Animation, Simulation, and Control of Soft Characters using Layered Representations and Simplified Physics-based Methods

Nico Galoppo

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2008

Approved by:

Ming C. Lin, Advisor

Miguel A. Otaduy, Reader

Dinesh Manocha, Reader

Markus Gross, Committee Member

Anselmo Lastra, Committee Member

Abstract

Nico Galoppo: Animation, Simulation, and Control of Soft Characters using Layered Representations and Simplified Physics-based Methods. (Under the direction of Ming C. Lin.)

Realistic behavior of computer generated characters is key to bringing virtual environments, computer games, and other interactive applications to life. The plausibility of a virtual scene is strongly influenced by the way objects move around and *interact* with each other. Traditionally, actions are limited to motion capture driven or pre-scripted motion of the characters. Physics enhance the sense of realism: physical simulation is required to make objects act as expected in real life. To make gaming and virtual environments truly immersive, it is crucial to simulate the response of characters to collisions and to produce secondary effects such as skin wrinkling and muscle bulging. Unfortunately, existing techniques cannot generally achieve these effects in real time, do not address the coupled response of a character's skeleton and skin to collisions nor do they support artistic control.

In this dissertation, I present interactive algorithms that enable physical simulation of deformable characters with high surface detail and support for intuitive deformation control. I propose a novel unified framework for real-time modeling of soft objects with skeletal deformations and surface deformation due to contact, and their interplay for object surfaces with up to tens of thousands of degrees of freedom. I make use of *layered models* to reduce computational complexity. I introduce *dynamic deformation textures*, which map three dimensional deformations in the deformable skin layer to a two dimensional domain for extremely efficient parallel computation of the dynamic elasticity equations and optimized hierarchical collision detection. I also enhance layered models with *responsive contact handling*, to support the interplay between skeletal motion and surface contact and the resulting two-way coupling effects. Finally, I present *dynamic*

morph targets, which enable intuitive control of dynamic skin deformations at run-time by simply sculpting pose-specific surface shapes. The resulting framework enables real-time and directable simulation of soft articulated characters with frictional contact response, capturing the interplay between skeletal dynamics and complex, non-linear skin deformations.

Acknowledgments

If someone would have told me six years ago that living in North Carolina was going to make a significant change in my life, I would have frowned at first, then I would have needed someone to point out the state on a map, and finally I would have declared that person crazy. But truth be told, the department of Computer Science at the University of North Carolina in Chapel Hill has been a wonderful place to work. Sitterson Hall is filled with friendly and always helpful faculty, staff and fellow students. There are too many too mention, really; but I am thankful for the supportive and stimulative environment that each and every one of them has provided.

I'd like to thank my advisor, Ming Lin for her guidance and for promoting my work at every possible occasion. It really is Ming's accomplishment that so many people know about my research. I am also grateful to to my committee members Miguel Otaduy, Markus Gross, Dinesh Manocha and Anselmo Lastra for their continuing support, technical guidance, and two incredible summers of research in Zürich. The work in this dissertation was published in various papers that would not have been possible without the help of my collaborators Sean Curtis, Paul Mecklenburg, Will Moss, Jason Sewall and Serhat Tekin.

While my work at UNC has always been very stimulating, my life in Carrboro and Chapel Hill really could not have been as entertaining and unforgettable without my friends. I have made friends for life here, more than anywhere else in the world. Henry, Luca, Mary and Paul; I can't imagine any better room mates than you! Brian and Lisa, how can I forget the many evenings together in the kitchen and the countless nights at the table; eating, chatting and laughing our worries and annoyances away. Jason and Sarah, you've always made sure to get me out of the office in time. Out and about, in

town or out of town: it did not matter, because it is the time with great friends that has made these events always memorable. Miguel, even though you weren't in North Carolina, I always had the feeling that you were. I will always cherish the moments we had in Switzerland; be it while drinking, chatting, partying in the streets of Zürich, or hiking through the Swiss Alps! Jur, Lara, Kat, Chris, Marc, Luv, Monika, Simona, and all the people in the lab in Zürich: you are the best friends anyone can imagine.

I would not be who I am now and I would not be where I am now, without the influence of my parents. You have always given me love and support, unconditionally, and no matter how far apart we are. The feeling that I can always depend on you is what brought me this far. Erika en Karolien, my two little sisters, I miss being closer to you. You have probably been the biggest influence on my personality, and I've always been able to confide in you!

Finally, to the most important person in my life, thank you, Allison, for all the wonderful things that we have shared so far, and all that is ahead!

Table of Contents

List of Tables

List of Figures

List of Abbreviations

BV . . . . . Bounding volume

BVH . . . . . Bounding volume hierarchy

CG . . . . . Conjugate gradients method

D2T . . . . . Dynamic deformation texture

DMT . . . . . Dynamic morph target

DoF . . . . . Degree of Freedom

FEM . . . . . Finite element method

GJK . . . . . Gilbert-Johnson-Keerthi algorithm for collision detection

GPU . . . . . Graphics processor unit

LCP . . . . . Linear complementarity problem

LMA . . . . . Linear modal analysis

PBO . . . . . Pixel buffer object

PCA . . . . . Principal component analysis

PCG . . . . . Preconditioned conjugate gradients method

PPU . . . . . Physics processing unit

PSD . . . . . Pose-space deformation

RBF . . . . . Radial basis function

SSD . . . . . Skeletal-subspace deformation

StVK . . . . . St. Venant-Kirchhoff

VBO . . . . . Vertex buffer object

WPSD . . . . . Weighted pose-space deformation

Chapter 1

Introduction

Believable animation of deformable, articulated characters is essential to realistic virtual environments, computer games, and other interactive applications. The feeling of being immersed in a virtual experience, such as a game or an animated feature film, is strongly influenced by the way objects move, interact, and react to the environment around them. Although often a tedious task, animators commonly prescribe these actions by animating the bones of a character or by the process of rigging, to set up control of *abstract* character traits in time. Smiling or frowning are good examples of such traits. For interactive environments such as games, this approach is not sufficient because the characters have to respond to user-generated or unexpected events in their environment. Many current games resort to simple event-based approaches: most of the action is limited to pre-scripted animations triggered by in-game events. As a result, every enemy that is shot down falls in the same pre-recorded fashion; and very different weapons cause the same damage on every wall. Players are left with a game that looks fine, but lacks the sense of realism and variability necessary to make the experience truly immersive. The addition of physical simulation to virtual scenes increases the sense of realism by making virtual characters behave as expected in real life.

Over the last few years, game technology has evolved from supporting only a limited number of rigid bodies to simulating a massive number of interacting rigid and articu-

lated objects in the scene. The feature animation industry has also turned to rigid body simulation to remove some of the burden of hand-animating certain actions, from piling food items to chopping vegetables [Sha07]. But, to make the virtual experience truly immersive, it is important to simulate collisions of *deformable* objects and to generate secondary skin effects. Physical simulation of deformable material allows for the automatic synthesis of effects that are difficult to animate or script otherwise, such as the sagging and vibration of tissue caused by gravity and locomotion, skin wrinkling and muscle bulging. Ideally, the animator should be free from designing the motion that is a direct result of physical laws, and concentrate only on the motion that expresses the intent or emotional state of the characters [Cap04].

The scope of this thesis is real-time simulation of soft articulated characters with secondary skin dynamics. I present a framework that supports coupled skeletal and skin contact response, as well as direct shape and tissue behavior control.

1.1 Deformable Simulation in Computer Graphics

Animation of skin and muscular deformations of human characters and other living creatures has long been one of the most important applications of deformable modeling in computer graphics, notably in feature animation and more recently in increasingly realistic computer games and interactive medical and training applications. Realistic deformation is a complex and subtle phenomenon due to the tightly coupled interplay of bones and musculature governing the deformations.

In order to clarify many of the recurring terms and concepts used throughout this thesis, this section first presents useful terminology and situates my work in the field of deformable simulation. Then, current as well as future applications of my thesis are discussed. Finally, I present the challenges of the design and implementation in my simulation framework.

1.1.1 Terminology: What is Deformable Simulation?

At the basis of this work lies the computer generation of motion and deformation of *soft, articulated characters*. While human simulation is certainly one of the major application areas of this work, I will use the term *character* in a broader sense, referring to any object that undergoes events, motion or deformation in a scene. Examples of such objects can be found across many application fields including molecular dynamics, where molecules undergo docking; robotics and machine assembly, where assembled parts are fit together with moving robotic arms; and computer graphics for video games and feature films, where dynamic objects range from toys and props in a scene, to cartoon-like animals and life-like human characters with realistically deforming skin. A character is said to be *articulated* when it has a skeletal structure of bones connected by joints. Most robotic parts and animal-like and human-like creatures with limbs fall into this category. The skeletal motion of an articulated character is primarily governed by the joint constraints which keep the bones together.

While many works have focused on kinematics, inverse kinematics, dynamic simulation [Bar96] and control [WTF06] of articulated rigid bodies, this work concentrates on *soft* bodies. The term soft usually refers to the fact that the outer surface of an object is *deformable*, usually because there is some underlying layer of elastic tissue, as with human skin. The terms soft and deformable are very tightly connected, but *not* equivalent. An articulated rigid body (e.g. a robot) is deformable because it can undergo some skeletal deformation along its joints (also called skeletal motion). However, it is not soft, because its surface is infinitely stiff.

