broad range of user-drawn shapes, the rigs expose the local scaling parameters of the underlying bones used to skin the models in addition to typical posing controls such as joint translation and rotation.

Instead of imposing a drawing order or requiring the user to specify which body part he or she is drawing, we use a database of 2D sketches containing several example instances of each body part together with a state-of-the-art category recognition approach [Eitz et al., 2012] to classify each drawn stroke. We then instantiate the detected part and optimize for both its shape and pose parameters simultaneously to place it into the scene. After exploring the current model in 3D, the user can choose to either redraw the stroke and repeat the fitting of the part, or continue to add new parts to the model.

We used our system to create a variety of dragons with different body shapes and numbers of extremities. We also allow simple animations of the created models by offsetting the matched rig parameters $\mathbf{p}$ with time-varying values from predefined animation curves provided for each body part. The differ-

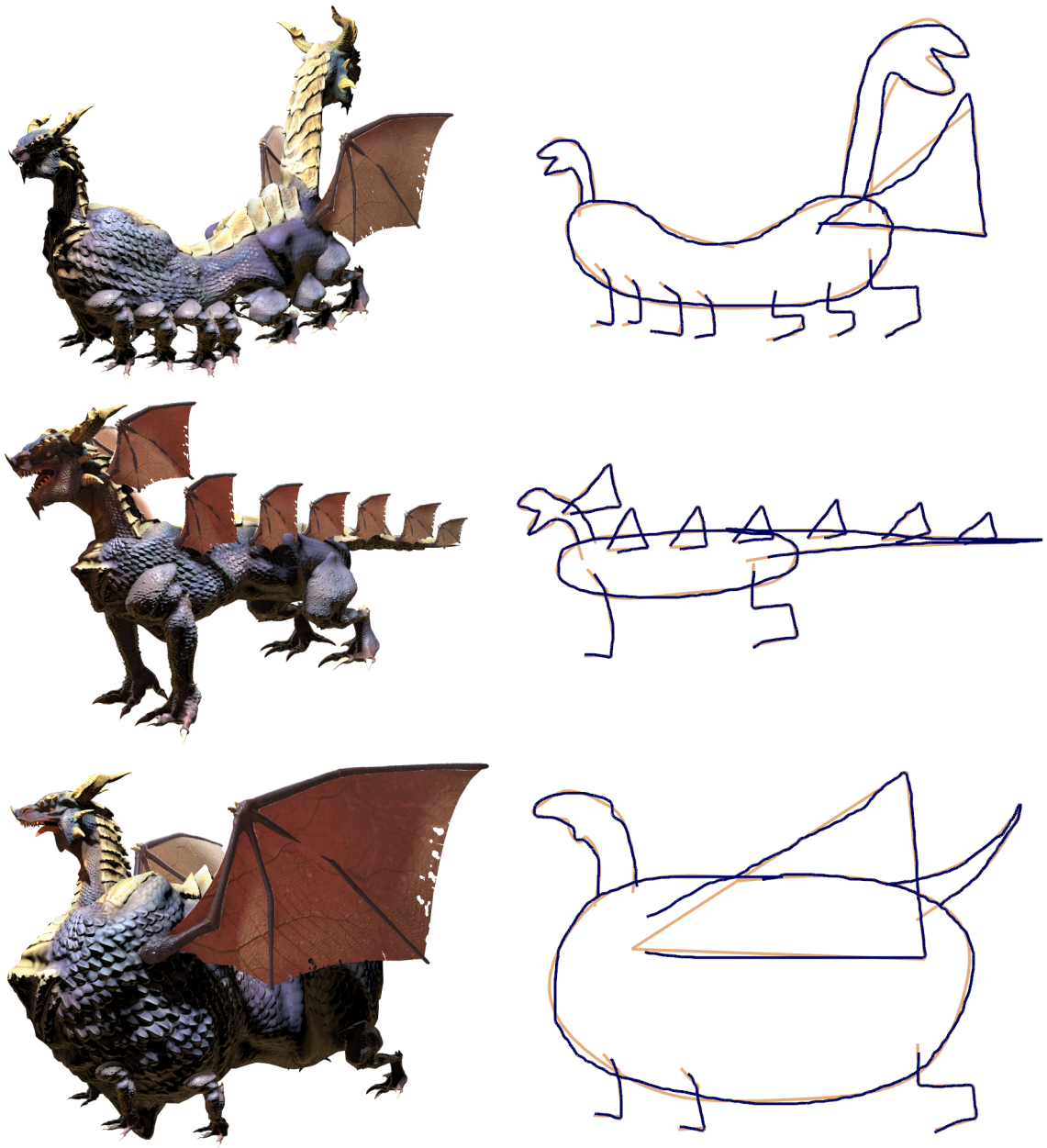

Figure 5.5: *Results for the dragon individualization application: 3D results (left), the corresponding input sketches (right, blue), and the optimized 2D representations (right, yellow).*

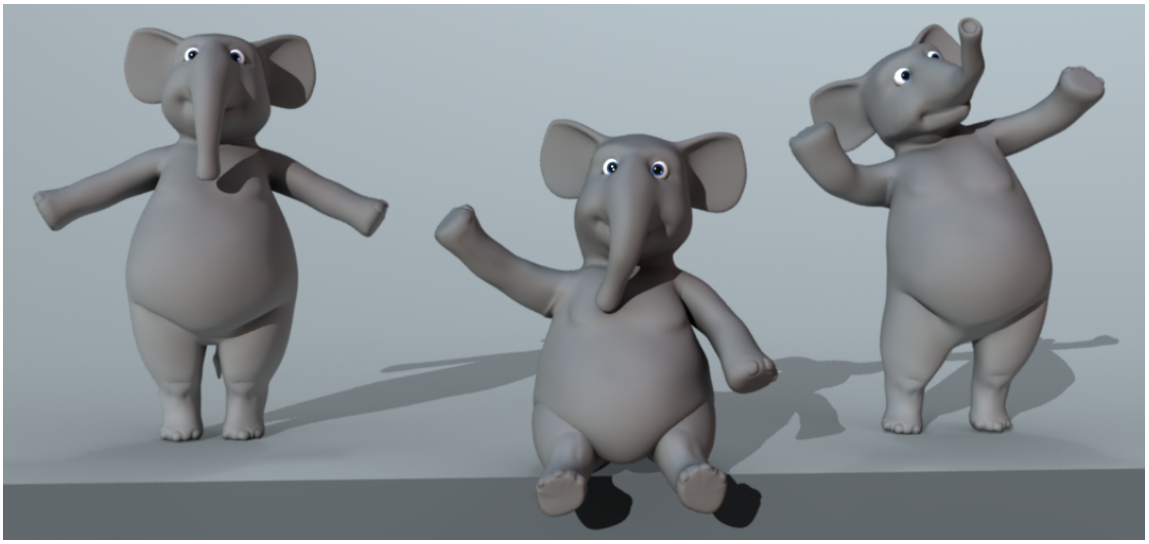

Figure 5.6: *The rest pose of the* Elephant *character (left) was transformed into two different poses using our draw-over posing system.*

ent dragon parts and their corresponding sketch abstractions are shown in Figure 5.4. A few selected dragon character examples created in our system with their corresponding input sketch drawings and the optimized sketch abstractions are shown in Figure 5.5.

5.3.3 Draw-Over Posing

In a third application, we enable *draw-over posing* of characters without predefined sketch abstractions. We first let the user create a custom abstraction by drawing a stroke onto the character mesh, and then match it to a second drawn stroke, effectively letting him or her choose what the best sketch representation is for the desired deformation. To demonstrate our draw-over posing system, we applied it to two different character rigs: *Elephant* and *Face*. The former is a complex production-quality linear blend skinning rig based on a skeletal structure with 49 joints. As a case in point, we downloaded the latter rig from the internet. It is a facial rig that features a large variety of complex controls combining various rigging techniques [Baskin, 2014]. Even without understanding how this rig works or which controls are available, we were able to successfully use our method to pose it, showcasing the ability of our system to extend to arbitrary deformers. Some of the resulting poses of experimental posing sessions for both of these rigs are shown in Figures 5.6 and 5.7.

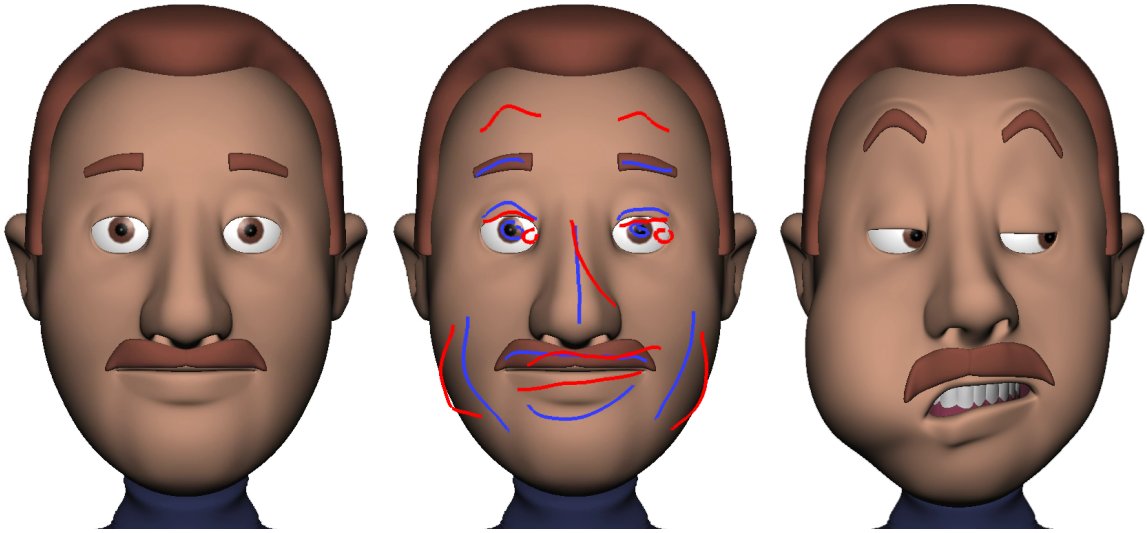

Figure 5.7: *The* Face *rig (left) was posed with 11 stroke pairs (middle) using our system to produce an expressive shape (right). Many thanks to Jason Baskin for making this rig freely available online [Baskin, 2014].*

5.4 Summary and Outlook

In this chapter, we proposed a sketch-based posing system for rigged 3D characters that allows artists to create custom sketch abstractions on top of a character's actual shape. A sketch abstraction is composed of rigged curves that form an iconographic 2D representation of the character from a particular viewpoint. When provided with a new input sketch, our optimization system minimizes a nonlinear iterative closest point energy to find the rigging parameters that best align the character's sketch abstraction to the input sketch. A custom regularization term addresses the underconstrained nature of the problem to select favorable poses. Although our system supports arbitrary black-box rigs, we showed how to optimize computations when rigging formulas and derivatives are available. We demonstrated our system's flexibility with examples showing different artist-designed sketch abstractions for both full body posing and the customization of individual components of a modular character. Finally, we showed that simple sketch abstractions can be built on the fly by projecting a drawn curve onto the character's mesh. Redrawing the curve allows the user to dynamically pose the character. Taken together, our system enables a new form of intuitive sketch-based posing in which the character designer has the freedom to prescribe the sketch abstraction that is most meaningful for the character.

While we will lay aside the concept of rig-space optimization in the next chapter about *Subspace Clothing Simulation*, we will make different use of the

deformation space that a character rig provides. Further, we will continue to employ analytic linear blend skinning rigs as a powerful tool to obtain the maximal performance for our simulation systems. Rather than using the analytic version of the rig's Jacobian with respect to its rig parameters, we will incorporate the rig rest state Jacobian to embed an object's deformation in the unrotated space provided by the rig's skeletal bone hierarchy.

CHAPTER 6

Subspace Clothing Simulation

Figure 6.1: *Example result of our method: A close-fitting sweater exhibits wrinkles and torsional folds under the effects of gravity and as the underlying torso is twisting. This example used only 12 adaptively chosen basis vectors and ran 18 times faster than a full simulation.*

6.1 Overview

In the previous chapters, we have seen how the concept of a rig's deformation space can be used to formulate optimization problems that operate directly in an environment that artists can create and control themselves. The resulting methods fit nicely into the workflow of standard ani-

mation pipelines, but they also implicitly assume that all possible deformation needs to be accounted for a priori at the rigging stage before motions or poses are created. Since the solvers are restricted to finding solutions inside the rig subspace, neither the rig-space physics simulation system nor the sketch-based posing system are able to produce poses that reach beyond what the rigging artist envisioned and embedded in the rig as controls. While this was a design choice rather than a limitation of these methods, there are other use cases for physical simulations where such a restriction of the possible deformations in advance is undesirable. One such scenario is the simulation of cloth in the form of a garment worn by a virtual character. While physics-based simulation is a well-explored field and tools for cloth are readily available, they are known for being notoriously slow since even single simulation frames require dozens of seconds of computation. In this chapter, we will explore the use of subspace methods in the context of clothing simulation to obtain increased runtime performance over existing full-space simulations.

Our method relies on two key insights. First, we note that subspace simulation is effective only when it can take advantage of structure in the simulation. And, although the movement of free-flowing *cloth* is largely arbitrary and unstructured, the movement of *clothing*—especially close-fitting clothing—does indeed contain a great deal of structure. We thus employ a kinematic deformation model as reference state that takes advantage of the structure of clothing simulation by capturing rotations that would otherwise prevent the use of a linear subspace—ill-suited for modeling rotational motions—from succeeding. Our second insight is that the rich deformations seen in clothing cannot be reproduced adequately with a global, low-dimensional basis. However, around a particular pose, the local space of deformations is much lower-dimensional. This observation motivates an adaptive, pose-dependent basis and allows our system to represent a broad range of complex wrinkles and folds while maintaining a small set of active basis vectors at any point during the simulation.

With these core insights in hand, we present a subspace approach to clothing simulation that uses a dynamically updated subspace basis in order to best reflect the deformations around the current pose. Our prototype implementation improves the performance of state-of-the-art cloth simulation codes by a factor of up to 22 while still reproducing the rich set of of wrinkles and folds evident in the full-space solution.