Global vs. Local Deformation For many years, there has been a lot of research towards efficient computation of global deformations in computer graphics. Efficiency, either for the animator or for a computer simulation, can typically be achieved by providing a mapping between a limited number of degrees of freedom and the final

shape output. In the case of a human animator, these degrees of freedom can be controlled in time through a user interface, for example with free-form deformation techniques [Bar84, SP86, Coq90]. The mapping then turns those controls into frames of animation of the final shape. It allows the animator to work much more efficiently, because he can work at a much higher level of abstraction. In the case of a computer simulation, these degrees of freedom could be the modal coordinates of a modal reduction approach [PW89, JP02, HSO03]. The laws of physics turn the modal coordinates into the final shape output. Due to the compactness of the mapping, the variation in possible final shapes is typically limited. This means that *global deformation modes* such as *bending, twisting* and *shearing* are usually supported, but not highly detailed *local* deformations on the surface. In Chapter 3, I present a method that targets fast simulation of highly detailed deformations of the surface, enabling effects such as *wrinkling* at real-time frame rates. Another example of global deformation is skeletal deformation. Here, the deformation is restricted by the skeletal structure, but the joints allow a limited number degrees of freedom by which an articulated character can deform. As explained later in this section and in Chapter 2, there are many methods available that provide the necessary mapping from skeletal parameters, such as joint angles, to the final shape output.

Animation vs. Simulation There are numerous approaches to generate continuous deformation of a character skin. They can be broadly categorized into two groups: *surface* deformation models on one side, either algorithmic or data-driven, and *skin* or *tissue* deformation models on the other side, the latter usually physically inspired [LCF00].

In the traditional computer animation pipeline, a common data-driven approach is *rigging* of characters such as animals and humans [Mae06]. This process is analogous to setting up a puppet to be controlled by strings. A rigged character's shape can be controlled via a set of abstract parameters, such as `frown' or `smile'. Thus, for

Figure 1.1: Comparison of linear blend skinning to our approach. Left: The fish touches the body of the snake, creating global response and skin deformations. Right: We turn off local skin deformations to show the importance of handling both global and surface response. Notice the highlighted interpenetrations, clearly visible through the fish's mouth.

each instant in time, the animator does not have to position each vertex of the surface mesh; he only needs to set the values of the control parameters. Once the character has been rigged, shape transformations such as a smiling and frowning can be reused and mixed together through shape interpolation. Unfortunately, the process of rigging is rather complicated and almost impossible to automate. The difficulty is mostly due to the inherent complexity of realistic shape deformation. Changes in the shape of a real character are due to the motion of underlying bones, muscles, tendons, as well as to physical forces. Skeletal-subspace deformation (SSD) [MTLT88], also referred to as linear blend skinning, is another surface deformation model. In this technique, the skin deformation is driven by the pose of the underlying skeleton, and a set of blend weights that associate bone contributions to vertices. Surface contact is difficult to model using traditional skinning techniques because the combination of bone transforms and blend weights completely determine the resulting (deformed) shape (Fig. 1.1).

Alternatively, some computer animation researchers have chosen to use physical laws to simulate the underlying deformable tissue for realistic motion and deformation of a character skin [CHP89, GTT89, BW92, JP99, CGC+02a]. These techniques can also be gathered under the name of *deformable simulation*. The use of physical laws to model motion and deformation is especially important for interactive applications such

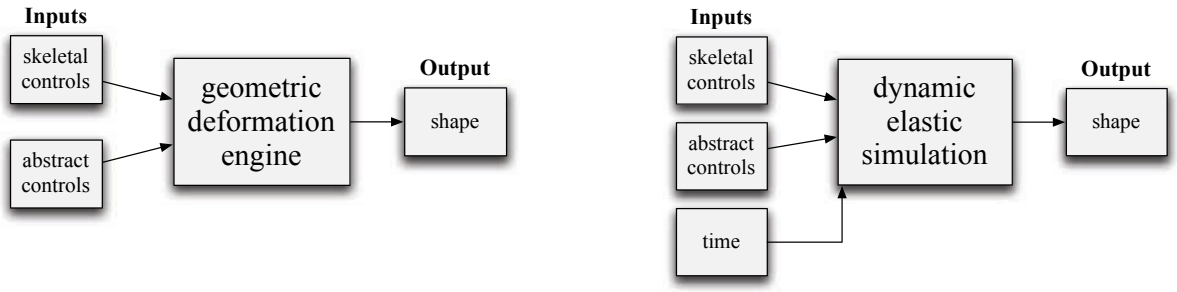


Figure 1.2: Animation and Simulation systems. On the left, the diagram illustrates a traditional character animation system. The deformation engine maps skeletal controls (such as joint angles) and abstract controls (such as *smile*) to the output shape. Animation is achieved by varying the controls over time. The diagram on the right illustrates the work of [Cap04]. Instead of a geometric deformation engine, the system is based on a dynamic elastic simulation. This introduces an explicit time dependence, so the final shape is guided by the input controls, but also varies over time according to the laws of elastic dynamics.

as games, because there is no animator available to respond to the user's input or to unexpected events in the environment of the character. Physical simulation is also useful for applications where secondary effects are too tedious to animate by hand and should be automatically generated, such as in animated feature films. For these applications, the computer can only respond in a realistic way through automation. Physical simulation is a good example of a model that makes such automation possible. The physical equations are encoded by a programmer, from which the computer can easily generate the appropriate motion. Capell [Cap04] has combined rigged character animation and physical simulation in a unified framework, as is illustrated in Figure 1.2.

Accuracy vs. performance Physically based methods in graphics try to mimic biomechanical models of skin tissue and musculature with varying degree of faithfulness. In terms of efficiency versus accuracy, these methods fall into two broad categories. The first category of algorithms aim for accuracy [CZ92, SPCM97, WG97, KGCvB96, ZCCD04, TSIF05, SNF05, LT06, SKP08] by simulating the actions of the individual muscles, bones and tendons in the skin. Interactive physically based approaches trade accuracy for performance [TPBF87, TW88, MT92, PDA01, JP99, JP02, CGC+02a, MG04]. Ex-

changing accuracy for performance has lead to labeling these methods *simplified physical models*. However, the lack of accuracy does not necessarily entail the lack of believable behavior. In deformable simulation for computer graphics, it is often sufficient and sometimes even preferred that a model behaves the way the artist intended, as opposed to exactly in line with physical reality.

Layered Models This work uses a *layered representation* for soft characters in computer animation, as will be elaborated on in Section 1.3.1. The term 'layered' simply refers to the fact that multiple deformation models are employed for different parts of a character's volume, and they are combined through some kind of interface, usually by use of physically inspired forces. For example, Wilhelms [WG97] models several classes of muscles algorithmically with attention to volume conservation; skin is a spring mesh anchored to underlying tissue or bone in appropriate areas. In this dissertation, the layered representation is essentially an integration of articulated body dynamics and skinning with displacement corrections. One of the challenges for modeling soft articulated characters that has not been well investigated previously is the interplay of skeletal motion and surface contact and the resulting two-way coupling effects. Efficient simulation of such interplay is a key design goal of my simulation framework.

Directable Deformations and Control Finally, this dissertation also builds solutions to *control* the shape and deformation behavior of simulated soft characters. It was mentioned earlier that most physically inspired methods use simplified models. These models cannot capture complex non-linear behavior such as muscle bulging, and skin wrinkling. On the other hand, such behavior can be of extreme importance from an artist's point of view. In fact, animators may even want explicit control over the amount and type of secondary effects and deformations in a physical simulation. For example, the deformation of dough in a recent feature film [Sha07], as illustrated in Fig. 1.4. Physically based methods can only provide control through the influence of

Figure 1.3: Snapshots of deformable dynamics in current game engines, compared to methods from this thesis. From left to right, deformable tires with my *dynamic deformation textures* method (2006, Chapter 3), deformable tires in the PhysX engine (2007), deformable torii with the open-source Bullet engine (2008).

forces. While methods that control global deformation modes have been around for a while [WW90], providing control of sculpted deformations for simulation of deformable models has only recently gained attention in graphics research. The work in this dissertation seeks to bridge the gap between geometric example-based methods that have explicit shape control and physically based approaches. This is the topic of Chapter 5.

1.1.2 Applications

Games Simulation of multi-body dynamics has become extremely important in the games and entertainment industry. Game developers are constantly pushing the limits of technology when it comes to achieving realism in gameplay, be it in graphical rendering or in simulation of physical phenomena. It was only a few years ago that games made the jump from 2D to 3D environments. Quickly, games have evolved from supporting only a limited number of rigid and rigidly skinned articulated bodies to simulating a massive number of interacting rigid objects in the scene. Although the focus has been mainly on rigid phenomena, games are now stepping into the realm of deformable physical phenomena that are computationally more challenging than rigid effects, such as fluids, fracture, and soft body behavior. The need and desire for such realism has become apparent from the upcoming support for deformable simulation in many popular physics

engines, such as the open-source Bullet [Bul08], Intel's Havok [Hav08] and NVIDIA's PhysX [Phy08]. The method that I propose in Chapter 3 actually predates some of the support for deformable bodies in the physics engines illustrated in Fig. 1.3. In addition, it is clear that simulation of physical phenomena for games has become a hot topic, judging from the emergence of hardware-based solutions that aid in simulating these phenomena in real-time. For example, GPUs have recently been used to run a 3D Navier-Stokes fluid solver on a $128 \times 64 \times 64$ grid at 120 to 180 fps, and NVIDIA's PhysX hardware solution has been applauded in the gaming industry as a crucial component for achieving real-time explosions with dynamically fracturing debris and complex smoke effects. NVIDIA has also recently added support on the PhysX chip for real-time soft body behavior. It is clear that the industry is quickly developing a growing need for smart and efficient algorithms that enable even more realistic effects such as fast soft character interaction or skin wrinkling. Likewise, there has been a radical move towards parallel architectures geared specifically towards the games market, with examples such as Larrabee [SCS+08] and NVIDIA PhysX. The algorithms in Chapters 3 and 4 are specifically geared towards parallel efficiency, and I demonstrate that they map well to such architectures. Luckily, a lot of effort has been put into supporting higher level programming models such as C++ and CUDA [CUD07] for these novel parallel architectures, which simplifies the task of porting the techniques of this dissertation to these platforms.

Animated Feature Films Physical simulation has taken an important role in the film industry, not only for producing special effects, but also as an aid to the animation process for animated feature films. Computer animation has one chief advantage over traditional hand-drawn animation. By providing a modeling layer between the animator and the output images, computer animation enables the artist to express the animation more succinctly, while ignoring unnecessary details. This leads to an improvement in animation quality and animator efficiency because his efforts are spent working at a

Figure 1.4: Snapshots from Pixar s Ratatouille. Dough rolling effect (top), requiring shape control in combination with physical deformation simulation. Several stages of the egg during the cracking (bottom). The techniques in Chapter 5 can be used to achieve deformation control.

higher level of abstraction. Furthermore, this dissertation presents methods to include deformable simulation and control in the modeling layer, supporting animators in a number of ways. First, this enables automatic generation of secondary effects of the animated shapes; when a virtual character with a fat belly is animated through the scene, we would expect to see his belly bounce and sway in correlation with changes in momentum caused by the animated path. Physical simulation is a good way to avoid animating this type of effect by hand and to make it more believable. Second, it enables passive deformable objects in the scene to react when in contact or influenced by explicitly animated characters. In Pixar's animated feature *Ratatouille*, the deformation of a blob of dough being deformed by an animated rolling pin was computed with physical simulation [GRPS07], as was the deformation of an egg yolk when an egg is cracked

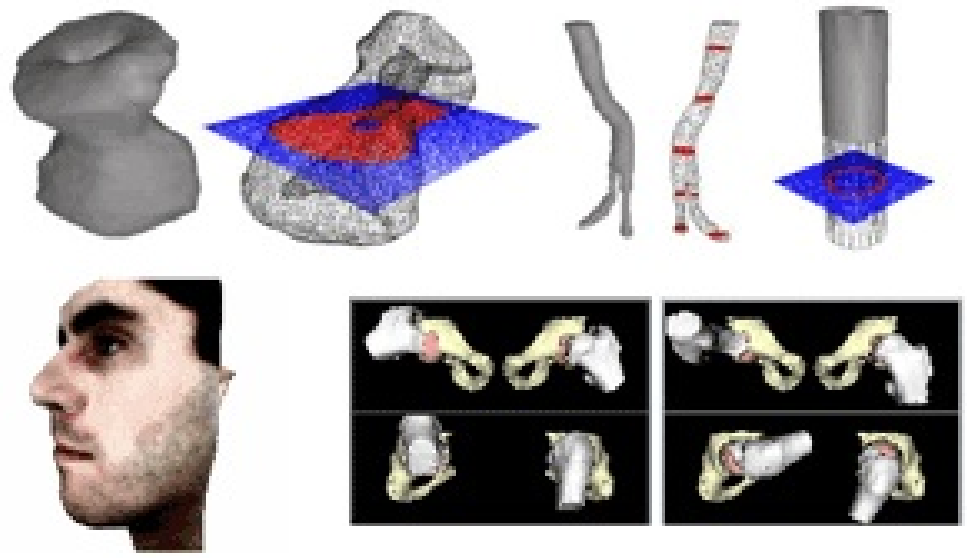


Figure 1.5: The Co-Me Project. The technologies in this thesis have been beneficial to the development of interactive methods for generalized surgery simulation and training.

open (Fig. 1.4). Finally, there seems to be a definite need for deformation control in animated features. In *Ratatouille*, the dough had to flatten under and bulge in front of the rolling pin, and it needed to form specific shapes at specific times based on creative direction [GRPS07]. The techniques presented in Chapter 5 can be used to provide such control, using intuitive animator controls.

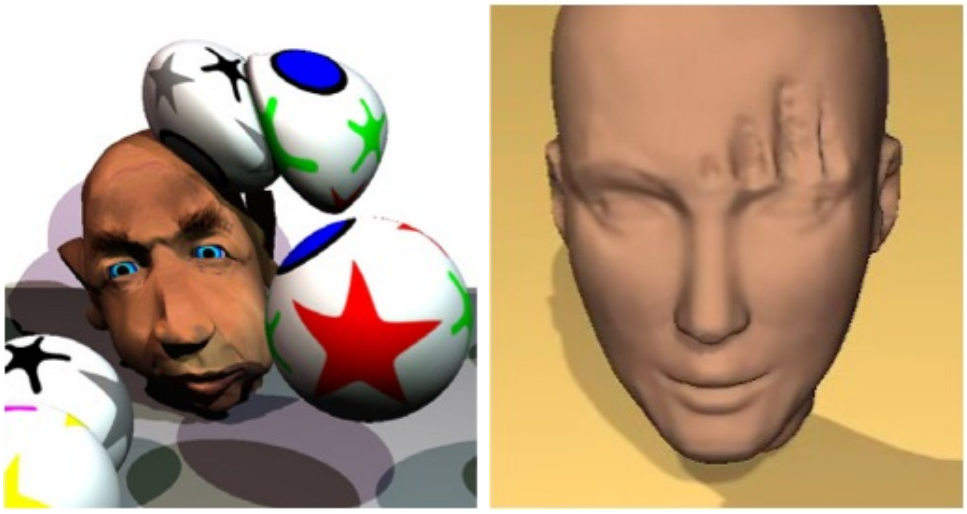
Surgical Training The technologies in this thesis have been beneficial to the development of interactive methods for generalized surgery simulation and training. For example, the Co-Me project [COM] of the National Center of Competence in Research (NCCR, Switzerland) aims to utilize information technology for improved health care. Based on the work in this dissertation and on other research in deformable and fracture modeling, collision handling, and point-based computer graphics, the CoMe project is developing techniques for real-time surgical simulators, supporting applications such as hysteroscopy simulation, simulation of stent placement, craniofacial surgery simulation, orthopedic surgery planning and modeling of soft tissue (Fig. 1.5).

Haptics Haptic rendering of forces and torques between interacting objects, also known as 6 degree-of-freedom (DoF) haptics, has been demonstrated to improve task performance in applications such as molecular docking, nanomanipulation, medical training, and mechanical assembly in virtual prototyping [LO08]. Haptic display of complex interaction between two deformable models is considered especially challenging, due to the computational complexity involved in computing contact response and performing proximity queries, including collision detection, separation distance, and penetration depth, between two deformable models at force update rates. The algorithms in this thesis focus on interactive deformations with support for contact and hence they apply to the domain of haptic applications. This was briefly investigated in the context of this thesis [GTO+07].

1.1.3 Challenges

One of the key challenges of deformable simulation is to satisfy the conflicting requirements of real-time interactivity and physical realism. In order to achieve realism, sometimes the first requirement is to achieve sufficient *deformation detail*, which then means that *real-time collision detection* becomes a much harder task, and *robust contact response* becomes problematic. Finally, marrying complex simulation algorithms with the needs and wishes of animators requires *intuitive control* methods. In this section, I will briefly elaborate on these sub-challenges.

Deformation Detail Simulation of detailed secondary effects such as skin wrinkling requires many degrees of freedom on the surface, up to tens of thousands of deformable vertices. This is at least an order of magnitude more than current interactive deformation systems. Up until recently, systems such as in Figure 1.6(a) have been able to show interactive rate deformable simulation in the range of hundreds of deformable elements, but at least a few thousand elements are required to show complex deformation detail,



(a) (b)

Figure 1.6: Deformation Detail. The Shape Matching algorithm [MHTG05], here illustrated on Fig. (a), can simulate hundreds of deformable elements in real time, but many more are required to show detail such as skin indentations simulated with my framework. The face model in Figure (b) has $40,000$ deformable surface vertices.

for example deforming tire threads or generating skin indentations as was done by my system in Figure 1.6(b).

Efficient Collision Detection As the complexity of the models goes up, collision detection tends to take up a considerable chunk of computation time in deformable dynamics systems because any precomputed acceleration data-structure has to be updated while the objects are deforming [TKH$^+$05]. Therefore, a fast collision detection algorithm targeted towards dynamic deformation dynamics is essential to obtain a real-time system. Likewise, handling self-collisions is a significant challenge, as they occur commonly for deforming articulated characters. Collision detection should be very efficient. Ideally, it should be independent of surface resolution.