6.2 Method

We will begin by briefly outlining the pipeline of our method in Section 6.2.1, before introducing the central concept and proposing a possible design of the pose-space database in Section 6.2.2. After these descriptions, we will formally state all of the necessary ingredients for our adaptive subspace simulation method in Section 6.2.3.

6.2.1 Pipeline

As input, we expect a linear blend skinning (LBS) rig for the character, including a rigid skeleton, an undeformed surface mesh, and a set of skinning weights that determine how the surface mesh deforms according to the pose of the character. Furthermore, we assume that there are animation sequences provided that are representative of the typical motion that is expected during animation. In a production environment, these animation clips could correspond to the calisthenics sequences that are typically set up for testing the rig. Finally, we expect geometry for the clothing to be provided and pre-positioned in a way that fits the neutral pose of the character. Using the input rig and the pre-positioned clothing for the neutral pose, we construct a kinematic deformation model for the clothing that will serve as the reference state during subspace simulation.

Given this input data, our pipeline follows the following stages, which are also visualized in Figure 6.2:

- **Training Stage**: For each of the input animations, we perform a full-space cloth simulation. The resulting cloth configurations are associated with the corresponding poses of the character. Typically, these animations will lead to multiple cloth configurations for the same point in pose space. For example, this will always be the case when the same animation is run at different speeds. But also simple motion like bending and straightening an arm will generally lead to different cloth configurations due to collisions, friction, and the overall nonlinear nature of clothing.
- **Pose-Space Database Construction**: The training stage provides us with a data structure that holds all simulation results associated with their corresponding points in pose space. However, we eventually need a data structure that provides us with a subspace basis for any point in pose space. To this end, we cluster the simulation data in

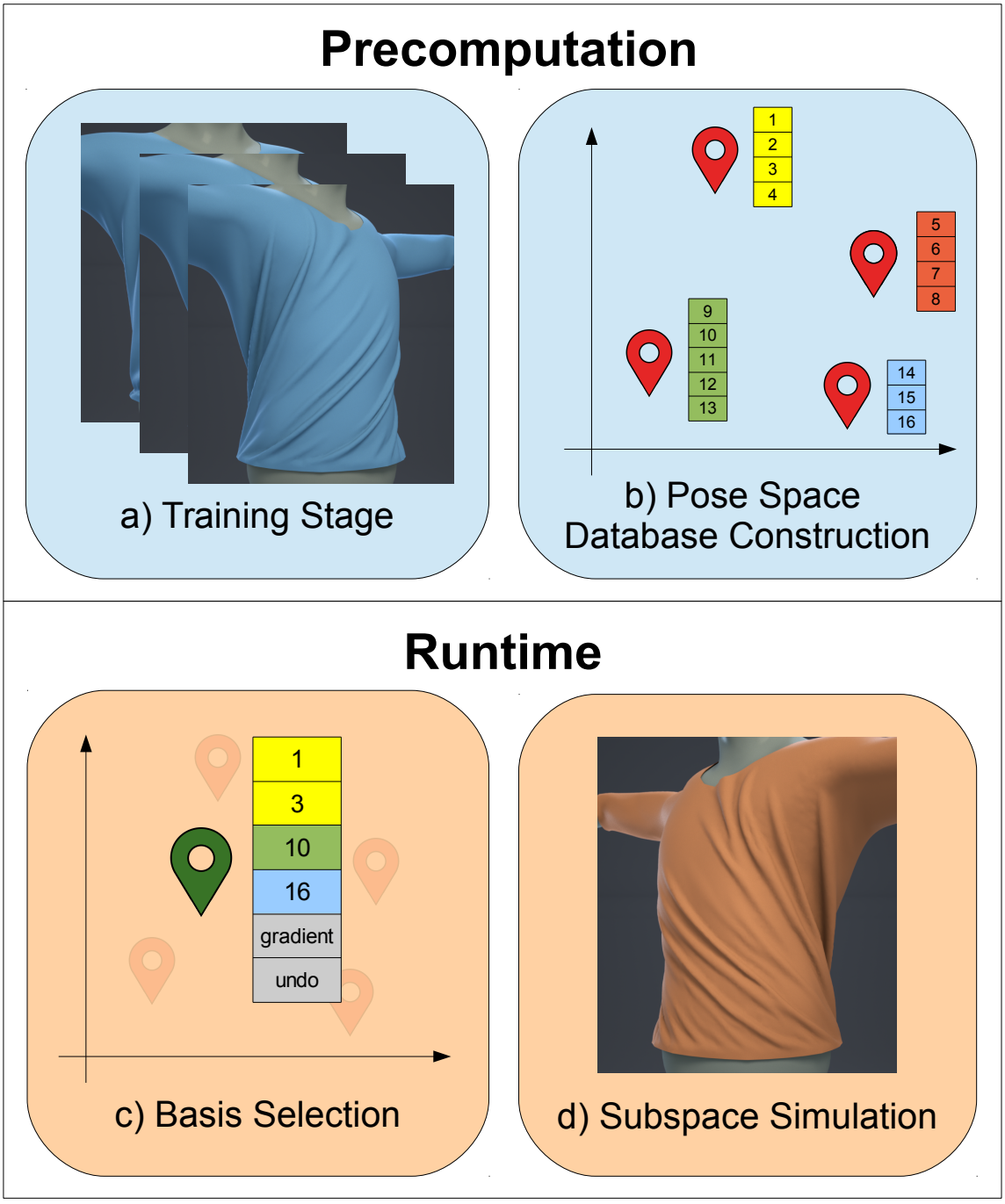

Figure 6.2: *Overview of our pipeline: (a) We perform full-space cloth simulation to create training data — shown in blue. (b) We construct a Pose-Space Database — a simplified 2D view with four sites and 16 basis vectors in total is illustrated. (c) At runtime, we adaptively select vectors from the database to obtain a basis for the current pose — in this case two from the yellow site, one from the green one, one from the blue one, plus gradient and undo vector. (d) We solve the subspace simulation problem using the selected basis — shown in orange.*

pose space and perform PCA on each cluster. We keep the most important modes and associate them with the pose-space point corresponding to the center of the cluster. We refer to such a database entry as a *site*.

- **Basis Selection** and **Subspace Simulation**: For each step of the subspace simulation, we want to retrieve a basis according to the current pose of the character and the current state of the clothing. Since the character's pose will generally not coincide with any of the sites, we need a way to construct the set of basis vectors from its surrounding sites. Our approach is to select basis vectors from neighboring sites considering their distance in pose space and how well they match the current dynamic state of the clothing.

6.2.2 Pose-Space Database

As one of the key concepts of our approach, the pose-space database (PSDB) is a data structure that holds bases within sites distributed across pose space. During subspace simulation, the PSDB is responsible for providing a low-dimensional basis that describes the behavior of the clothing around a given current pose. There are a number of questions that we must answer in order to design such a PSDB. We must find an adequate pose-space parameterization, we have to determine how to associate data with pose, and we have to decide which information to extract and store from the large amount of training data.

Relation between Pose and Clothing Deformation

A central question that influences subsequent design decisions is whether we should assume a locality relation between the character's pose and the deformation of its clothing. Clearly, the deformations induced by moving the shoulder are most significant around the shoulder—but how far do they extend? Wang and colleagues [2010] segment the clothing mesh into parts according to the body joints and assume that each joint influences only the two clothing segments adjacent to it. While this approach has the advantage that localized data can be stored per joint, a disadvantage is that far-reaching deformation effects such as the torsional folds at the waist that arise when moving the shoulder—which can be seen in Figure 6.1 or by self-experimentation—are not captured. We consider such effects to be essential for our target application and therefore assume that each change in pose can potentially induce deformations everywhere in the clothing. Consequently,

we associate deformation data with character poses represented by points in pose space.

Our method is based on the key observation that tight-fitting clothing typically exhibits motions very similar to those of the character wearing the garment. Thus, if we can extend the character's pose to a neutral state of the cloth model, this can serve as an initial reference state for our subspace simulation. To this end, we extend the skinning of the underlying linear blend skinning rig of the character model to the cloth mesh to build a *kinematic cloth reference*, which we will describe in more detail in Section 6.2.3.

Pose-Space Parameterization

We assume a linear blend skinning rig as input, which we parameterize by joint angles and bone scales as described earlier in Section 3.4.3. The natural way of representing the pose of the character is thus by a rig parameter vector holding those angle and scale values for each joint. However, this approach is not without problems when applied to our task of building a database for two reasons:

- First, the number of angles required to describe the pose of a character is typically quite large—already 57 for our *Torso* model. Sampling in such a high-dimensional space is impracticable since, unless there are truly redundant dimensions, the likelihood that two samples are far away from each other is very high—a phenomenon known as the *curse of dimensionality*.
- For another, many rigs exhibit redundant controls around the clavicle and the shoulder that can be convenient for an artist but problematic in our context when associating deformations with poses.

An alternative approach is to measure the distance between two poses by the amount of *induced difference in geometry* of the corresponding meshes. This approach allows for more intuition as to what distance means, and it avoids the problem of redundancies. We perform a Principal Component Analysis (PCA) on all the posed meshes in the training data, truncating the basis according to significance in singular values, and normalizing the dimensions to have equal variance. This results in a transformation to a coordinate space where the $L2$ distance is such a desired measure.

Data Generation and Model Reduction

We perform simulation runs for each of the input animations and associate the clothing shape for each time step with its corresponding character pose. Storing such a massive amount of data is neither practical nor useful and we would prefer a form which captures the diversity of deformations in a concise way. Principal Component Analysis with truncation is a natural candidate for this purpose as it provides a principled way to balance between the captured variability of the data and the dimensionality of the approximation. Since PCA on the training data produces global deformation modes, the method of Neumann and colleagues [2013] could alternatively be used to create localized basis vectors.

We observed that even performing a global PCA often results in basis vectors with small magnitudes for large areas of the model. Since this is sufficient for our subspace simulation setting, we choose this simple approach over more involved localized ones. Also, rather than constructing a single basis from all training data, our approach relies on many separately computed bases distributed across pose space. In this way, we can exploit the fact that deformations can be approximated with a low-dimensional basis when the current pose is close to a known one, but still account for the large variability of deformations across pose space.

Basis Creation

In order to construct a set of distributed bases, we must decide how many sites with corresponding local bases to use, where to put them, and what data to use for each site. We note that, once the number of sites is determined, the question of where to place the sites and which data points to associate with them is answered in a natural way using Voronoi partitioning. We found that two to three sites per input animation work well when using calisthenics-like input sequences that cycle through one specific motion. We then run the k-means algorithm in order to compute the desired number of clusters, obtaining the pose-space locations of the sites and the sets of data belonging to them.

Before we can analyze the clusters, there is one more transformation that needs to be applied to the data. If we were to perform PCA directly on the geometry of the clothing as returned by the full-space simulator, the variability would be dominated by the motion of the body, i.e., rotations of its joints. However, we are not interested in changes that are captured by the kinematic model, we want to analyze how the clothing changes its shape

relative to the kinematic model. For this reason, we optimize for a displacement vector *prior* to the skinning transformation of the cloth's current kinematic reference.

Once this transformation is applied to all data points, we are all set for analyzing the data. For each site, we perform Principal Component Analysis on the full-space training example sets obtained from the clustering. The resulting basis vectors are then stored with their sites in the pose-space database, indexed by the site's corresponding pose-space location.

Data Retrieval

Once the PSDB is populated, the question is how to retrieve data at run time: Given a pose for the character, what data should be returned? Obviously, since the pose will generally not coincide with one of the sites, we cannot just return one of the bases but need a retrieval scheme for returning adequate inbetween data that reflects the influence of neighboring sites.

One possibility for a such a retrieval scheme would be to interpolate between the bases of different sites. This bears the promise that a smooth basis interpolation would translate into temporal smoothness for the simulation. Unfortunately, constructing a principled scheme for bases interpolation appears to be difficult. While interpolating pairs of vectors can be done easily, extending this concept to interpolating between sets of vectors—or whole bases—is difficult because the correspondence between vectors in different sets is unclear. But even without this problem, it remains questionable whether an interpolated basis is meaningful to begin with.

For these reasons, we settled for an approach that only uses basis vectors from the original sites. More concretely, when the database is queried with a given pose, we create a pool of candidate basis vectors that includes the vectors of all sites that are within a search radius in pose space from the current pose. The subspace simulator then selects a set of basis vectors from this candidate pool as explained in Section 6.2.3.

6.2.3 Adaptive Subspace Simulation

After motivating and discussing the different design choices for our system, we will now proceed to providing the mathematical details of our adaptive subspace simulation algorithm. To this end, we will once more pass over all the stages of the pipeline, while this time treating them in a much more formal way than previously.

Input

As briefly mentioned earlier, the input to our method consists of the following components:

- A linear blend skinning rig of a virtual character as described in Section 3.4.3, including surface geometry of the character, a joint hierarchy, as well as a skinning weight matrix $\mathbf{W}$.
- An undeformed cloth mesh $\mathbf{X}$ corresponding to the neutral pose $\mathbf{P}$ of the character.
- A sequence of l rig parameter configurations $\mathbf{e}^1, \mathbf{e}^2, \ldots, \mathbf{e}^l \in \mathbb{R}^r$ to be used as training poses.

We begin by simulating the cloth mesh using the character's surface mesh as collision body for each of the l frames, resulting in as many clothing deformations $\mathbf{f}^1, \mathbf{f}^2, \ldots, \mathbf{f}^l \in \mathbb{R}^{3n}$ for each of these poses. To perform these unreduced simulations in full space, we use the physical model introduced earlier in Section 3.1 with the elastic cloth model from Section 3.1.2 to provide stretching and bending energies. We perform the minimization of the physical energy as described in Section 3.2, while using the impulse-based contact model from Section 3.5.3 to resolve both self-collisions within the cloth as well as contact with the underlying character mesh. A few selected training poses of the clothing simulated in full space can be seen in Figure 6.3.