Responsive Contact Handling Collisions and interaction with the scene not only give rise to surface deformations, but also cause global skeletal deformations, and they

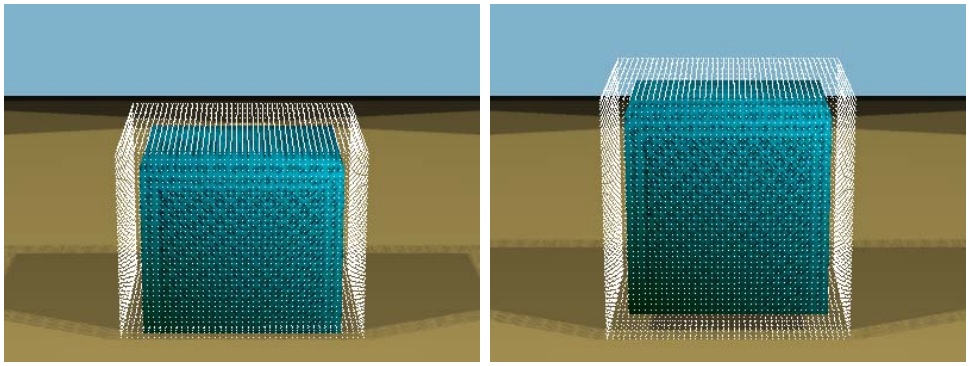


Figure 1.7: Responsive Contact Handling. The figure shows what can go wrong when contact handling is not responsive for the entire model. On the left, the outer surface layer, drawn with white dots, is prevented from violating the ground plane constraint, but the object as a whole, with the blue inner core layer is not. The figure on the right shows the correct behavior, simulated with my system.

influence the global motion of the objects. The overall robustness of a simulation framework, such as the one presented here, can only be guaranteed if the contact response algorithm is robust enough to account for these effects simultaneously. The contact handling algorithm has to be *responsive* such that skeletal and global motion are handled simultaneously and naturally with surface deformations. Figure 1.7 shows an example of what can go wrong when contact handling is not responsive for the entire model. With layered model methods, it often happens that the surface layer is prevented from violating the constraint, but the object as a whole is not prevented from doing so.

Deformation control Regardless of the realism that can be achieved by numerical simulation, animators and game content creators crave the ability to steer the behavior of characters and their material properties. If a character's skin is supposed to wrinkle on the forehead but not on the back, there has to be a simple way to express that behavior. Or, as in Figure 1.4, animators may prefer certain shapes of a bulging dough ball over others. The challenge is to find intuitive tools that enable animators and modelers to specify *how and when* materials are to deform.

1.2 Thesis

My thesis is:

> Using appropriate *layered* representations and *simplified physics-based models*, it is possible to *generate* and *control* believable articulated soft-body behavior at *interactive* rates.

In support of this thesis, I present a unified framework for real-time modeling of soft objects with up to tens of thousands of degrees of freedom. This framework complements animated articulated characters with skeletal dynamics and detailed secondary skin effects. My framework also supports combined skeletal and skin contact response, where the coupled nature of (global) skeletal deformations and (local) skin deformations is gracefully captured.

I propose the use of *layered models* to reduce the computational complexity. Layered models have been proposed previously, but I introduce novelty on two fronts:

1. The three dimensional deformations in the deformable layer are mapped to a two dimensional domain to reduce complexity even further. A re-parameterized version of this domain onto a regular grid, called *dynamic deformation textures* (Chapter 3), is very amenable to parallel computation of the dynamic elasticity equations.

2. I enhance layered models to support simulating the interplay of (global) skeletal motion and surface contact and the resulting two-way coupling effects. I apply *physically-inspired simplification* to drastically reduce the computational complexity of previous contact response methods for deformable dynamics.

In addition, I also propose a method that enables intuitive control over shape and skin behavior at run-time. This approach bridges the gap between artist-controlled an-

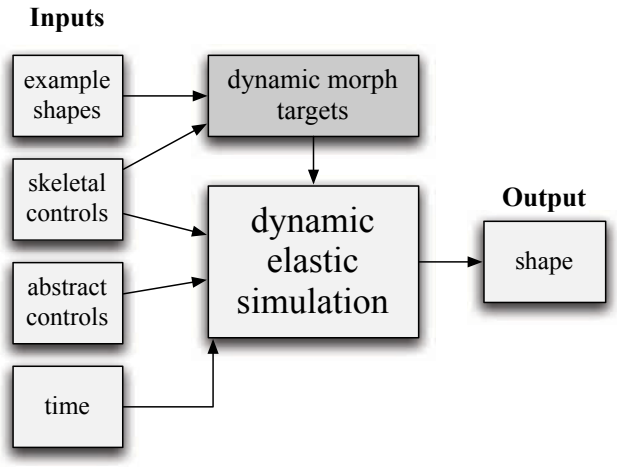


Figure 1.8: Dynamic Morph Targets Pipeline. To support artistic deformation control, I propose dynamic morph targets (Chapter 5) to complement the traditional animation and simulation pipelines. Dynamic morph targets are created from artist-provided examples (pairs of example shapes and associated skeletal poses) and enhance the runtime dynamic simulation.

imated behavior, shape control, and computer-generated secondary skin deformation effects. My approach complements the traditional animation pipeline with intuitive control metaphors to support directable deformations: globally with traditional skeletal animation and locally with *dynamic morph targets* (presented in Chapter 5). Dynamic morph targets enable animators to express the way material deforms in particular configurations. I achieve this by deriving a pose-dependent material model that is able to retarget artist-provided example inputs to unforeseen motions. Figure 1.8 illustrates how I complement the traditional animation and simulation pipelines (Fig. 1.2) to support secondary skin dynamics using dynamic morph targets.

1.3 Main Results

In this section, I present the main results of this dissertation in detail, as summarized in the previous section. I categorize the results in the areas of layered deformable models, pose-space skin dynamics, parallelization and physically-based simplification

for efficient skin dynamics, hierarchical collision detection, responsive contact handling, and directable deformations.

1.3.1 Layered Models for Soft-body Simulation

I propose the use of layered models to reduce the computational complexity of soft articulated character simulation. My approach enables interactive simulation of objects with tens of thousands of deformable surface points. I achieve such performance by complementing layered models with several novel methods. First, I propose *dynamic deformation textures*, a method that enables fast parallelized computations. I also present physically-based skin dynamics approximations that reduce the complexity to enable interactive frame rates while preserving plausible behavior.

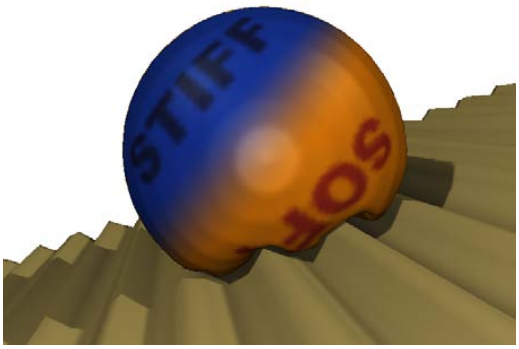
More specifically, I model each deformable object as a *core* covered by a layer of (possibly heterogeneous) deformable material. This layered representation enables modeling of:

1. Detailed small scale deformations over large regions of the object's surface.

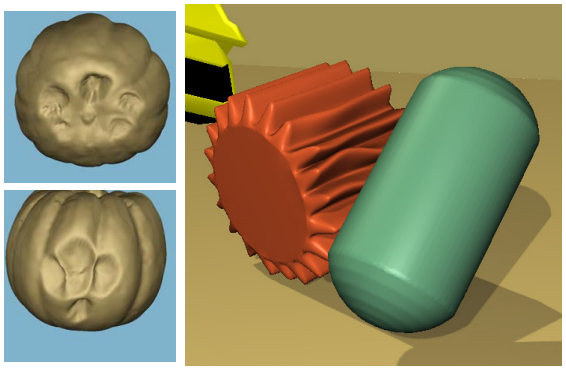
2. Global deformations of skeletal nature.

3. During contact response, the dynamic interplay between the (global) skeletal motion of the character and surface deformations, as shown in Fig. 1.10.

For the skin dynamics model, I propose to start from a sound physics-based method by formulating the dynamic motion equations of soft articulated characters using Lagrangian continuum mechanics [GPS02, Sha89], discretizing the continuous deformable layer with a finite element mesh (FEM) with linear tetrahedral elements. With this layered model, I have successfully captured large deformations that reach as much as $30 - 40\%$ of the radii of the objects, as illustrated in Fig. 1.9.



(a) Efficient decoupled implicit integration enables fast simulation of a heterogeneous cylinder. The cylinder has 21,000 deformable vertices (161,000 tetrahedra).



(b) Left: Detail view of the rich deformations produced on the top pumpkin (30,000 vertices and 183,000 tetrahedra) during contact. Right: A dropped cylinder produces rich dynamic deformations on the ridges of a gear (29,000 vertices and 173,000 tetrahedra).

Figure 1.9: Simulation of Heterogeneous Materials and Examples of Detailed Deformations.

1.3.2 Pose-space Skin Dynamics

I present novel formulations of elastic deformations in body space for non-articulated objects (Chapter 3)) and in pose-space for articulated characters (Chapter 4). It is well known that linearization of the elasticity laws does not correctly model large deformations, because linear strain models are not rotation invariant (see Section 2.1.2). Akin to previous co-rotational methods for non-articulted objects [TW88, MG04] and methods using skin displacement corrections [KJP02, JT05] for articulated characters, I solve this problem by expressing skin strain in a floating frame of reference that is aligned with the rest configuration (or pose-space) of the articulated character. I either track the bone state during the simulation to transform bone space deformations to world space, or I derive bone kinematics from a character animation or motion capture sequence. In contrast to previous approaches [MG04, CGC+02a], my model also optionally considers centripetal and Coriolis forces introduced by the inertia of the deformable layer. With my formulations, the motion equations derived from Lagrangian mechanics naturally produce the desired interplay between skin and skeleton.

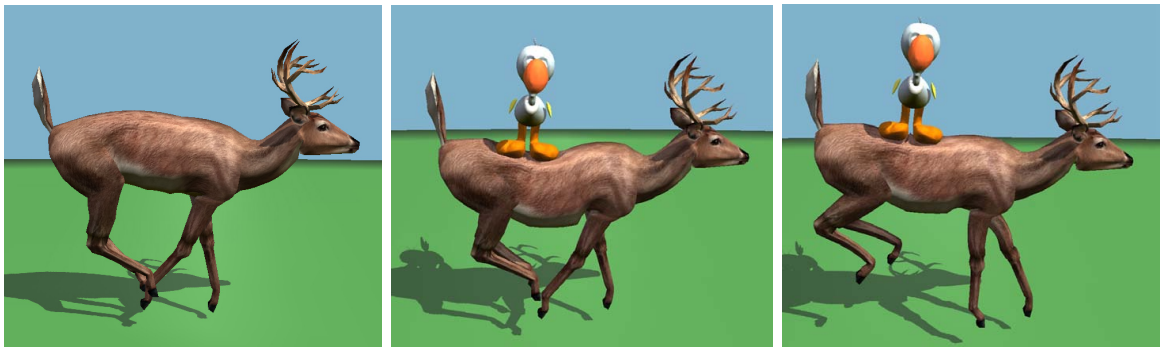


Figure 1.10: Interactive Deformation of an Articulated Deer. The deer, consisting of 34 bones and 2 755 deformable surface vertices is being deformed interactively (almost 10 fps on average) by a rigid bird model. The interplay between small-scale contact deformations and the skeletal contact response is successfully captured.

1.3.3 Parallelization and Physically-based Simplification

A key advantage of my layered model is the reduction of the computational complexity that comes from the simplification of the interior volume dynamics. In fact, by only retaining 6 degrees of freedom (DoFs) per rigid bone, I focus the computational power on the simulation of detailed skin deformations as opposed to volume deformation while I still maintain the ability to model global skeletal motion.

Using linear FEM discretization of the displacement field of the deformable skin in pose-space (Section 3.1.1), I map 3-dimensional deformations to a 2-dimensional parametric domain. This enables a highly parallelized algorithm, called *dynamic deformation textures* to compute elasticity dynamics of a layer of deformable material. Together with careful approximation of Schur complements (Section 3.2.2), this formulation enables efficient decoupled simulation of highly detailed dynamic objects that have tens of thousands of surface elements with two-way coupling of global object motion and surface deformation at interactive rates.

Dynamic Deformation Textures I propose a re-parameterization of the surface deformation field onto a 2-dimensional domain. If the topology and shape of the surface

geometry allows for it, this parameterization can correspond to a mapping of the surface deformation field to a regular grid, called *dynamic deformation textures*. The regular grid implicitly defines the meshing of the deformable layer, which, after FEM discretization leads to a *regular* sparse SPD system on the regular data grid. Such systems can be efficiently solved on massively parallel streaming architectures because branching and pointer chasing are eliminated due to the regularity of the system. In Section 3.6, I demonstrate the performance advantage in a fast GPU implementation that employs texture memory to store the parameterization and fragment shaders to compute linear system solver kernels. This implementation is up to an order of magnitude faster than other methods that enable large time steps, for single-bone objects.

Efficient Dynamic Updates by Physically-based Approximation and Parallellization As mentioned in Section 1.1.3, one of the key challenges of achieving realistic soft character dynamics is to capture the physical interplay between skeletal dynamics and local skin deformation, leading to a tightly coupled dynamic system. I reformulate the motion equations such that the solution of this coupled and inherently non-parallel problem can be split into a massively parallel subproblem solve, followed by a coupling step to update the global (skeletal) dynamics. Full solution of the skeletal dynamics of a character with k bones and n surface points is known to have brute-force $O(nk)$ complexity [Bar96]. A key contribution of this thesis is the reduction of this complexity to $O(n+k)$ (in practice) while preserving physically plausible global and local deformation effects. I achieve this by:

1. Splitting of parallellizable skin computations from serial skeletal computations by exploiting Schur complements, also known as matrix condensation [BNC96].

2. Employment of a fast approximate inverse of the skin inertia matrix to accelerate computation of Schur complements (see Section 4.3.1).

3. Computation of a fill-reducing reordering of the condensed system matrix and

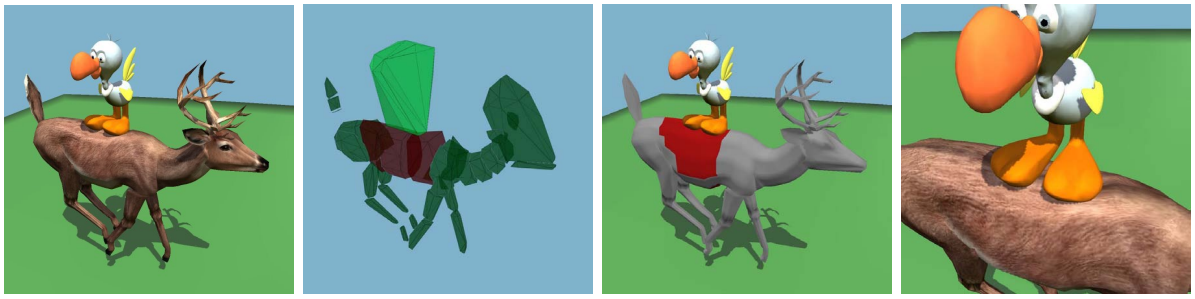


Figure 1.11: Layered Representation and Collision Detection. From left to right: Contact between the bird and the deer, with skin deformations on the back of the deer; Proxies used for hierarchical pruning of collision queries, with potentially colliding proxies of the deer highlighted in red; Triangles influenced by the potentially colliding bones (in red) are the only ones passed to the image-based collision detection algorithm; The resulting detail around the contact area.

off-line pre-computation of its symbolic factorization.

I demonstrate that the elastic and skeletal update can be separated by observing that the elastic energy due to pose-space strain is only dependent on the degrees of freedom of the skin layer. Fast and massively parallel solvers can be exploited to solve the skin update. In Chapter 3, I show that coupled layered dynamics of a rigid core with soft skin can be implemented very efficiently on a parallel architecture such as graphics processing units (GPUs) to simulate highly detailed surfaces at interactive frame rates, while also minimizing costly communication between GPU and CPU host.

The intuition for the approximation of the inverse of the skin inertia matrix is physically based. Even though this approximation yields surface deformation velocities that differ slightly from those of the full solution, it does not jeopardize the fulfillment of joint or contact constraints. Moreover, the approximation still captures the coupling of the elastic forces in the deformable layer that account for the interplay between (global) skeletal motion and surface deformation during contact dynamics (Sections 3.4 and 4.2.3).

1.3.4 Two-stage Hierarchical Collision Queries

I present a two-stage collision detection algorithm that exploits low-resolution acceleration data structures. These data structures are constructed from proxy geometry for the character's bones (Section 4.3.3). They can be efficiently updated at run-time to check for potentially colliding surface patches.

In particular, I adopt a fast image-based algorithm that exploits the layered representation of soft characters. Collision detection is performed in two steps:

1. Identification of contact patches with object-space techniques using low-resolution proxies [EL00].

2. High-resolution skin surface interference detection and collection of colliding skin vertices using image-space techniques with the aid of graphics hardware.

My method shares the two-step approach of others used for rigid bodies [OJSL04]. Unlike these methods, I perform collision handling of deformable objects and compute contact information for many colliding surface points. My collision query algorithm also performs hierarchical pruning to eliminate large portions of the objects from collision queries by exploiting the skeletal nature of the deformation. An example of the pruning can be seen in Fig. 1.11. The worst-case cost of the collision detection is $O(n)$ for a pair of tested objects with n surface nodes; the actual cost depends only on the size of the contact area. Once contact areas have been identified, I exploit image-space techniques in a GPU-accelerated surface interference algorithm to make the cost of the collision detection sub-linear in the number of surface nodes in practice. My algorithm achieves this by parallelizing surface node interference detection, based on orthogonal projection of potentially intersecting surface patches onto low-resolution contact planes (Section 3.3).

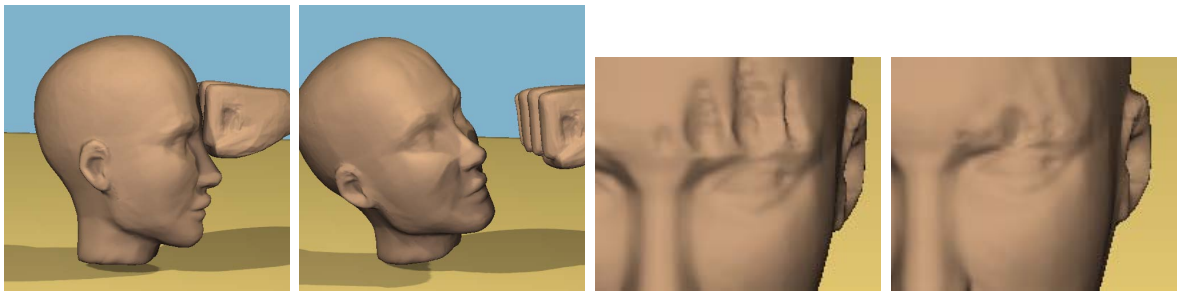


Figure 1.12: Deformations of a Virtual Head. Left: A fist hits a deformable head (attached by springs in the neck area), producing both local deformations and global motion. Right: Detail of the deformations produced near the eyebrow by the impact.