Kinematic Cloth Reference

The basis of our subspace simulator is a kinematic cloth reference, which we construct by extending the skinning transformation of the linear blend skinning rig onto the clothing. In the neutral pose $\mathbf{P}$ of the character, we find for each vertex $\mathbf{X}_i$ of the pre-positioned clothing the closest point on the character's surface. To instantiate Equation (3.62) from Section 3.4.3 for the cloth, we then determine skinning weights $\mathbf{W}^R$ corresponding to the bones of the linear blend skinning rig using barycentric interpolation of the character's original skinning weights $\mathbf{W}$. As this process might introduce stair-stepping artifacts when the cloth is not very tightly fitting the character, we run a few iterations of Laplacian smoothing to avoid the hard discontinuities. In more complicated cases, an example-based automatic skinning approach such as the one by James and Twigg [2005] could be employed.

Rather than looking at the linear blend skinning transformation as a function of the rig parameters $\mathbf{p}$ like we did in earlier chapters, we will rewrite it as

Figure 6.3: *Sample frames of three of the training sequences we used as input for our method. From top to bottom:* Arms Bend, Twist, Plane.

a function of an *untransformed state* $\check{\mathbf{x}}$ that replaces the neutral mesh $\mathbf{X}$ in Equation (3.62). The kinematic reference of the current pose in then given as

$$\mathbf{r}\left(\check{\mathbf{x}}\right)_i = \varphi_{\text{LBS}}\left(\check{\mathbf{x}};\mathbf{p}\right)_i = \begin{bmatrix}\mathbf{I}_3 & \mathbf{0}\end{bmatrix} \sum_{j=1}^{m} \left(\mathbf{W}^R\right)_{ij} \cdot \mathbf{C}_j(\mathbf{p}) \hat{\check{\mathbf{x}}}_i, \tag{6.1}$$

where we consider $\mathbf{p}$ to be a metaparameter of the LBS function φ_{LBS}, and where $\hat{\check{\mathbf{x}}}_i$ denotes the untransformed vertex $\check{\mathbf{x}}_i$ lifted to homogeneous 4D space. Unlike the original LBS formulation, the kinematic reference in Equation (6.1) is an invertible function.

Pose-Space Parameterization

To construct a pose-space parameterization, we begin by evaluating the cloth skinning for each of the poses and assembling them as:

$$\mathbf{S} = \left[\varphi_{\text{LBS}}\left(\mathbf{X};\mathbf{e}^1\right) \;\middle|\; \varphi_{\text{LBS}}\left(\mathbf{X};\mathbf{e}^2\right) \;\middle|\; \ldots \;\middle|\; \varphi_{\text{LBS}}\left(\mathbf{X};\mathbf{e}^l\right)\right] \in \mathbb{R}^{3n\times l} \tag{6.2}$$

Following the steps of Principal Component Analysis, we first subtract the mean column $\bar{\mathbf{s}}$ of $\mathbf{S}$ from all columns to obtain $\bar{\mathbf{S}}$, and then then perform a Singular Value Decomposition (SVD) on the matrix $\bar{\mathbf{S}}$ to obtain:

$$\bar{\mathbf{S}} = \underbrace{\mathbf{U}}_{\in\mathbb{R}^{3n\times 3n}} \quad \underbrace{\boldsymbol{\Lambda}}_{\in\mathbb{R}^{3n\times r}} \quad \underbrace{\mathbf{V}^T}_{\in\mathbb{R}^{r\times r}} \tag{6.3}$$

Truncating the all except the first u columns of $\mathbf{U}$ results in a projection matrix $\tilde{\mathbf{U}} \in \mathbb{R}^{3n\times u}$ that we will use as *pose-space parameterization*:

$$\mathbf{z}(\mathbf{x}) = \tilde{\mathbf{U}}^T\mathbf{x} \in \mathbb{R}^u \tag{6.4}$$

We used $u = 4$ for all of our examples, but a suitable value could also be automatically obtained by considering the magnitudes of the singular values in $\boldsymbol{\Lambda}$.

Subspace Cloth Model

Unlike the applications in the previous chapters, we are not interested in a rig-space projection as described in Section 3.3. Rather, we are looking to minimize the physical energy in a linear subspace with respect to a reduced coordinate vector $\mathbf{q} \in \mathbb{R}^r$. Using a subspace basis encoded by a matrix $\mathbf{A} \in \mathbb{R}^{3n\times r}$ containing the basis vectors in its columns, a full-space *displacement vector* can be computed from the subspace coordinates as:

$$\mathbf{u}(\mathbf{q}) = \mathbf{A}\mathbf{q} \tag{6.5}$$

The simplest approach to now compute the full-space cloth vertex positions $\mathbf{x}$ would be to directly add the displacements in world space as:

$$\mathbf{x}(\mathbf{q}) = \mathbf{X} + \mathbf{u}(\mathbf{q}) \tag{6.6}$$

The disadvantage of the formulation in Equation (6.6) is that all displacements are computed with respect to a single neutral mesh $\mathbf{X}$, and thus none of the pose information is taken into account. We can improve the model to incorporate this information by basing the displacements on the current pose of the cloth rig:

$$\mathbf{x}(\mathbf{q}) = \varphi_{\text{LBS}}(\mathbf{X}; \mathbf{p}) + \mathbf{u}(\mathbf{q}) \tag{6.7}$$

While the formulation in Equation (6.7) improves the situation, the subspace coordinates $\mathbf{q}$ still result in world-space displacements $\mathbf{u}$. However, a world-space displacement is only meaningful for the pose at which it was computed—rotations induced by the rig by would immediately invalidate the displacement. We therefore choose to compute *untransformed displacements* $\check{\mathbf{u}}$ with respect to the untransformed setting, and then make use of the kinematic cloth reference from Equation (6.1) to compute

$$\mathbf{x}(\mathbf{q}) = \varphi_{\text{LBS}}\left(\underbrace{\mathbf{X} + \check{\mathbf{u}}(\mathbf{q})}_{\check{\mathbf{x}}(\mathbf{q})}; \mathbf{p}\right) = \varphi_{\text{LBS}}(\check{\mathbf{x}}(\mathbf{q}); \mathbf{p}) = \mathbf{r}(\check{\mathbf{x}}(\mathbf{q})), \tag{6.8}$$

where $\check{\mathbf{x}}$ denotes the untransformed clothing state. The formulation from Equation (6.8) has the advantage that a displacement vector for a given pose will look plausible throughout a certain region in pose space.

Site Creation

Pose-Space Clustering To create the sites for our pose-space database, we first obtain their location by clustering the pose-space positions of all training poses. Using the pose-space parameterization from Equation (6.4) and the skinned cloth matrix from Equation (6.2), we compute the positions of the training poses in pose-space as:

$$\mathbf{z}^i = \mathbf{z}((\mathbf{S})_i) = \mathbf{z}\left(\varphi_{\text{LBS}}\left(\mathbf{X}; \mathbf{e}^i\right)\right) = \tilde{\mathbf{U}}^T \varphi_{\text{LBS}}\left(\mathbf{X}; \mathbf{e}^i\right) \tag{6.9}$$

We then run the k-means algorithm on the l parameters $\mathbf{z}^i$ in order to compute a desired number of k clusters with centroids that we will denote as $\mathbf{s}^j$

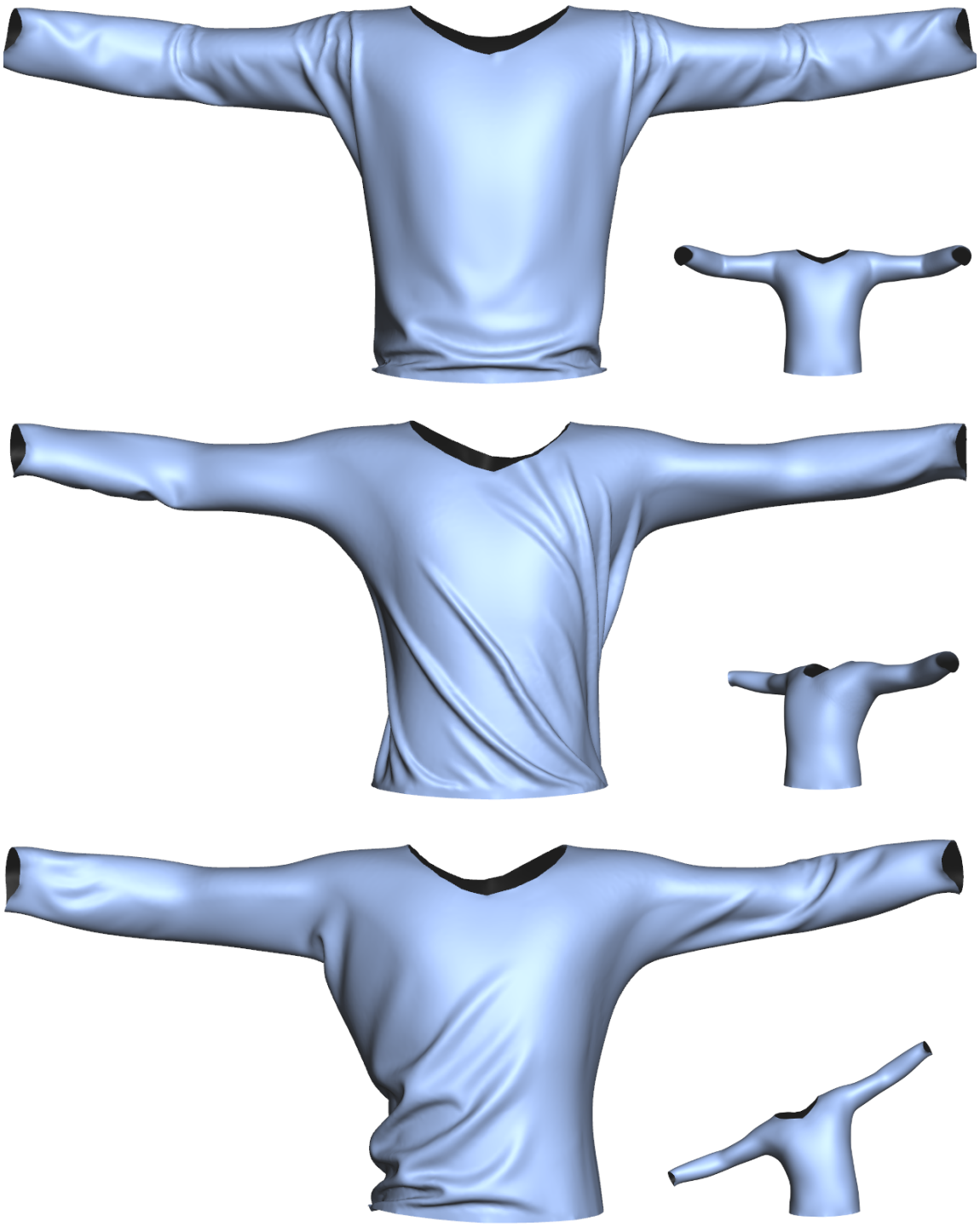

Figure 6.4: *Using the inverse kinematic cloth reference, we untransformed the three full-space deformation frames from Figure 6.3 to the neutral pose of the rig. The small images on the bottom right of each deformation show the "subtracted" rig deformation.*

and use as descriptors of the different sites. Further, each full-space training cloth deformation $\mathbf{f}^i$ is assigned to the site with the closest descriptor $\mathbf{s}^j$ to its parameterized pose $\mathbf{z}^i$ based on $L2$ distance. The training set $\mathcal{D}_j$ of the site j is then given as:

$$\mathcal{D}_j = \left\{ \mathbf{f}^i \;\middle|\; j = \arg\min_{\hat{j}} \left\| \mathbf{z}^i - \mathbf{s}^{\hat{j}} \right\|_2 \right\} \tag{6.10}$$

Basis Vector Generation For each site j of the pose-space database, we untransform the full-space cloth deformations in its training set $\mathcal{D}_j$ using the inverse of the kinematic cloth reference from Equation (6.1). Figure 6.4 shows the effect of this transformation for the three full-space cloth deformations previously seen in Figure 6.3. Despite those three deformations being assigned to different sites by our pose-space clustering algorithm, the untransformation successfully brings them to a common space independent of the site they originate from and removes all rotational deformation components due to the character's skeletal motion.

We then perform Principal Component Analysis on the untransformed cloth deformations of the training set $\mathcal{D}_j$. This is carried out analogously to the PCA for the pose-space parameterization in Equation (6.3), except that we do not use a fixed number of columns to truncate the respective SVD factor $\mathbf{U}$. Instead, we truncate the basis when the ratio between the corresponding singular value and the largest singular value drops below a given threshold value ε_{PCA}. We used $\varepsilon_{\text{PCA}} = 0.01$ for all of our examples. The resulting basis vectors in the columns of the truncated respective $\tilde{\mathbf{U}}$ are then stored with the site in the pose-space database, indexed by the corresponding pose-space location $\mathbf{s}^j$ of the site.

Subspace Optimization

System Assembly In every time step, our subspace simulation system solves the following optimization problem:

$$\mathbf{q}^{t+1} = \arg\min_{\mathbf{q}} H(\mathbf{r}(\check{\mathbf{x}}(\mathbf{q}))) \tag{6.11}$$

To minimize the objective function in Equation (6.11), we again use the subspace simulation framework described in Section 3.3. Rather than an artist-created rig, the kinematic reference $\mathbf{r}$ applied to the undeformed cloth configuration $\check{\mathbf{x}}$ determined by the unknown coefficients $\mathbf{q}$ takes up the role of

the rig in Equation (3.46). The Jacobian required to compute the reduced derivatives in Equation (3.48) and Equation (3.49) can be obtained using the chain rule as

$$\mathbf{J} = \frac{\partial}{\partial \mathbf{q}} \mathbf{r}\left(\check{\mathbf{x}}\left(\mathbf{q}^t\right)\right) = \underbrace{\frac{\partial}{\partial \check{\mathbf{x}}} \mathbf{r}\left(\check{\mathbf{x}}\left(\mathbf{q}^t\right)\right)}_{\mathbf{B}} \underbrace{\frac{\partial}{\partial \mathbf{q}} \check{\mathbf{x}}\left(\mathbf{q}^t\right)}_{\mathbf{A}} = \mathbf{B}\mathbf{A}, \tag{6.12}$$

where $\mathbf{B}$ is the rest state derivative of the linear blend skinning rig as defined in Section 3.4.3 by Equation (3.70).

The resulting reduced system of Equation (3.50) is dense but only of dimensions $r \times r$ instead of $n \times n$ for the full-space variant. For a typical example like the sweater, the dimension of the reduced space was $r = 12$, while the full-space clothing had $3n = 88530$ degrees of freedom. Indeed, the cost of solving the reduced system is negligible, but the assembly can take up a significant fraction of the overall computation time. The main contributors are the computation of the full-space Hessian $\mathbf{H}$ and the multiplications with the projection matrix $\mathbf{J}$.

Solver Optimizations In our experiments, we found that simulations in a linear subspace exhibit very good convergence and are generally much less susceptible to instabilities than in full space. In particular, we never encountered indefinite systems—a major struggle for full-space simulation—removing the need for line-search and expensive regularization altogether. Similar to the *Efficient Rig-Space Physics Simulation* application from Chapter 4, we also found that, even when reusing the same full-space Hessian over many times steps, stability was not affected and the visual impact on the simulation results was minimal. Our approach is therefore to keep the Hessian constant and only do one Newton step by default. Only if the norm of the current basis vectors projected onto the full-space gradient is still above a threshold σ, we recompute the full-space Hessian $\mathbf{H}$ and perform further steps. Choosing $\sigma = 0.1$ worked well for all our examples.

Basis Construction

Adaptive Basis Selection Our subspace integration algorithm selects a new set of basis vectors in every iteration of the Newton solver. We would like this basis to be low-dimensional, and we want it to capture the deformations that the clothing can undergo around the current character pose. Given the current pose, the pose-space database provides a pool of candidate vectors, typically much larger than the desired dimension of the subspace. We

Figure 6.5: *Using a fixed basis of 100 vectors for the* Torso Twist *sequence produces large jumps between different cloth poses. The three frames shown here were captured within a short duration of only 0.12 seconds.*

therefore select a subset of vectors according to how far away the site is from the current pose and how well a given vector fits the current configuration.

Each iteration of the Newton-Raphson algorithm solves the full-space linear problem $\mathbf{H}\Delta\mathbf{x} = -\mathbf{g}$ from Equation (3.35) projected into the current subspace. Clearly, if we can find a basis that spans the full-space solution $\mathbf{H}^{-1}\mathbf{g}$, we will be able to accurately solve the full-space linear problem in the subspace. How helpful is a given basis vector $\mathbf{a}_i$ for this purpose? Assuming that $\mathbf{H}$ is positive definite, we know that $\mathbf{a}_i$ and $\mathbf{g}$ must have a positive dot product for $\mathbf{a}_i$ to be a descent direction for the objective function H from Equation (3.31). Put differently, if this dot product is zero, $\mathbf{a}_i$ cannot help in solving the full-space system. This observation motivates a selection scheme that gives preference to basis vectors that are well-aligned with the gradient at the current configuration.

More concretely, when the database is queried with a given runtime pose $\mathbf{z}^t$, we create a pool of candidate basis vectors that includes the vectors of all sites that are within a search radius ρ in pose space from that pose. We typically set ρ to the average distance between the sites in the pose space. To let the subspace simulator select a set of basis vectors from this candidate pool, we then transform all basis vectors in the pool to world space using the kinematic reference and project them onto the full-space gradient $\mathbf{g}$. We use the resulting dot products as a score and pick untransformed versions of the r best ones to form our subspace basis $\mathbf{A}$. In order to give preference to vectors from sites close to the current pose, we additionally scale the score with the inverse distance before sorting. We note that there is no need to

Figure 6.6: *One frame of the* Torso Twist *sequence when simulated with only the gradient as basis (left), adaptive basis selection without adding the gradient to the basis (middle), and our method using the gradient in addition to the adaptive basis selection.*

maintain an orthogonal basis since potential redundancies do not cause any adverse effect.

In order to analyze the efficiency of our selection scheme, we compared it to using a fixed basis created from extracting 100 PCA vectors from the training data for simulation. We noticed that even though the fixed basis is able to reproduce deformations for many individual poses correctly, it tends to jump between them, making the resulting animation not smooth as can be seen in Figure 6.5.

Adding the Gradient Even though blindly following the negative gradient direction leads to poor convergence results for minimization problems in general, we observed that adding the full-space gradient $\mathbf{g}$ as a basis vector leads to significantly improved simulation results, as can be seen in Figure 6.6. Doing this guarantees that the solution to the subspace problem will always reduce the objective function H from Equation (3.35) while at the same time enlarging the range of possible deformations, which could be one possible explanation for this positive effect.

Incremental State Updates and Undo Vector A particular aspect of our approach is that we always start optimizing with the subspace coordinates $\mathbf{q}$ equal to zero as initial guess in each Newton-Raphson iteration, which leads to the step direction $\Delta\mathbf{q}$ being equivalent to the subspace coordinates $\mathbf{q}$. After solving and before starting the next iteration, we then update the full-

space configuration $\mathbf{x}$ with the subspace displacements $\mathbf{Aq}$. This becomes necessary because—as a notable difference to existing works—the full-space configuration is generally not in the span of the current subspace basis. Indeed, due to the large range of deformations observed in clothing, we find that it is impractical to restrict the full-space solution in this way while still obtaining the same degree of variability in clothing shape. However, when the basis changes in every iteration, it is not possible to change the component of the current state that is not in the span of the current basis. We solve this problem by always adding the difference vector $\check{\mathbf{x}} - \mathbf{X}$ to the basis, thus allowing the clothing to *undo* the solution of the previous step, if necessary.

Final Algorithm

Algorithm 5: *Subspace integration with adaptive basis.*

while further iterations necessary **do**
  update full-space state $\mathbf{x} \leftarrow \mathbf{r}(\check{\mathbf{x}})$
  $(\mathbf{H}, \mathbf{g}) \leftarrow$ COMPUTEFULLSPACESYSTEM($\mathbf{x}$)
  $\mathbf{A} \leftarrow$ CONSTRUCTADAPTIVEBASIS($\mathbf{g}$)
  $(\mathbf{K}, \mathbf{r}) \leftarrow$ COMPUTEREDUCEDSYSTEM($\mathbf{g}, \mathbf{H}, \mathbf{A}$)
  solve reduced system $\mathbf{Kq} = -\mathbf{r}$ for $\mathbf{q}$
  update untransformed state $\check{\mathbf{x}} \leftarrow \check{\mathbf{x}} + \mathbf{Aq}$
end while

The required steps to perform the simulation of one time step in our adaptive subspace setting are summarized in Algorithm 5. During subspace simulation, we always preserve the invariant that the current cloth deformation is the transformed version of the untransformed deformation $\check{\mathbf{x}}$ with respect to the linear blend skinning rig. COMPUTEFULLSPACESYSTEM comprises the computation of the full-space derivatives as described in Section 3.1.6, while only recomputing the Hessian $\mathbf{H}$ if necessary. CONSTRUCTADAPTIVEBASIS performs the adaptive basis selection as previously described in this section. COMPUTEREDUCEDSYSTEM uses the Jacobian from Equation (6.12) to project the full-space derivatives into the subspace as detailed in Equations (3.48) and (3.49) of Section 3.3.2. All of the steps in Algorithm 5 are understood to be performed with respect to the current rig configuration $\mathbf{p}^t$ of the underlying character.

6.3 Results

In this section, we present results of our adaptive subspace simulation method and compare it to full-space simulation. We further explore the ability of our method to generalize to motions not present in the training data.

6.3.1 Setup

For the *Torso* example, we first ran a state-of-the-art cloth simulation code on the *Shirt* mesh with 29,510 vertices and 58,660 triangles to generate around 10,000 frames of training data for our method. Some of the resulting deformations can be seen in Figure 6.3. Using this training data, we automatically created a pose-space database using 18 sites with 10 to 20 basis vectors each, resulting in a total number of 212 basis vectors. Four of the basis vectors are visualized in Figure 6.7 and are shown with respect to the pose of the site that they are associated with.

6.3.2 Validation

We first validated our subspace simulation method by running it on the same animations that were used to create the training data. One such example for the *Twist* animation can be seen in Figure 6.1. Using our fast subspace simulation method, we were able to obtain total runtime speedups of up to 22x over the full-space simulation. Comparing only the simulation time yields a speedup of up to 60x. The timings and speedups for all tested sequences of the *Torso* example are reported in Table 6.1. We also ran our pipeline on a second *Legs* rig wearing a pair of pants and obtained results similar to the *Torso* example. A sample frame of the *Legs Bend* sequence can be seen in Figure 6.8.

We observed no visual differences between the naïve subspace solver and the optimized solver as described in Section 6.2.3. Increasing the size of our adaptive basis also only has small effects on the resulting deformations when using the same pose-space database. Increasing the number of vectors per site in the pose-space database by reducing ε_{PCA} also has no noticeable effect unless combined with an increased adaptive basis size. While we observed slightly more dynamics in the formation of the wrinkles in the clothing, this gain comes at a significant computational cost.

Sequence	n_{frames}	**Full Space** t_{frame}	**Collisions** t_{frame}
Torso Twist	1701	13.20s	0.47s
Torso Arms Bend	1451	14.09s	0.41s
Torso Arms Up	1251	16.32s	0.45s
Torso Arms Down	1251	12.95s	0.40s
Torso Lean Back	601	12.68s	0.48s
Torso Lean Forward	601	12.59s	0.46s
Torso Plane	1501	12.41s	0.47
Torso Yo	1176	—	0.39
Torso Gym	4701	—	0.42

Sequence	n_{frames}	**Naïve Subspace** t_{frame}	tsp	ssp
Torso Twist	1701	1.11s	12x	20x
Torso Arms Bend	1451	0.95s	15x	25x
Torso Arms Up	1251	1.29s	13x	19x
Torso Arms Down	1251	1.17s	11x	16x
Torso Lean Back	601	1.40s	9x	13x
Torso Lean Forward	601	1.35s	9x	14x
Torso Plane	1501	1.23s	10x	16x
Torso Yo	1176	1.05s	—	—
Torso Gym	4701	1.07s	—	—

Sequence	n_{frames}	**Optimized Subspace** t_{frame}	tsp	ssp
Torso Twist	1701	0.71s	18x	52x
Torso Arms Bend	1451	0.63s	22x	60x
Torso Arms Up	1251	0.92s	18x	34x
Torso Arms Down	1251	0.89s	15x	25x
Torso Lean Back	601	0.82s	16x	37x
Torso Lean Forward	601	1.07s	12x	20x
Torso Plane	1501	0.90s	14x	28x
Torso Yo	1176	0.86s	—	—
Torso Gym	4701	0.82s	—	—

Table 6.1: *Timings and speedups for the* Torso *sequences we simulated on an Intel Core i7-3930K 6 x 3.2Ghz where* t_{frame} *is computation time per frame in seconds,* tsp *is the total speedup over the full-space simulation and* ssp *is the simulation speedup (not counting collision time) over the full-space simulation.* Yo *and* Gym *are generalization examples that did not appear in the training data, and we did not need to run the full-space solver for them.*

6.3.3 Generalization to Novel Poses

Even though we use global bases that deform all vertices of the cloth to construct our subspaces, many of the basis vectors tend to be sparse and local even though we do not ask for this explicitly. For this reason, different basis vectors from different sites can combine into new configurations that were not in the training data set. To demonstrate the ability of our method to generalize to new motions, we created two new animations that are not part of our training set and explore new portions of pose space. The *Yo* sequence simultaneously combines both twisting and arm bending, while *Gym* is a longer sequence that comprises a variety of upper body motions. Figure 6.9 shows that our method is able to generalize to these previously unseen animations and produce compelling folds and wrinkles.

In the more general case where no similar input motions are available at all from the training data, we cannot expect to produce deformations with detail close to what a full-space solver would provide. In those cases, our method transitions smoothly back to the skinned basis of our kinematic cloth reference—that is, the displacements $\check{\mathbf{u}}$ tend to zero. To demonstrate this, we created a simple cylindrical piece of cloth wrapped around a bending tube, but only simulated full-space training data for motions in the horizontal plane to create the pose-space database. When using our subspace simulation technique on a sequence that bends the tube vertically, our solver still converges and does not generate additional artifacts, as shown in Figure 6.10.

6.3.4 Application to Elastic Solids

Even though we presented our adaptive subspace simulation method for the computation of wrinkles and folds in tight-fitting clothing, the assumption that a physical deformation is closely related to a character's pose also holds for the flesh on its body. To investigate whether the proposed algorithm also extends to this case, we tried to apply it to an elastic solid model with a tetrahedral volume mesh of a character using the elastic energy formulation described in Section 3.1.2. However, while the kinematic reference described in Section 6.2.3 is sufficient to build a subspace model for simulating clothing, it is not yet applicable to elastic solids due to its lack of articulation. The simulated clothing such as the one in Figure 6.3 was implicitly articulated by contact with the surface of the underlying character wearing the garment. In contrast, all vertices of the character in the solid case are driven by the simulation and we need to take the skeletal input motion explicitly into account.

We achieve this by constraining all vertices within a radius δ of the character's bones in the rest pose to their respective closest points on the skeleton. This has the same effect as treating the bones as rigid components inside the elastic solid that drive the motion, while the visible surface of the character remains fully governed by the physical simulation. We used ten percent of the smallest diameter of the character's limbs as value for δ.

As a simple test, we ran the adaptive subspace simulation algorithm on 300 animated frames of an *Octopus* character rig, with a pose-space database created from full-space training data generated using the same input frames. The number of vertices of the tetrahedral simulation mesh was chosen such that the resulting full-space simulation time roughly matches the one for the cloth as in Table 6.1. Using this setup, our adaptive subspace solver achieved a speedup of 11x, and some of the resulting simulated frames can be observed in Figure 6.11. We did not enable any contact handling for this test. Interestingly, the subspace simulation is visually almost indistinguishable from the full-space simulation, and unlike in our clothing examples we noticed no loss of dynamic motion. The tradeoff seems to be that the speedup factor for our subspace solid simulation is significantly lower than the one we observed in the cloth case. However, the reasons for this phenomenon were not thoroughly investigated.