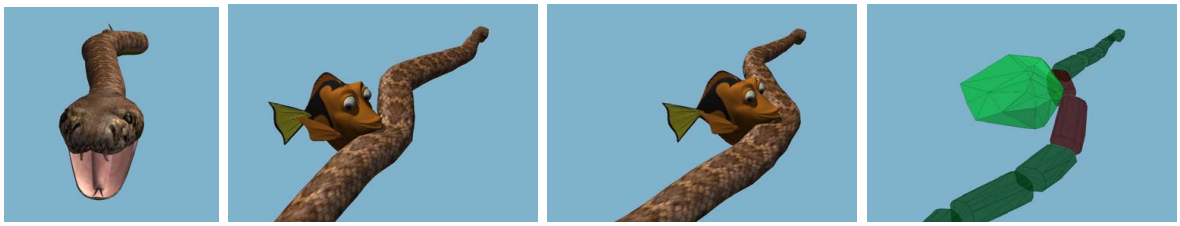


Figure 1.13: Skeletal Deformations of a Soft Snake. Simulation sequence with a fish touching the snake, showing the global deformation of the snake. The last image shows the proxies for collision detection.

1.3.5 Fast Contact Response for Soft Articulated Characters

As mentioned before, I aim to achieve highly responsive contact response that accounts for both the global (skeletal) effect and the local surface deformations. For example, when a fist punches a face (Fig. 1.12), the head motion as well as the eyebrow deformation should be handled in a hybrid contact resolution framework. Another example of such responsive behavior can be seen in Figure 1.13 where a fish causes global skeletal motion of a snake as well as local deformation of its skin. I propose a novel efficient and highly parallelizable solution that enables robust contact handling in the simulation with very large time steps, based on Lagrange multipliers, implicit integration, and physically-based approximation of elastic deformation forces. I also propose *approximated anticipation of the skeletal response* to reduce the typical $O(mnk)$ complexity for deformable characters with m contacts, n vertices and k bones to $O(m+n+k)$ in practice.

Responsiveness In many previous algorithms, contact response is computed by explicitly integrating the constraint forces, which is equivalent to applying an instantaneous change of momentum to the surface nodes at the end of each time step. Unfortunately, with those methods the elastic deformation forces are unable to counteract the momentum of the core upon collision, and the core may penetrate the constraints (Fig. 1.7). In Section 3.4, I propose and describe the computation of the collision impulse through implicit integration which produces a robust and responsive reaction of the object's core with large time steps. In Section 4.2.3, it is shown that the same technique can be applied to ensure responsive contact of the skeleton of a soft articulated character.

Skeleton response anticipation Previous existing methods for solving multi-body dynamics of *rigid* articulated characters with joint and contact constraints propose the anticipation of contact constraints to resolve contact impulses [Bar96]. However, this approach has a worst-case cost of $O(mk)$ for a scenario with m contacts and k joint constraints. In the context of this thesis, I exploit the use of equality contact constraints, the fact that each colliding surface node yields one constraint, and the approximation of skin force Jacobians. Combining these techniques, I propose *anticipation of skeleton response* for *soft* articulated characters. I first solve for the contact impulses while anticipating the skeletal response under influence of the joint constraints. The overall computational cost of expensive contact constraint anticipation is thereby reduced from worst-case $O(mnk)$ complexity to $O(m + n + k)$ in practice. This is demonstrated in Section 4.3.4.

1.3.6 Control of Deformations with Dynamic Morph Targets

I present a method to control the behavior of elastic, deformable material in a dynamic simulation. In Chapter 5, I introduce *dynamic morph targets*, the equivalent in dynamic simulation to the geometric morph targets in (quasi-static) modeling. Dynamic morph

targets define the pose-dependent physical state of soft objects, including surface deformation and elastic and inertial properties. Given these morph targets, my algorithm then derives a dynamic model that can be simulated in time-pose-space, interpolating the dynamic morph targets at the input poses. My approach seeks to bridge the gap between geometric example-based methods and physically based approaches. It easily integrates with current modeling and animation pipelines: at different poses, an artist simply provides a set of dynamic morph targets. Whether these input states are physically plausible is completely up to the artist. The resulting deformable models expose fully dynamic, pose-dependent behavior, driven by the artist-provided morph targets, complete with inertial effects. The success of dynamic morph targets relies on three key results:

- A pose-space method for interpolation of simple elastic deformation models that allows the artist to author complex nonlinear deformation behavior.
- A compact way of interpolating skin geometry, elastic forces, and their derivatives, all in a unified manner using pose-space polynomial interpolation.
- The extension of the method to support modal reduction, resulting in a very efficient implementation that is linear in the number of coefficients of the force polynomial.

The main advantages of my method over previous approaches are three-fold: *quality of deformations, dynamic behavior* and *computational efficiency*. Although my method is physically based, it avoids expensive modeling of musculature or tendon influences, and instead relies on physical constitutive models of deformable material to minimize skin pinching artifacts and bypass complex rigging requirements that are common to purely geometric approaches. The use of such constitutive material models also enables response to external forces and inertial effects in dynamic simulations. Due to performance requirements, one is usually restricted to linear or quasi-linear models that

cannot model pose-dependent effects such as bulging and wrinkling. Instead, I guide dynamic simulations with dynamic morph targets discrete pose-space examples of skin properties and deformations.

The result is an efficient framework for directable physically-based skin deformations that extrapolates well to unforeseen poses. Soft characters that have been complemented with dynamic morph targets can be plugged into existing dynamic simulation engines, either forming interactive, deformable content in real-time games or providing secondary dynamic effects for kinematically-driven characters in feature animation films. My method also facilitates certain time-consuming rigging procedures, by providing a physically based approach to resolve co-articulation deficiencies in traditional skinning methods, such as in shoulder regions, fully automatically. These results are demonstrated with my real-time implementation described in Section 5.4.

1.4 Organization

The rest of this dissertation is organized as follows. The next chapter summarizes related work in deformable simulation and control. Chapter 3 presents *dynamic deformation textures* for fast simulation of soft objects with a rigid core and soft skin, including support for coupled contact handling. Chapter 4 extends my approach to *soft articulated* characters with fast contact handling. Chapter 5 introduces a method to *control* the deformations of soft character skin based on *dynamic morph targets*. Finally, Chapter 6 gives a summary of the thesis conclusions and discusses future research.

Chapter 2

Related Work

Creating appealing and realistic animation of deformable characters is a multi-disciplinary problem. In this chapter, I present both related work in the domains of character simulation, animation, contact handling and control with emphasis on *deformable* characters.

2.1 Simulation of Deformable Bodies

Since Lasseter's discussion of *squash and stretch* [Las87] and, concurrently, Terzopoulos et. al's seminal paper on elastically deformable models [TPBF87], numerous researchers have participated in the quest for the visually and physically plausible animation of deformable objects and fluids. This inherently interdisciplinary field elegantly combines Newtonian dynamics, continuum mechanics, numerical computation, differential geometry, vector calculus, approximation theory and computer graphics (to name a few). In this section, I will discuss physical simulation of deformable, elasto-plastic material, focusing on previous research that is directly related to this thesis. For comprehensive summaries, I'd like to refer readers to [GM97, NMK+05, TKH+05].

2.1.1 Continuum Elasticity

A deformable body is typically represented by its undeformed shape (also called equilibrium configuration, rest or initial shape) and by a set of material parameters that define

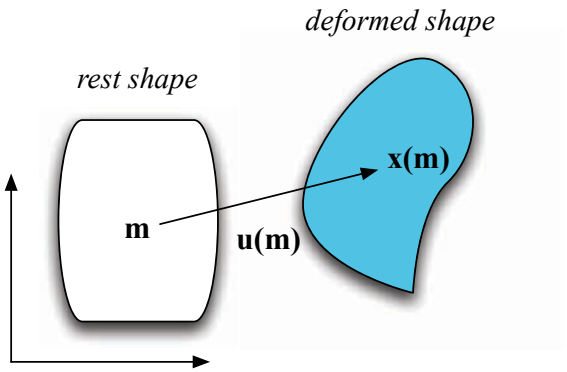


Figure 2.1: Continuous Displacement Field. Deformation of an object in rest shape causes a material point originally at $\mathbf{m}$ to be transformed to a new position $\mathbf{x}(\mathbf{m})$ through the continuous displacement field $\mathbf{u}(\mathbf{m})$.

how it deforms under applied forces. If we think of the rest shape as a continuous connected subset M of $\mathbb{R}^3$, then the coordinates $\mathbf{m} \in M$ of a point in the object are called material coordinates of that point. In the discrete case M is a discrete set of points that sample the rest shape of the object. When forces are applied, the object deforms and a point originally at location $\mathbf{m}$ (i.e. with material coordinates $\mathbf{m}$) moves to a new location $\mathbf{x}(\mathbf{m})$, the *spatial* or *world coordinates* of that point. Since new locations are defined for all material coordinates $\mathbf{m}$, $\mathbf{x}$ is a vector field defined on M. Alternatively, the deformation can also be specified by the displacement vector field $\mathbf{u}(\mathbf{m}) = \mathbf{x}(\mathbf{m}) - \mathbf{m}$ defined on M (see Fig. 2.1). From $\mathbf{u}(\mathbf{m})$ the elastic strain ϵ is computed. This quantity is dimensionless; in the (linear) 1D case it is simply $\Delta l/l$. A spatially constant displacement field represents a translation of the object with no strain. Therefore, it becomes clear that strain must be measured in terms of spatial variations of the displacement field $\mathbf{u} = \mathbf{u}(\mathbf{m}) = (u, v, w)^T$.