6.4 Summary and Outlook

We presented a new approach to clothing simulation using low-dimensional linear subspaces with temporally adaptive bases. Our method exploits full-space simulation training data in order to construct a pool of low-dimensional bases distributed across pose space. For this purpose, we interpret the simulation data as offsets from a kinematic deformation reference that captures the global shape of clothing due to body pose. During subspace simulation, we select low-dimensional sets of basis vectors according to the current pose of the character and the state of its clothing. Thanks to this adaptive basis selection scheme, our method is able to reproduce diverse and detailed folding patterns with only a few basis vectors. Our experiments demonstrate the feasibility of subspace clothing simulation and indicate its potential in terms of quality and computational efficiency.

While the subspace clothing simulation method presented here does not perform a rig-space optimization as the methods of the previous chapters, it demonstrates that rigs can be a powerful building block to develop novel simulation tools. Even though previous subspace simulation methods were

unable to handle the complicated case of cloth simulation, taking the underlying character motion provided by a rig into account allowed us to tackle subspace simulation of tight-fitting clothing. In the next chapter, we will conclude the work presented in this thesis and discuss future directions and limitations of the different discussed methods.

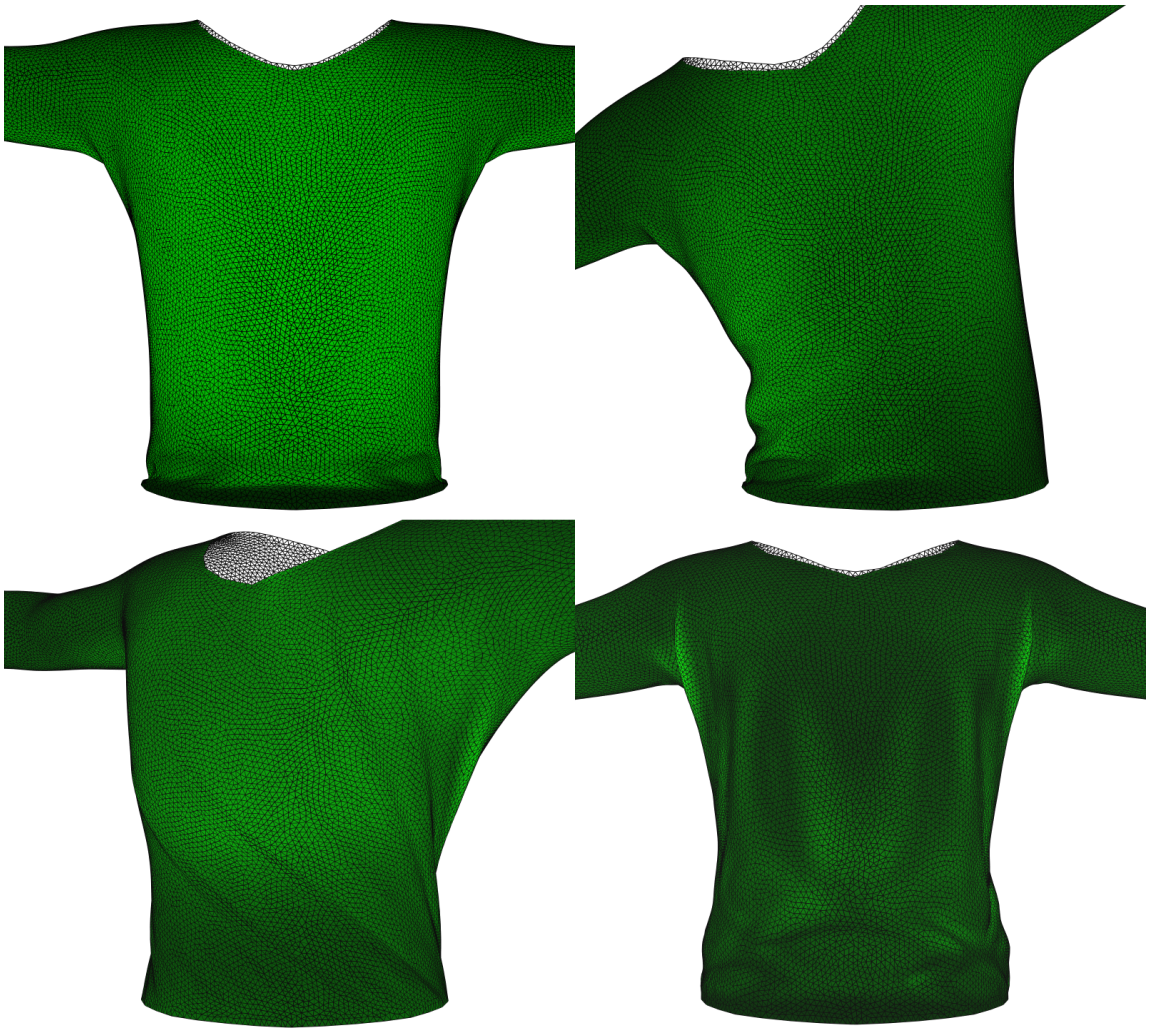

Figure 6.7: *Visualization of four basis vectors extracted from four different sites in our PSDB. The color intensity indicates the relative amount by which regions move when exciting the respective basis vector (dark: no motion, bright green: strong motion).*

Figure 6.8: Pants *example: Deformations for one frame of the* Legs Bend *sequence using full-space simulation (left) and adaptive subspace simulation with our method (right).*

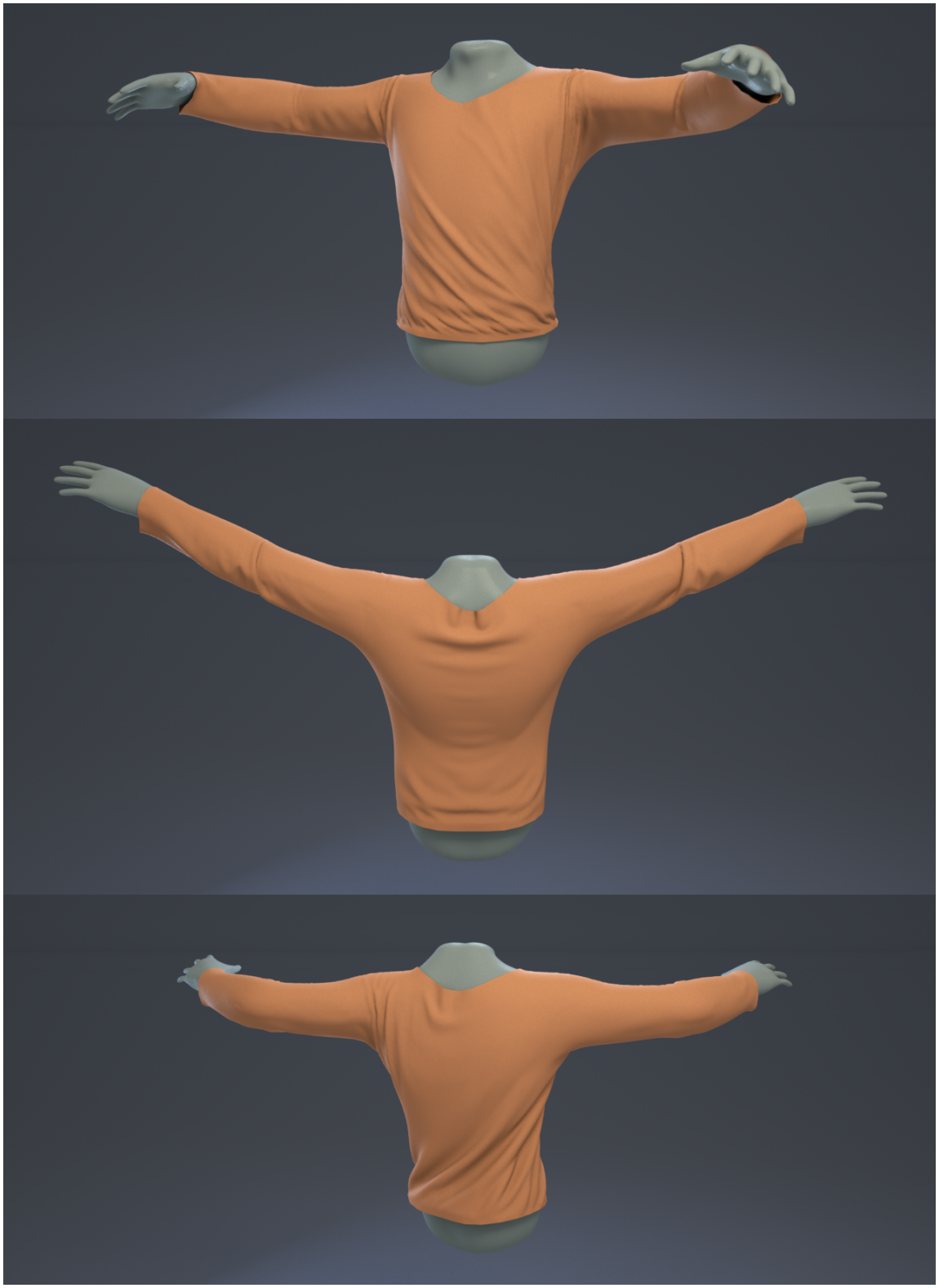

Figure 6.9: *Top: The* Yo *sequence exhibits simultaneous torsional folds and wrinkles around the armpits, which was not in the training data. Middle / Bottom: Two frames of the* Gym *sequence showing the combined effects of leaning back and stretching out the arms / twisting and bending the arms.*

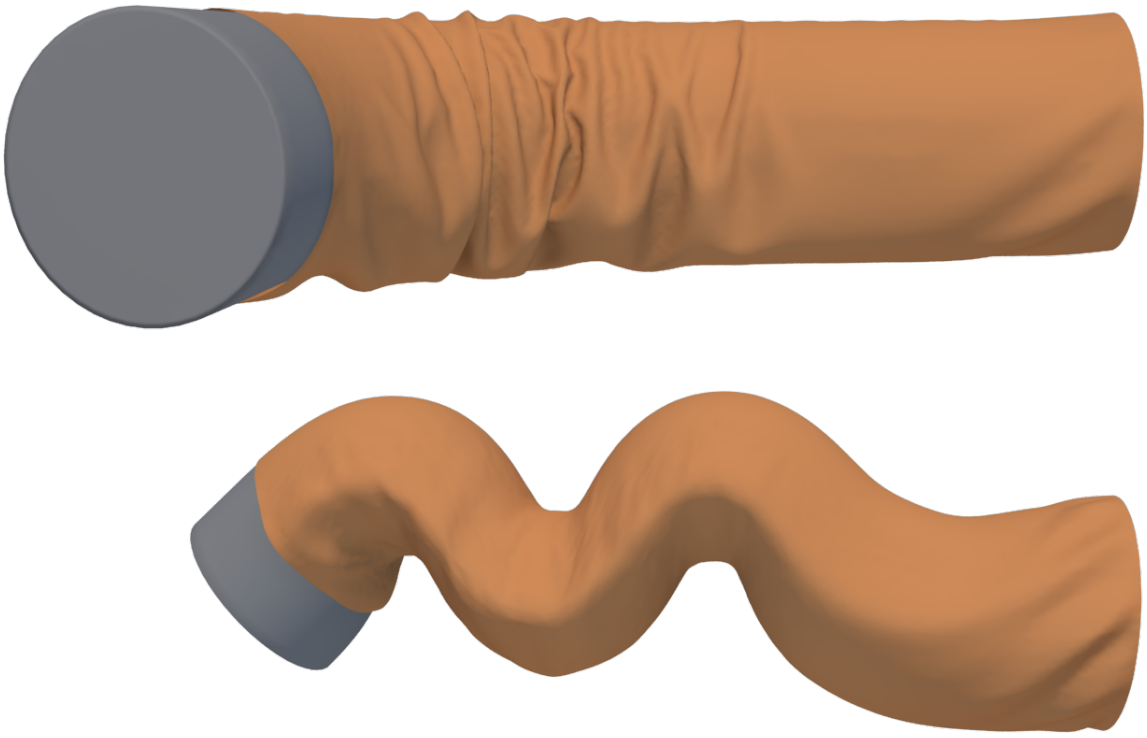

Figure 6.10: *When simulating this cloth with a PSDB featuring only motions in the horizontal plane, our method reproduces plausible wrinkling when bending the left end of the tube to the front (top). When moving to a pose very far from the training data, our solver produces deformations very close to the skinned basis (bottom).*

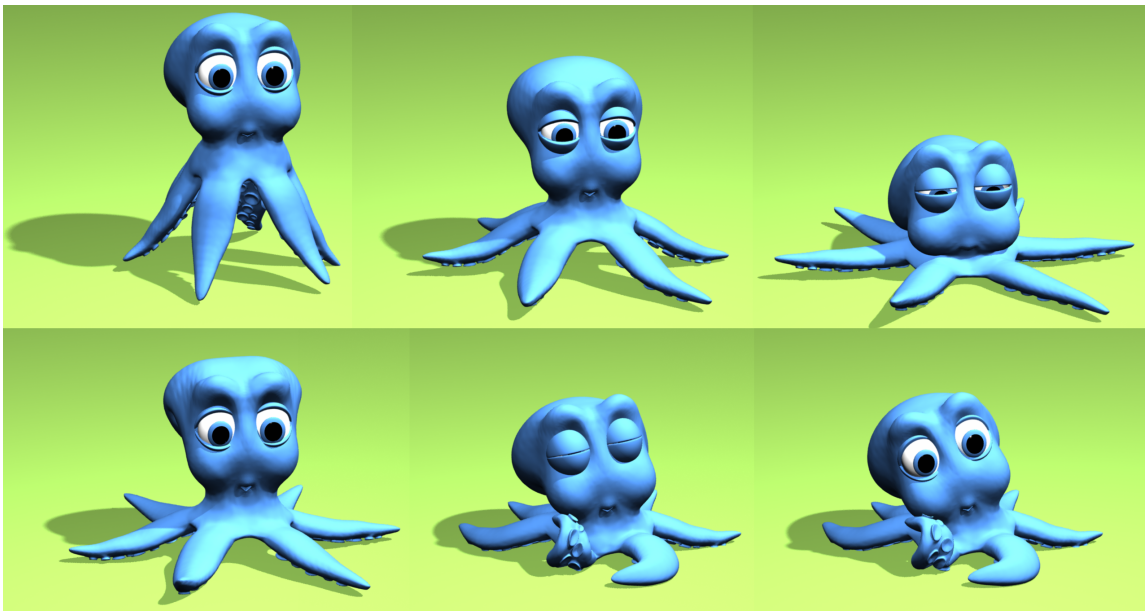

Figure 6.11: *From left to right, top to bottom: Six frames of adaptive subspace simulation of the* Octopus *character's body exhibiting jiggling due to the inertia caused by the skeletal input animation.*

CHAPTER 7

Conclusion

This chapter concludes the thesis by discussing the presented methods of this work and their contributions, as well as limitations and potential avenues for future work.