Popular choices in computer graphics are

$$\epsilon_G = \frac{1}{2}(\nabla\mathbf{u} + [\nabla\mathbf{u}]^T + [\nabla\mathbf{u}]^T\nabla\mathbf{u}) \tag{2.1}$$

$$\epsilon_C = \frac{1}{2}(\nabla\mathbf{u} + [\nabla\mathbf{u}]^T) \tag{2.2}$$

where the symmetric tensor $\epsilon_G \in \mathbb{R}^{3\times3}$ is Green's *non-linear* strain tensor and $\epsilon_C \in \mathbb{R}^{3\times3}$ its linearized version, Cauchy's *linear* strain tensor. The gradient of the displacement field is denoted by the 3 by 3 matrix $\nabla\mathbf{u}$.

A constitutive law (or also called material law) is used for the computation of the symmetric internal stress tensor $\sigma \in \mathbb{R}^{3\times3}$ for each material point $\mathbf{m}$ based on the strain ϵ at that point. Most computer graphics papers use Hooke's linear material law

$$\sigma = \mathbf{E} \cdot \epsilon, \tag{2.3}$$

where $\mathbf{E}$ is a rank four tensor which relates the coefficients of the stress tensor linearly to the coefficients of the strain tensor. For isotropic materials (a material which has the same mechanical properties in all directions), the coefficients of $\mathbf{E}$ depend only on Young's modulus and Poisson's ratio. Two very common elastic models used in computer graphics are:

- The fully linear elastic model, using the linear Cauchy strain tensor ϵ_C and Hooke's linear material law.
- The St. Venant-Kirchoff elastic model, abbreviated as StVK, using the non-linear Green's strain tensor in combination with Hooke's linear material law.

The choice of elastic model influences the physical accuracy and also the computational complexity of the resulting elastic model.

2.1.2 Finite Element Method

To date, Finite Element Methods (FEM) have often been used to discretize the partial differential equations that describe the dynamics of continuum deformable, elasto-plastic models, and result in (generally nonlinear) second-order ordinary differential motion equations [Sha89].

The accuracy and computational complexity of a method depend on multiple factors in the design of a simulation model. More specifically, they depend mainly on the choice of elastic force model, time integration method and spatial discretization method. Many papers in computer graphics use the *explicit* Finite Element Method, where both masses and internal forces are lumped to the vertices [OH99, DDCB01, MDM$^+$02, MG04]. This choice relates to the *spatial* integration method, it is not to be confused with explicit time integration. The explicit FEM method can be integrated in time both explicitly or implicitly.

Implicit vs. Explicit Time Integration Numerical time integration of ordinary differential equations is used to advance the state or in other words *simulate* the motion equations that follow from physical laws. The survey paper of Hauth *et al.* [HES03] is an excellent overview in the context of deformable modeling in computer graphics. *Explicit* integration methods are easy to implement but are only conditionally stable because they blindly extrapolate force values into the future. As a consequence, increasingly smaller time steps have to be used for increasingly stiffer materials. This obviously affects the overall computational complexity of such scheme. On the other hand, *implicit* schemes express unknown force values implicitly in the equations. In other words, these quantities are implicitly given as the solution of a system of equations. For example, the implicit (or backward) Euler scheme is stable for arbitrarily large time steps. This gain comes with the price of having to solve an algebraic system of equations at each time step, which has a negative effect on the computational complexity of implicit schemes.

A combination of the St. Venant-Kirchhoff elasticity model (see Section 2.1.1) with explicit time integration methods has been used successfully to produce fast simulations of soft deformable bodies of moderate complexity for animation [ZC99] and for medical applications [PDA01]. O'Brien *et al.* [OH99, OBH02] present a FEM based technique for simulating brittle and ductile fracture in connection with elasto-plastic materials. They

use tetrahedral meshes in connection with linear basis functions and Green's strain tensor. The resulting nonlinear equations are solved explicitly and integrated explicitly. The method produces realistic and visually convincing results, but it is not designed for interactive use. In addition to the strain tensor, they use the so-called strain rate tensor (the time derivative of the strain tensor), to compute damping forces.

As long as the equation of motion is integrated explicitly in time, non-linear elastic forces resulting from Green's strain tensor can be computed fairly efficiently. The nonlinear formulas for the forces are simply evaluated and used directly to integrate velocities and positions as in [OH99]. However, as mentioned earlier, implicit integration or quasi-static approximation methods enable significantly larger time steps than explicit integration methods, as proven by many researchers [BW98, TPBF87, TSIF05, MG04, BNC96, JP99]. However, for implicit integration, a system of algebraic equations needs to be solved at every time step. The use of Cauchy's linear stress tensor can yield a linear algebraic system which can be solved more efficiently and more stably than non-linear ones. Unfortunately, linearized elastic forces are only valid for small deformations. Large rotational deformations yield highly inaccurate restoring forces (see Fig. 2.2).

Co-rotational Methods and Stiffness Warping To eliminate these artifacts, Müller *et al.* extract the rotational part of the deformation for each finite element and compute the forces with respect to the non-rotated reference frame [MDM$^+$02, MG04]. In his method, named *Stiffness Warping*, the rotation of each tetrahedral element with respect to the rest configuration is estimated from the deformed vertices, by performing a polar decomposition of the matrix that describes the transformation of the tetrahedron from the rest configuration to the current configuration. This rotation is used to *warp* the vertex deformations back to the rest shape before internal elastic stresses are computed. This strain is then transformed back to the current configuration for time integration.

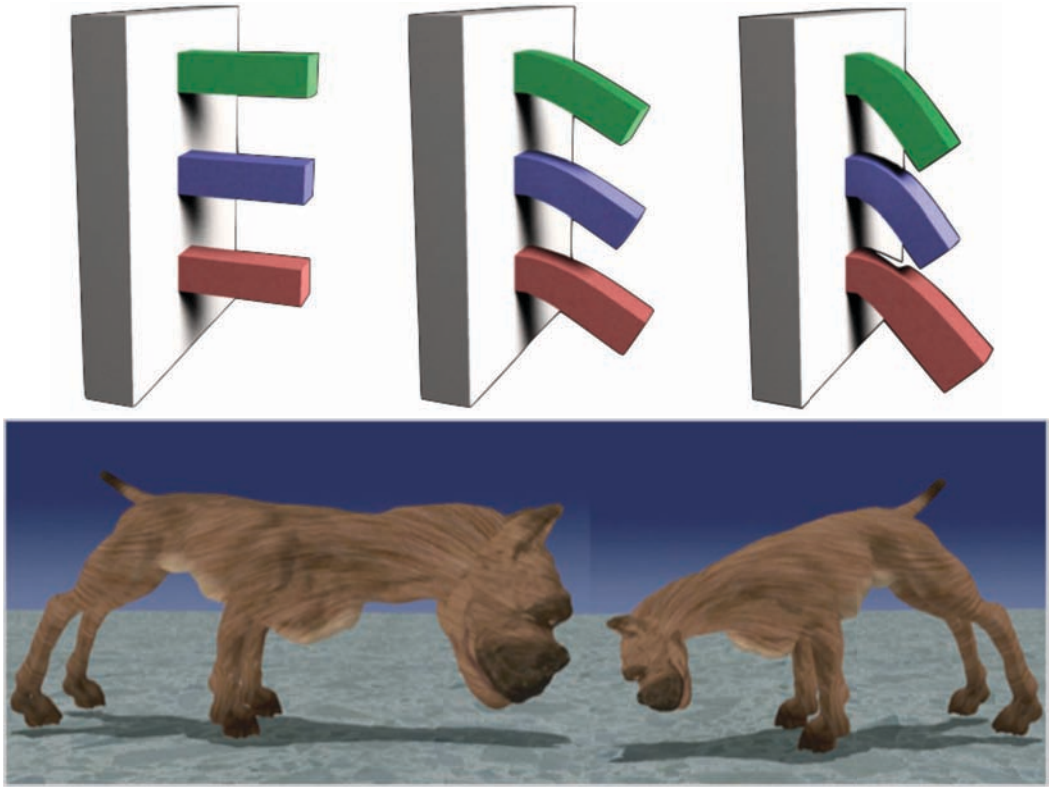


Figure 2.2: Stiffness Warping. Linearized elastic forces are only valid for small deformations (left). To solve these artifacts, Müller introduces *stiffness warping* [MDM+02] (right).

This yields stable, fast and visually pleasing results. In an earlier approach, they extracted the rotational part not per element but per node [MDM+02]. In this case, the global stiffness matrix does not need to be reassembled at each time step but ghost forces are introduced. Müller's approach can be categorized in a more general class of methods called co-rotational FEM schemes [Fel00, HS04]. However, most of these methods do not consider centripetal and Coriolis forces introduced by the moving reference frame.