7.1 Discussion

In Chapter 1 of this work, we gave an introduction to the field of computational methods for virtual character animation, and stated the motivation for the different applications presented in the thesis. Relevant previous work related to the different areas touched on by these applications was discussed in Chapter 2. We then continued to lay the mathematical foundations of the simulation framework of our choice in Chapter 3, and described the physical model, the energy minimization technique, as well as the subspace solver we use. The chapter also touched on some analytic forms of widely used rigs and on contact handling strategies for the physics-based simulations. We used this simulation framework and its solver components for all of the main contributions of the thesis, which were presented in the context of the individual applications in the subsequent chapters.

For the first application of *Efficient Rig-Space Physics Simulation* in Chapter 4, we revisited the problem of simulating the surface of a virtual character in the deformation subspace defined by its artist-designed rig, which was first explored by Hahn and colleagues [2012] in their original *Rig-Space Physics* system. While this original system first demonstrated the concept of solving

for a physically plausible deformation of the character mesh by optimizing for the degrees of freedoms in the rig parameters rather than the surface vertex positions, its mathematical formulation rendered its performance prohibitively slow for practical use. Furthermore, the original system was only able to handle dozens of rig parameters at a time, while practical rigs used for the industrial production of animated movies or video games often expose hundreds of rig parameters, making the method unable to deal with some real-world examples. We addressed this problem by proposing a novel formulation of the problem that is computationally much more efficient, as well as several solver optimizations to obtain the maximal simulation performance.

We proceeded to reuse several of these optimized subspace techniques for the *Sketch Abstractions for Character Posing* application presented in Chapter 5. Rather than optimizing for a physically plausible pose of a rigged character, we proposed an optimization framework that solves for a character pose best matching a user-drawn sketch. Since matching a 2D sketch with the 3D pose of a virtual character is a highly underconstrained problem, we proposed the concept of a 2D sketch abstraction embedded into the character rig and driven by its controls to define a fully two-dimensional matching energy. A series of regularization terms helps resolve the remaining depth ambiguities. In addition to finding the pose of an existing character rig best matching a drawn sketch, we also demonstrated the ability of our method to be used for character individualization. By also using the user drawing as a descriptor for a model part from a database of pre-rigged character components, our system can enable novice users to assemble complex virtual characters with varying topologies from a sequence of drawn 2D strokes.

In Chapter 6, we used the rigging concept to design and implement a *Subspace Clothing Simulation* solver for tight-fitting garments worn by virtual characters. Even though previous subspace physics techniques were unable to tackle the hard problem of solving for cloth deformations in a reduced space, we were able to formulate an adaptive subspace simulation framework applicable to our clothing case. By transferring the skinning of the underlying character rig to the cloth mesh and using it as a kinematic reference, we were able to remove most of the nonlinear deformations exhibited by the cloth and untransform it back to a space where a linear subspace basis is powerful enough to express the remaining motions. Rather than using a single subspace for the whole simulation, we further proposed the use of a pose-space database to let the solver adapt even better to the change in pose of the character. We demonstrated the ability of our method to achieve significant speedups over unreduced full-space simulations, as well as its ability to generalize to novel motions not contained in its training data.

7.2 Limitations and Future Work

The focus of this work was to apply the idea of treating a character rig as a deformation space rather than just a practical tool for artists to various topics in the field of virtual character animation. Concretely, we looked at the three use cases of placing keyframes to animate a character rig, matching user-drawn sketches to pose a character rig, and using a character rig to drive the physical deformation of simulated cloth. We believe that the concept is powerful enough to extend to other stages of the animation pipeline and enable novel applications, in particular:

- In typical animation pipelines, modeling and rigging are two separate stages that are performed by different specialized artists working in different departments of a studio producing character animation. Recent work has explored methods that enable the coupled design of a character model and its rig by automatically fitting a skeleton to a sculpted body [Baran and Popović, 2007; Borosán et al., 2012]. It would be interesting to go beyond skeletal rigs and develop methods that can fit arbitrary black-box rigging hierarchies to a character as it is modeled.

- Once a character rig is created, it becomes difficult to change the behavior of its rig parameters without affecting existing poses or animations that have already been created. An automatic method could be used to optimize in the new rig space for a configuration that best matches a sequence of given deformations. A related application would be the retargeting of animations for one character to another character with potentially very different rigging, or even the matching of motion capture data with the controls of a complex production rig. While some retargeting methods such as the one proposed by Hecker and colleagues [2008] are available, they typically only apply to skeletal rigs rather than hierarchies of arbitrary geometric deformers, like the ones used by artists in practice.

- Since real-world character rigs used for the production of animated movies tend to expose hundreds of rig parameters driving a large variety and whole hierarchies of different geometric deformers, their evaluation tends to become very slow, making them barely interactive enough for animators to work with them. It would thus be very useful to have access to rig optimization methods that tackle this performance problem. Once possible approach would be to approximate the rig's deformation space using interpolation techniques, and then use the approximated rig in place of the real one if the interpola-

tion error is small enough. While Meyer and colleagues [2007] have proposed an interpolation approach to speed up the evaluation of facial rigs, it would be interesting to explore the optimization of rigs controlling the full body pose of a character.

- As we have seen in Chapter 6, the motion of simulated objects is often driven by an articulated character rig and their deformations exhibit high correlation with the character's pose. Rather than performing a subspace simulation to increase the performance of generating physical secondary motion for an object, an interesting alternative approach would be to directly "bake" the offsets that a physical simulation would produce into the rig. Kry and colleagues [2002] have proposed such a system for static poses using pose-space deformation rigging to handle the interpolation of the deformations, but it would be interesting to extend this idea to dynamic simulations and arbitrary rigging mechanisms.

While these ideas about using rig deformation spaces for novel applications were more general in nature, we will describe further limitations and potential future work directions specific to the three applications described in Chapters 4, 5 and 6 in the following subsections.

7.2.1 Efficient Rig-Space Physics Simulation

Editing the material stiffness for individual rig parameters is not currently supported by our method. As a related challenge, it can be cumbersome to find material parameters that yield soft behavior around the rest state but do not lead to excessive deformations for fast motion. A promising direction for future work involves investigating the intuitive design and art direction of such materials. In this endeavor, our method could provide quick feedback on the outcome.

For the examples shown in Section 4.3 of this work, we used a constant time step size 0.01s, which is a fraction of the upper bound as dictated by the number of frames per second. But while parts of an animation might actually admit this maximum step size, sequences with rapid motion and large deformations will typically require smaller steps in order to maintain stability. An adaptive time stepping scheme could exploit this fact, thereby increasing robustness and efficiency.

Our system currently uses a single rest state mesh and extreme character poses can potentially lead to very distorted or even inverted elements. An

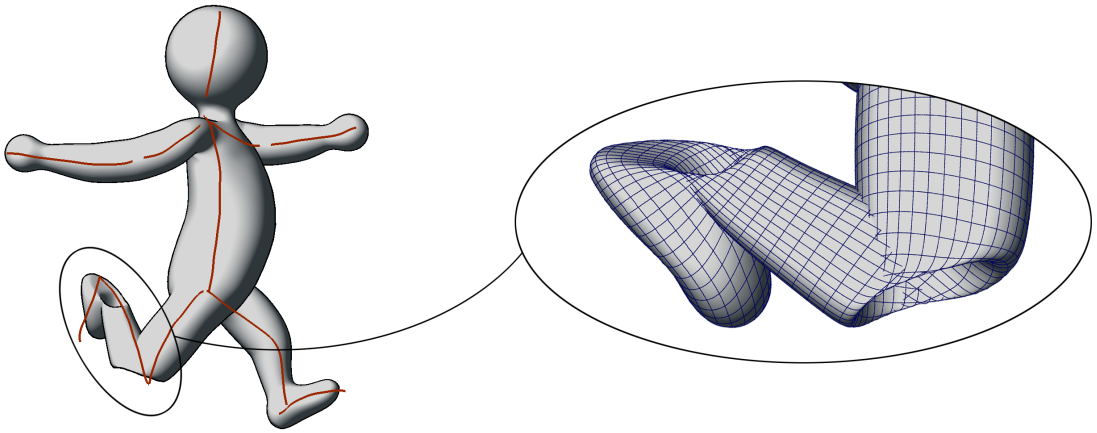

Figure 7.1: *Enabling limb scaling parameters for the* Cartoon Man *character can lead to negative scaling values, resulting in artifacts.*

interesting avenue for future work would be to investigate remeshing approaches or even meshless discretizations. Our method seems to invite such adaptive approaches since it uses only the free rig parameters as real degrees of freedom, making adaptations to the underlying mesh a lightweight process.

7.2.2 Sketch Abstractions for Character Posing

Our work provides a novel sketch-based posing framework that allows posing using artist-created 2D iconographic representations of animated characters. Although we demonstrate several applications enabled by our method, existing limitations in our work direct us to areas of future research. One limitation comes from the fact that the quality of the results ultimately depends greatly on the 2D matching quality. The uniform arc-length sampling of the sketch abstraction and the target user sketch may not be optimal in cases where a partial match could lead to better results or if the user desired non-uniform stretching along the stroke direction. Rusinkiewicz and Levoy [2001] introduced the concept of *normal space sampling*, which could be applied to potentially improve our method. Guay and colleagues [2013] report better perceived matching results for their line of action posing system when considering *tangent differences* between source and target strokes. For best results, sketches must be made in a prescribed order. This limitation could be alleviated by using the *smart scribbles* method of Noris and colleagues [2012].

Since our method completely operates in the deformation space spanned by the extended rig mapping, we cannot express poses beyond the limits

of what the rigging controls allow. While this intended behavior gives the artist full control over the character's range of poses, it can lead to situations where our method is unable to match sketches that described a pose outside the space of rig deformations. Another limitation arises from the fact that our optimization treats all rig parameters as dimensions of an unbounded continuous vector space, and thus does not adhere to any parameter bounds. For the *Cartoon Man* character, we had to disable the scaling parameters on the limb bones to prevent our solver from finding solutions with negative scaling parameters that invert limbs in order to better match the sketch. One example of such a problematic pose is shown in Figure 7.1. For a future version of our system, we would like to investigate numeric methods to add bounds to selected parameters.

One of the core contributions of our work is the generic and flexible energy formulation that is independent of the particular choice of rigging formulation. We believe that this numeric formulation could also impact other sketching paradigms. We would like to extend our system to allow silhouette sketching and line of action drawing [Zimmermann et al., 2007; Guay et al., 2013] by automatically generating sketch abstractions for these tasks. Since users would not have to know the specifics of the complex rigging mechanisms involved in modeling a 3D character, they could simply choose the abstraction that best matches their intention, further bridging the gap between the worlds of 2D and 3D animation.

Both our draw-over and redraw posing methods fit well into the commonly employed keyframe animation workflow for the creation of 3D motions for feature films and video games. While individual poses can easily be sketched using our system, typical animation systems only allow indirect control over the interpolation between these keyframe poses by, for example, adjusting the tangents of the temporally varying rig control curves. It would therefore be interesting to consider *motion lines* and other temporal cues well-known from sketchy 2D character animation and directly translate them into 3D transitions between poses, as explored in the recent work by Guay and colleagues [2015].

7.2.3 Subspace Clothing Simulation

While we are able to faithfully reproduce wrinkles and folds for our test examples, one limitation of our method is the limited ability to reproduce dynamics. Nevertheless, we feel that performing a dynamic simulation is useful since it makes the deformations depend on past history. This enables the reproduction of hysteresis effects where different deformations are achieved

for the same pose depending on the motion path that lead to it. The observation that real wrinkles tend to move semi-rigidly in the normal direction of their fold could motivate the use of more a sophisticated basis vector creation than PCA. On a related note, shaping the subspace vectors manually to reflect the desired deformation could enable stylization and artist control.

Previous work by An and colleagues [2008] and Kim and colleagues [2009] showed that the performance of subspace simulations can be significantly increased using *cubature*. The underlying idea is to evaluate the full-space energy and its derivatives approximately using a small number of key elements. However, compared to typical deformations for volumetric solids, the folds observed in cloth are large and localized, making efficient cubature challenging and thus an interesting avenue for future research.

We currently handle collisions uniformly on a per-vertex basis for both our full-space and subspace simulation methods. Our initial tests suggest that culling-based subspace collision techniques could be adapted to our setting, promising an additional potential speedup for our subspace method.

Another limitation is that it is currently not possible to adapt the material properties of the clothing in the subspace. Consequentially, the look of the cloth is determined by the full-space simulation that was used to generate the training data. While changing stiffness and bending coefficients would be easily possible, the physical interpretation of this is unclear and the results would be difficult to predict. That said, enabling fast subspace resimulation to explore the effect of different material properties would be an interesting application.

APPENDIX A

Notation

A.1 Symbols, Variables and Operators

Scalars will typically be denoted using lower-case variables. For instance, x could be a variable storing the position of a vertex on the x axis.

Vectors will typically be denoted using boldface lower-case variables, for instance $\mathbf{a}$, and will be laid out as a column unless otherwise mentioned. To define a vector and its elements explicitly, we will use the square bracket notation using [and]. For example, a vertex $\mathbf{v} \in \mathbb{R}^3$ containing the dimensions x, y, and z could be defined as:

$$\mathbf{v} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{A.1}$$

To access individual elements of a vector, we will use the *bracket subscript operator* using (and) as well as a subscript containing the index of the desired vector dimension. In the case of the vector $\mathbf{a}$, the notation $(\mathbf{a})_i$ refers to the i-th element of the vector.

One notable exception to writing vectors as boldface lower-case variables occurs when we distinguish between deformed and undeformed physical states. In those cases, we will often denote the undeformed state as boldface upper-case $\mathbf{X}$, which corresponds to a deformed state $\mathbf{x}$.

Matrices will typically be denoted using boldface upper-case variables, for instance $\mathbf{A}$. To define a matrix and its elements explicitly, we will again use the square bracket notation using [and]. For example, a matrix $\mathbf{A} \in \mathbb{R}^{2\times 3}$ could be defined as:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \tag{A.2}$$

We will again employ the bracket subscript operator using (and) to access elements of a matrix. A subscript with one index refers to a specific column of a matrix, whereas a subscript with two indices refers to a single matrix entry. For example, $(\mathbf{A})_i \in \mathbb{R}^n$ refers to the i-th column of the matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, but $(\mathbf{A})_{ij} \in \mathbb{R}$ refers to the element in the i-th row and the j-th column of $\mathbf{A}$.

Transposes are indicated by a superscript ^{T}. For instance, if $\mathbf{A} \in \mathbb{R}^{m\times n}$, then $\mathbf{A}^T \in \mathbb{R}^{n\times m}$ is its transpose matrix. If $\mathbf{a} \in \mathbb{R}^m$ is a column vector, then $\mathbf{a}^T$ denotes its row vector representation.

Matrix and Vector Multiplications are expressed as simple concatenations without any explicit symbol. For example, $\mathbf{AB}$ denotes the multiplication between matrices $\mathbf{A}$ and $\mathbf{B}$, while $\mathbf{a}^T\mathbf{b}$ denotes the scalar product, also called dot product, between vectors $\mathbf{a}$ and $\mathbf{b}$ of matching size.

Scalar Multiplication of a vector or matrix is written using the $\cdot$ symbol to make it visually more distinguishable from matrix multiplication. For example, $k \cdot \mathbf{A}$ denotes a matrix of the same size as $\mathbf{A}$ with each element multiplied by the scalar $k \in \mathbb{R}$. If $\mathbf{A} \in \mathbb{R}^{m\times n}$ is a matrix and $\mathbf{b} \in \mathbb{R}^n$ a compatible vector, then $(\mathbf{A})_{ij} \cdot \mathbf{b} \in \mathbb{R}^n$ denotes a scalar multiplication, while $\mathbf{Ab} \in \mathbb{R}^m$ would be a matrix-vector multiplication. We omit the $\cdot$ symbol for scalar multiplications with fractions for aesthetic reasons.

Cross Products between two three-dimensional vectors are denoted by the $\times$ symbol. Given two vectors $\mathbf{x} = [x_1, x_2, x_3]^T$ and $\mathbf{y} = [y_1, y_2, y_3]^T$, their cross product is defined as:

$$\mathbf{x} \times \mathbf{y} = \begin{bmatrix} x_2 \cdot y_3 - x_3 \cdot y_2 \\ x_1 \cdot y_3 - x_3 \cdot y_1 \\ x_1 \cdot y_2 - x_2 \cdot y_1 \end{bmatrix} \in \mathbb{R}^3 \tag{A.3}$$

Identity Matrices with ones on the diagonal and zeros otherwise are always called $\mathbf{I}_n$, and their subscript n denotes their dimension. For example, $\mathbf{I}_3 \in \mathbb{R}^{3\times 3}$ represents a 3D identity matrix.

Inverses of regular square matrices are indicated by a superscript $^{-1}$. For example, if $\mathbf{A} \in \mathbb{R}^{m\times m}$ is a regular matrix, then $\mathbf{A}^{-1} \in \mathbb{R}^{m\times m}$ is its inverse such that $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}_m$.

Norms of vectors and matrices are written using two enclosing $\|$ symbols, where an additional subscript denotes the type of norm. For example, the $L2$ norm of a vector $\mathbf{a}$ is expressed as $\|\mathbf{a}\|_2$, while the Frobenius norm of a matrix $\mathbf{A}$ is expressed as $\|\mathbf{A}\|_F$.

A.2 Matrix and Vector Derivatives

Throughout this thesis, we are often making use of derivatives of or with respect to vectors, often resulting in matrices or in some cases in even higher-order tensors. When writing derivatives in the form of matrix multiplications, there are two competing notations that can be used, depending on how the derivative of one vector $\mathbf{x} \in \mathbb{R}^m$ with respect to another vector $\mathbf{y} \in \mathbb{R}^n$ is laid out:

- In the *numerator* layout, the derivative is expressed as a matrix $\frac{\partial \mathbf{x}}{\partial \mathbf{y}} \in \mathbb{R}^{m\times n}$ and is equal to the Jacobian matrix of $\mathbf{x}$ with respect to $\mathbf{y}$. As a consequence, the gradient $\frac{\partial \mathbf{x}}{\partial s}$ with respect to a scalar s should be laid out as a row vector.
- In the *denumerator* layout, the derivative is expressed as a matrix $\frac{\partial \mathbf{x}}{\partial \mathbf{y}} \in \mathbb{R}^{n\times m}$ and is equal to the transposed Jacobian matrix of $\mathbf{x}$ with respect to $\mathbf{y}$. As a consequence, the gradient $\frac{\partial \mathbf{x}}{\partial s}$ with respect to a scalar s should be laid out as a column vector.

In this work, we will use the former numerator layout whenever possible, but will sometimes introduce variables that refer to the transpose of a derivative to simplify some of the formulas.

The most important identity for our derivations is the *chain rule*. As an example, let $\mathbf{x} : \mathbb{R}^n \to \mathbb{R}^m$, $\mathbf{y} : \mathbb{R}^k \to \mathbb{R}^n$ and $\mathbf{z} \in \mathbb{R}^k$. Then the derivative of

$\mathbf{x}(\mathbf{y}(\mathbf{z}))$ with respect to $\mathbf{z}$ is given by the chain rule as:

$$\underbrace{\frac{\partial}{\partial \mathbf{z}}\mathbf{x}(\mathbf{y}(\mathbf{z}))}_{\in \mathbb{R}^{m\times k}} = \underbrace{\frac{\partial}{\partial \mathbf{y}}\mathbf{x}(\mathbf{y}(\mathbf{z}))}_{\in \mathbb{R}^{m\times n}} \underbrace{\frac{\partial}{\partial \mathbf{z}}\mathbf{y}(\mathbf{z})}_{\in \mathbb{R}^{n\times k}} \tag{A.4}$$

Another identity we sometimes use is the *product rule*. As an example, let $\mathbf{x} : \mathbb{R}^n \to \mathbb{R}^m$, $\mathbf{y} : \mathbb{R}^n \to \mathbb{R}^m$ and $\mathbf{z} \in \mathbb{R}^n$. Then the gradient of $\mathbf{x}(\mathbf{z})^T\mathbf{y}(\mathbf{z})$ with respect to $\mathbf{z}$ is given as

$$\underbrace{\frac{\partial}{\partial \mathbf{z}}\mathbf{x}(\mathbf{z})^T\mathbf{y}(\mathbf{z})}_{\in \mathbb{R}^{n}} = \mathbf{x}(\mathbf{z})^T \underbrace{\frac{\partial}{\partial \mathbf{z}}\mathbf{y}(\mathbf{z})}_{\in \mathbb{R}^{m\times n}} + \mathbf{y}(\mathbf{z})^T \underbrace{\frac{\partial}{\partial \mathbf{z}}\mathbf{x}(\mathbf{z})}_{\in \mathbb{R}^{m\times n}}, \tag{A.5}$$

where the left-hand side is understood to be laid out as a column vector. In most cases in this thesis, one of the Jacobians in Equation (A.5) is zero and we will simply state the nonzero term without writing out the product rule explicitly.

In order to perform Newton-Raphson iterations as described in Section 3.2.2, we require the second derivative of a scalar function with respect to its parameter vector. However, we cannot simply apply the gradient operator twice since the first applications results in a row vector which is incompatible with the second application. Since we assume the Hessian matrix to be symmetric, we circumvent this problem by simply defining it as the gradient of the transposed gradient in a slight abuse of notation. For example, the second derivative of the scalar function $S : \mathbb{R}^n \to \mathbb{R}$ with respect to the vector $\mathbf{x} \in \mathbb{R}^n$ is given as:

$$\frac{\partial^2}{\partial \mathbf{x}^2}S(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}}\left(\frac{\partial}{\partial \mathbf{x}}S(\mathbf{x})\right)^T \in \mathbb{R}^{n\times n} \tag{A.6}$$

References

[An et al., 2008] Steven S. An, Theodore Kim, and Doug L. James. Optimizing cubature for efficient integration of subspace deformations. *ACM Trans. Graph.*, 27(5):165:1–165:10, December 2008.

[ApS, 2015] MOSEK ApS. The mosek c optimizer api manual version 7.1 (revision 32) [Online; accessed Jul 24th, 2015], `http://docs.mosek.com/7.1/capi/index.html`, 2015.

[Autodesk, 2015] Autodesk. Autodesk maya 2014 c++ api reference [Online; accessed Jun 9th, 2015], `http://docs.autodesk.com/MAYAUL/2014/ENU/Maya-API-Documentation/cpp_ref/class_m_fn_ik_joint.html`, 2015.

[Baraff and Witkin, 1998] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proceedings of SIGGRAPH 98*, Annual Conference Series, pages 43–54, 1998.

[Baran and Popović, 2007] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3d characters. *ACM Trans. Graph.*, 26(3), July 2007.

[Barbič and James, 2005] Jernej Barbič and Doug L. James. Real-time subspace integration for st. venant-kirchhoff deformable models. *ACM Trans. Graph.*, 24(3):982–990, July 2005.

[Barbič and James, 2008] Jernej Barbič and Doug L. James. Six-dof haptic rendering of contact between geometrically complex reduced deformable models. *IEEE Trans. Haptics*, 1(1):39–52, January 2008.

[Barbič and James, 2010] Jernej Barbič and Doug L. James. Subspace self-collision culling. *ACM Trans. Graph.*, 29(4):81:1–81:9, July 2010.

[Barbič et al., 2009] Jernej Barbič, Marco da Silva, and Jovan Popović. Deformable object animation using reduced optimal control. *ACM Trans. Graph.*, 28(3):53:1–53:9, July 2009.

[Baskin, 2014] Jason Baskin. Mike and Tina character rig 2.5.0 [Online; accessed Jan 20th, 2015], `http://www.creativecrash.com/maya/downloads/character-rigs/c/mike-and-tina-character-rig`, 2014.

[Bengio and Goldenthal, 2013] Julien Cohen Bengio and Rony Goldenthal. Simplicial interpolation for animating the hulk. In *ACM SIGGRAPH 2013 Talks*, SIGGRAPH '13, pages 7:1–7:1, 2013.

[Bonet and Wood, 1997] J. Bonet and R. D. Wood. *Nonlinear Continuum Mechanics for Finite Element Analysis*. Cambridge Univ. Press, 1997.

[Borosán et al., 2012] Péter Borosán, Ming Jin, Doug DeCarlo, Yotam Gingold, and Andrew Nealen. Rigmesh: Automatic rigging for part-based shape modeling and deformation. *ACM Trans. Graph.*, 31(6):198:1–198:9, November 2012.

[Bridson et al., 2002] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph.*, 21(3):594–603, July 2002.

[Brochu and Bridson, 2009] Tyson Brochu and Robert Bridson. Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing*, 31(4):2472–2493, 2009.

[Chang and Jenkins, 2006] Edwin Chang and Odest Chadwicke Jenkins. Sketching articulation and pose for facial animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 271–280, 2006.

[Chen et al., 2008] Yanqing Chen, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Softw.*, 35(3):22:1–22:14, October 2008.

[Choi et al., 2012] Myung Geol Choi, Kyungyong Yang, Takeo Igarashi, Jun Mitani, and Jehee Lee. Retrieval and visualization of human motion data via stick figures. *Comput. Graph. Forum*, 31(7-1):2057–2065, 2012.

[Coros et al., 2013] Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W. Sumner, Wojciech Matusik, and Bernd Bickel. Computational design of mechanical characters. *ACM Trans. on Graphics*, 32(4):83:1–83:12, July 2013.

[Davis et al., 2003] James Davis, Maneesh Agrawala, Erika Chuang, Zoran Popović, and David Salesin. A sketching interface for articulated figure animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 320–328, 2003.

[de Aguiar et al., 2010] Edilson de Aguiar, Leonid Sigal, Adrien Treuille, and Jessica K. Hodgins. Stable spaces for real-time clothing. *ACM Trans. Graph.*, 29(4):106:1–106:9, July 2010.

[Eitz et al., 2012] Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Trans. Graph.*, 31(4):44:1–44:10, July 2012.

[Faloutsos et al., 1997] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Dynamic free-form deformations for animation synthesis. *IEEE Trans. on Visualization and Computer Graphics*, 3(3), 1997.

[Feng et al., 2010] Wei-Wen Feng, Yizhou Yu, and Byung-Uck Kim. A deformation transformer for real-time cloth animation. *ACM Trans. Graph.*, 29(4):108:1–108:9, July 2010.

[Funkhouser et al., 2004] Thomas Funkhouser, Michael Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David Dobkin. Modeling by example. *ACM Trans. Graph.*, 23(3):652–663, August 2004.

[Gilles et al., 2011] Benjamin Gilles, Guillaume Bousquet, Francois Faure, and Dinesh K. Pai. Frame-based elastic models. *ACM Trans. Graph.*, 30(2):15:1–15:12, April 2011.

[Gingold et al., 2009] Yotam Gingold, Takeo Igarashi, and Denis Zorin. Structured annotations for 2d-to-3d modeling. *ACM Trans. Graph.*, 28(5):148:1–148:9, December 2009.

[Grinspun et al., 2003] Eitan Grinspun, Anil N. Hirani, Mathieu Desbrun, and Peter Schröder. Discrete shells. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 62–67, 2003.

[Guan et al., 2012] Peng Guan, Loretta Reiss, David A. Hirshberg, Alexander Weiss, and Michael J. Black. Drape: Dressing any person. *ACM Trans. Graph.*, 31(4):35:1–35:10, July 2012.

[Guay et al., 2013] Martin Guay, Marie-Paule Cani, and Rémi Ronfard. The line of action: An intuitive interface for expressive character posing. *ACM Trans. Graph.*, 32(6):205:1–205:8, November 2013.

[Guay et al., 2015] Martin Guay, Rémi Ronfard, Michael Gleicher, and Marie-Paule Cani. Space-time sketching of character animation. *ACM Trans. Graph.*, 34(4):1, May 2015.

[Gunnarsson and Maddock, 2010] Orn Gunnarsson and Steve C. Maddock. Sketch-based posing of 3d faces for facial animation. In *Theory and Practice of Computer Graphics*, pages 223–230, 2010.

[Hahn et al., 2012] Fabian Hahn, Sebastian Martin, Bernhard Thomaszewski, Robert Sumner, Stelian Coros, and Markus Gross. Rig-space physics. *ACM Trans. Graph.*, 31(4):72:1–72:8, July 2012.

[Harmon and Zorin, 2013] David Harmon and Denis Zorin. Subspace integration with local deformations. *ACM Trans. Graph.*, 32(4):107:1–107:10, July 2013.

[Hasler et al., 2010] Nils Hasler, Thorsten Thormählen, Bodo Rosenhahn, and Hans-Peter Seidel. Learning skeletons for shape and pose. In *Proc. of Symp. on Interactive 3D Graphics '10*, 2010.

[Hecker et al., 2008] Chris Hecker, Bernd Raabe, Ryan W. Enslow, John DeWeese, Jordan Maynard, and Kees van Prooijen. Real-time motion retargeting to highly varied user-created morphologies. *ACM Trans. Graph.*, 27(3):27:1–27:11, August 2008.

[Igarashi et al., 1999] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3d freeform design. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 409–416, 1999.

[Irving et al., 2004] G. Irving, J. Teran, and R. Fedkiw. Invertible finite elements for robust simulation of large deformation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 131–140, 2004.

[Jacobson and Sorkine, 2011] Alec Jacobson and Olga Sorkine. Stretchable and twistable bones for skeletal shape deformation. *ACM Trans. Graph.*, 30(6):165:1–165:8, December 2011.

[Jacobson et al., 2012] Alec Jacobson, Ilya Baran, Ladislav Kavan, Jovan Popović, and Olga Sorkine. Fast automatic skinning transformations. *ACM Trans. Graph.*, 31(4):77:1–77:10, July 2012.

[James and Fatahalian, 2003] Doug L. James and Kayvon Fatahalian. Precomputing interactive dynamic deformable scenes. *ACM Trans. Graph.*, 22(3):879–887, July 2003.

[James and Pai, 1999] Doug L. James and Dinesh K. Pai. Artdefo: Accurate real time deformable objects. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 65–72, 1999.

[James and Twigg, 2005] Doug L. James and Christopher D. Twigg. Skinning mesh animations. *ACM Trans. Graph.*, 24(3):399–407, July 2005.

[Joshi et al., 2007] Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. Harmonic coordinates for character articulation. *ACM Trans. Graph.*, 26(3), July 2007.

[Ju et al., 2005] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph.*, 24(3):561–566, July 2005.

[Karpenko and Hughes, 2006] Olga A. Karpenko and John F. Hughes. Smoothsketch: 3d free-form shapes from complex sketches. *ACM Trans. Graph.*, 25(3):589–598, July 2006.

[Kavan and Sorkine, 2012] Ladislav Kavan and Olga Sorkine. Elasticity-inspired deformers for character articulation. *ACM Trans. Graph.*, 31(6):196:1–196:8, November 2012.

[Kavan et al., 2008] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Geometric skinning with approximate dual quaternion blending. *ACM Trans. Graph.*, 27(4):105:1–105:23, November 2008.

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

[Kavan et al., 2011] Ladislav Kavan, Dan Gerszewski, Adam W. Bargteil, and Peter-Pike Sloan. Physics-inspired upsampling for cloth simulation in games. *ACM Trans. Graph.*, 30(4):93:1–93:10, July 2011.

[Kho and Garland, 2005] Youngihn Kho and Michael Garland. Sketching mesh deformations. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, I3D '05, pages 147–154, 2005.

[Kim and James, 2009] Theodore Kim and Doug L. James. Skipping steps in deformable simulation with online model reduction. *ACM Trans. Graph.*, 28(5):123:1–123:9, December 2009.

[Kim and Vendrovsky, 2008] Tae-Yong Kim and Eugene Vendrovsky. Drivenshape: A data-driven approach for shape deformation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 49–55, 2008.

[Kim et al., 2013] Doyub Kim, Woojong Koh, Rahul Narain, Kayvon Fatahalian, Adrien Treuille, and James F. O'Brien. Near-exhaustive precomputation of secondary cloth effects. *ACM Trans. Graph.*, 32(4):87:1–87:8, July 2013.

[Kraevoy et al., 2009] Vladislav Kraevoy, Alla Sheffer, and Michiel van de Panne. Modeling from contour drawings. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '09, pages 37–44, 2009.

[Kry et al., 2002] Paul G. Kry, Doug L. James, and Dinesh K. Pai. Eigenskin: Real time large deformation character skinning in hardware. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 153–159, 2002.

[Krysl et al., 2001] P. Krysl, S. Lall, and J. E. Marsden. Dimensional model reduction in non-linear finite element dynamics of solids and structures. *International Journal for Numerical Methods in Engineering*, 51:479–504, 2001.

[Kurihara and Miyata, 2004] Tsuneya Kurihara and Natsuki Miyata. Modeling deformable human hands from medical images. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 355–363, 2004.

[Lau et al., 2009] Manfred Lau, Jinxiang Chai, Ying-Qing Xu, and Heung-Yeung Shum. Face poser: Interactive modeling of 3d facial expressions using facial priors. *ACM Trans. Graph.*, 29(1):3:1–3:17, December 2009.

[Le and Deng, 2012] Binh Huy Le and Zhigang Deng. Smooth skinning decomposition with rigid bones. *ACM Trans. Graph.*, 31(6):199:1–199:10, November 2012.

[Lee and Funkhouser, 2008] Jeehyung Lee and Thomas Funkhouser. Sketch-based search and composition of 3d models. In *Proceedings of the Fifth Eurographics Conference on Sketch-Based Interfaces and Modeling*, SBM'08, pages 97–104, 2008.

[Lee, 2009] Gene S. Lee. Evaluation of the radial basis function space. In *ACM SIGGRAPH ASIA 2009 Sketches*, SIGGRAPH ASIA '09, pages 42:1–42:1, 2009.

[Lewis et al., 2000] J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 165–172, 2000.

[Lin et al., 2010] Juncong Lin, Takeo Igarashi, Jun Mitani, and Greg Saul. A sketching interface for sitting-pose design. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, SBIM '10, pages 111–118. Eurographics Association, 2010.

[Lipman et al., 2008] Yaron Lipman, David Levin, and Daniel Cohen-Or. Green coordinates. *ACM Trans. Graph.*, 27(3):78:1–78:10, August 2008.

[Liu et al., 2013] Tiantian Liu, Adam W. Bargteil, James F. O'Brien, and Ladislav Kavan. Fast simulation of mass-spring systems. *ACM Trans. Graph.*, 32(6):214:1–214:7, November 2013.

[Magnenat-Thalmann et al., 1989] N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proc. of Graphics Interface '88*, 1989.

[Martin et al., 2011] Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. Example-based elastic materials. *ACM Trans. Graph.*, 30(4):72:1–72:8, July 2011.

[McAdams et al., 2011] Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph.*, 30(4):37:1–37:12, July 2011.

[McLaughlin et al., 2011] Tim McLaughlin, Larry Cutler, and David Coleman. Character rigging, deformations, and simulations in film and game production. In *ACM SIGGRAPH 2011 Courses*, 2011.

[Meyer and Anderson, 2007] Mark Meyer and John Anderson. Key point subspace acceleration and soft caching. *ACM Trans. Graph.*, 26(3), July 2007.

[Miranda et al., 2012] José Carlos Miranda, Xenxo Alvarez, João Orvalho, Diego Gutierrez, A. Augusto Sousa, and Verónica Orvalho. Sketch express: A sketching interface for facial animation. *Computers & Graphics*, 36(6):585 – 595, 2012.

[Müller and Chentanez, 2010] Matthias Müller and Nuttapong Chentanez. Wrinkle meshes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 85–92, 2010.

[Nealen et al., 2005] Andrew Nealen, Olga Sorkine, Marc Alexa, and Daniel Cohen-Or. A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph.*, 24(3):1142–1147, July 2005.

[Nealen et al., 2006] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics. *Computer Graphics Forum*, 25(4):809–836, 2006.

[Nealen et al., 2007] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Fibermesh: Designing freeform surfaces with 3d curves. *ACM Trans. Graph.*, 26(3), July 2007.

[Neumann et al., 2013] Thomas Neumann, Kiran Varanasi, Stephan Wenger, Markus Wacker, Marcus Magnor, and Christian Theobalt. Sparse localized deformation components. *ACM Trans. Graph.*, 32(6):179:1–179:10, November 2013.

[Nocedal and Wright, 2006] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.

[Noris et al., 2012] G. Noris, D. Sýkora, A. Shamir, S. Coros, B. Whited, M. Simmons, A. Hornung, M. Gross, and R. Sumner. Smart scribbles for sketch segmentation. *Computer Graphics Forum*, 31(8):2516–2527, 2012.

[Olsen et al., 2009] Luke Olsen, Faramarz F. Samavati, Mario Costa Sousa, and Joaquim A. Jorge. Sketch-based modeling: A survey. *Computers & Graphics*, 33(1):85 – 103, 2009.

[Öztireli et al., 2013] A. Cengiz Öztireli, Ilya Baran, Tiberiu Popa, Boris Dalstein, Robert W. Sumner, and Markus Gross. Differential blending for expressive sketch-based posing. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 155–164, 2013.

[Press et al., 2007] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. 3 edition, 2007.

[Rohmer et al., 2010] Damien Rohmer, Tiberiu Popa, Marie-Paule Cani, Stefanie Hahmann, and Alla Sheffer. Animation wrinkling: Augmenting coarse cloth simulations with realistic-looking wrinkles. *ACM Trans. Graph.*, 29(6):157:1–157:8, December 2010.

[Rusinkiewicz and Levoy, 2001] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the ICP algorithm. In *3-D Digital Imaging and Modeling*, pages 145–152, 2001.

[Schmidt et al., 2007] Mark Schmidt, Glenn Fung, and Rómer Rosales. Fast optimization methods for l1 regularization: A comparative study and two new approaches. In *Machine Learning: ECML 2007*, volume 4701 of *Lecture Notes in Computer Science*, pages 286–297, 2007.

[Selle et al., 2009] Andrew Selle, Jonathan Su, Geoffrey Irving, and Ronald Fedkiw. Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction. *IEEE Transactions on Visualization and Computer Graphics*, 15(2):339–350, March 2009.

[Shin and Igarashi, 2007] Hyojong Shin and Takeo Igarashi. Magic canvas: Interactive design of a 3-D scene prototype from freehand sketches. In *Proceedings of Graphics Interface 2007*, GI '07, pages 63–70, 2007.

[Singh and Fiume, 1998] Karan Singh and Eugene Fiume. Wires: A geometric deformation technique. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 405–414, 1998.

[Sloan et al., 2001] Peter-Pike J. Sloan, Charles F. Rose, III, and Michael F. Cohen. Shape by example. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 135–143, 2001.

[Terzopoulos et al., 1987] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 205–214, 1987.

[Thomaszewski et al., 2008] Bernhard Thomaszewski, Simon Pabst, and Strasser Wolfgang. Asynchronous cloth simulation. In *Proceedings of Computer Graphics International 08*, 2008.

[Wang and Phillips, 2002] Xiaohuan Corina Wang and Cary Phillips. Multi-weight enveloping: Least-squares approximation techniques for skin animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 129–138, 2002.

[Wang et al., 2010] Huamin Wang, Florian Hecht, Ravi Ramamoorthi, and James F. O'Brien. Example-based wrinkle synthesis for clothing animation. *ACM Trans. Graph.*, 29(4):107:1–107:8, July 2010.

[Weber et al., 2007] Ofir Weber, Olga Sorkine, Yaron Lipman, and Craig Gotsman. Context-aware skeletal shape deformation. *Computer Graphics Forum*, 26(3):265–274, 2007.

[Wei and Chai, 2011] Xiaolin K. Wei and Jinxiang Chai. Intuitive interactive human-character posing with millions of example poses. *Computer Graphics and Applications, IEEE*, 31(4):78–88, July 2011.

[Wong et al., 2013] Sai-Keung Wong, Wen-Chieh Lin, Chun-Hung Hung, Yi-Jheng Huang, and Shing-Yeu Lii. Radial view based culling for continuous self-collision detection of skeletal models. *ACM Trans. Graph.*, 32(4):114:1–114:10, July 2013.

[Wyvill et al., 2005] B. Wyvill, K. Foster, P. Jepp, R. Schmidt, M. C. Sousa, and J. A. Jorge. Sketch based construction and rendering of implicit models. In *Proceedings of the Eurographics Conference on Computational Aesthetics in Graphics, Visualization and Imaging*, pages 67–74, 2005.

[Zheng and James, 2012] Changxi Zheng and Doug L. James. Energy-based self-collision culling for arbitrary mesh deformations. *ACM Trans. Graph.*, 31(4):98:1–98:12, July 2012.

[Zhou et al., 2005] Kun Zhou, Jin Huang, John Snyder, Xinguo Liu, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Large mesh deformation using the volumetric graph laplacian. *ACM Trans. Graph.*, 24(3):496–503, July 2005.

[Zimmermann et al., 2007] Johannes Zimmermann, Andrew Nealen, and Marc Alexa. SilSketch: Automated sketch-based editing of surface meshes. In

Proceedings of the Eurographics Workshop on Sketch-based Interfaces and Modeling, pages 23–30, 2007.

[Zurdo et al., 2013] Javier S. Zurdo, Juan P. Brito, and Miguel A. Otaduy. Animating wrinkles by example on non-skinned cloth. *IEEE Trans. Vis. Comput. Graph.*, 19(1):149–158, January 2013.