Local Linearization The seminal work by Terzopoulos and Witkin [TW88] is another approach to account for the fact that linear strain models are not invariant to rotations. They explicitly track a single rigid body rotation for the entire deformable body, instead of estimating the rotation of each element. They propose a hybrid approach, where linear strain models are exploited for large deformations by decoupling

Figure 2.3: Local linearization. Capell *et al.* [CGC+02a] associate each region of the finite element mesh with the bone of a simple skeleton and then locally linearize the elastic forces in the frame of that bone.

the rigid body motion from the deformation field. Although geometrically less accurate than Müller's work, it accounts for inertial forces introduced by the moving and warped frame. Shabana [Sha89] proposes a similar approach for articulated characters. Another solution to this problem is proposed in [CGC+02a]: each region of the finite element mesh is associated with the bone of a simple skeleton (Figure 2.3) and then locally linearized. The regions are blended in each time step, leading to results which are visually indistinguishable from the non-linear solution, yet much faster.

2.1.3 Reduced Models

Unfortunately, employing linear elasticity models is usually not sufficient by itself to obtain real-time performance for highly tesselated solids. Several recent techniques have been proposed to reduce the number of the degrees of freedom (DoFs) in deformation simulations.

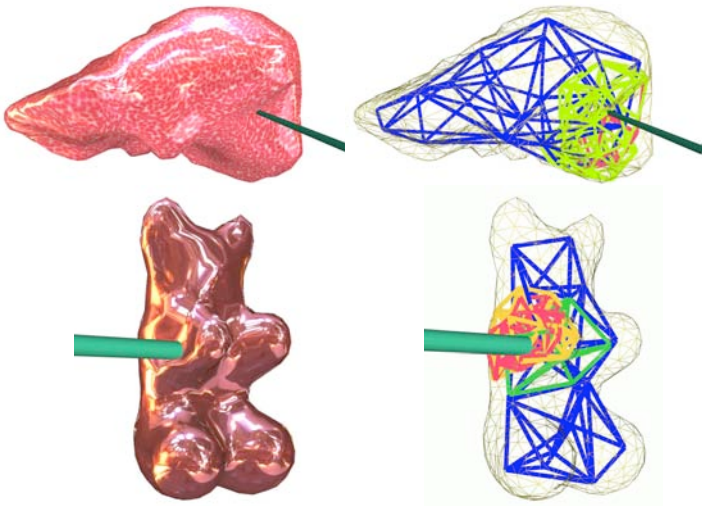


Figure 2.4: Multi-resolution methods. Debunne *et al.*partition the object in a non-nested multi-resolution hierarchy of tetrahedral meshes [DDCB01].

Multi-resolution methods For example, multi-resolution methods [DDCB01, GKS02, CGC+02b] focus computations with many DoFs at locations where high accuracy or high detail is required. Debunne *et al.* [DDCB01] partition the object in a non-nested multi-resolution hierarchy of tetrahedral meshes. The local resolution is determined by a quality condition that indicates where and when the resolution is too coarse. As the object moves and deforms, the sampling is refined to concentrate the computational load on the regions that deform the most. This is illustrated in Figure 2.4. Grinspun *et al.* [GKS02] and Capell *et al.* [CGC+02b] employ a subdivision scheme directly in the finite element discretization scheme. They build a hierarchical basis using volumetric subdivision, allowing the simulation to choose the appropriate subdivision level at runtime, adding detail where it is needed. Otaduy *et al.* [OGRG07] integrate multigrid algorithms and collision detection by identifying boundary conditions while inherently exploiting adaptivity.

Modal Reduction Reduced coordinate methods based on modal analysis [JP02, JF03, HSO03, BJ05, CK05] achieve rich deformations with a low computational cost by describing global deformations as the combination of a few DoFs. For example,

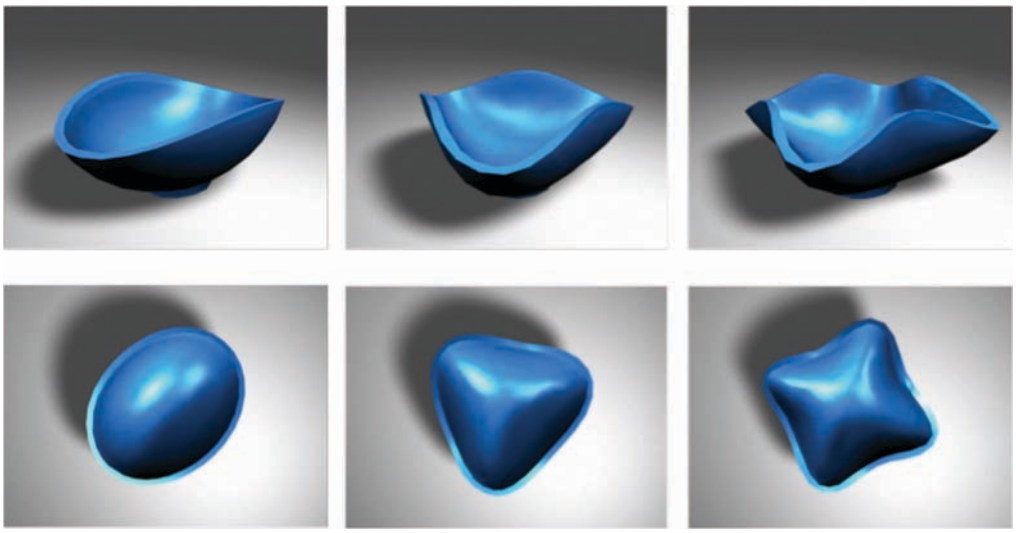


Figure 2.5: Modal Reduction. Reduced coordinate methods based on modal analysis deformations have a low computational cost by describing global deformations as the combination of a few DoFs. In this figure, the principal modes of deformation of a deformable shell are shown [HSO03].

Figure 2.5 shows the principal modes of deformation of a deformable shell. Barbič and James [BJ05] have recently shown how to exploit St. Venant-Kirchhoff models along with reduced coordinate methods, thus producing very fast, large rotation-invariant deformations. However, reduced coordinate methods based on modal analysis are not intended for generating deformations with local support, which often arise during contact. Existing multi-resolution and reduced coordinate methods implicitly assume that a small number of DoFs or a few global deformation bases are sufficient to describe meaningful and possibly very large deformations. Choi *et al.*'s *modal warping* method [CK05] extends stiffness warping to support modal reduction, but only supports moderate deformations of constrained objects that are attached to rigid supports. Free-floating objects are supported in the work by Hauser *et al.* [HSO03], and they also have basic support for constraints in their modal simulation framework.

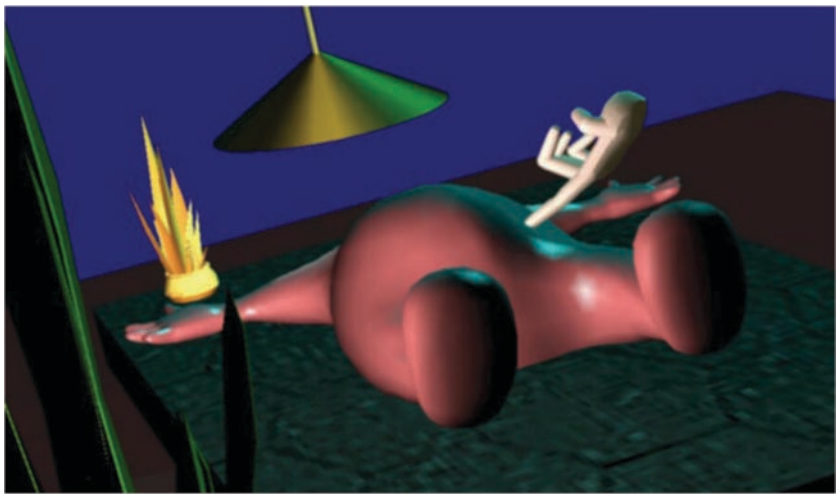


Figure 2.6: Surface oriented methods. Boundary Element Methods [JP99] have been proposed for obtaining elasto-static formulations that reduce the computations to surface nodes.

2.1.4 Surface Oriented Methods

Condensation [BNC96] and Boundary Element Methods (BEM) [JP99] have been proposed for obtaining elasto-static formulations that reduce the computations to surface nodes, while accounting for internal material properties. They exploit linear elasticity to pre-compute the inverse of a matrix that is dense with respect to the number of surface nodes. James and Pai [JP99] further optimized this approach by performing incremental updates in situations with contact coherence, resulting in an interactive framework, as illustrated in Figure 2.6. Unfortunately, these methods still suffer the disadvantages of linear strain metrics under rotational motion. Zhuang *et al.* [ZC99] propose the use of a *graded* mesh to reduce the complexity of the 3D problem by one order of magnitude asymptotically. The spatial tesselation of a graded mesh has a higher resolution near the surface of the mesh than on the inside. They suggest that if the size of the element increases proportionally to the distance to the surface, one will lose little accuracy with respect to static forces exerted on the surface.

Figure 2.7: Image-based techniques. The technique illustrated here by Sumner et al. [SOH99] animates surface deformations induced by contact, but does not handle global motion effects.

2.1.5 Layered Deformable Models

Layered deformable models [CHP89, TT93, Gas98, CGC+02a, CBC+05] overlay layers of deformable material on top of an articulated skeleton that drives the motion. Upon contact, these models typically produce only surface deformations and are often not designed to capture the two-way coupling of the global motion of the colliding objects. Novel rigid body models with compliance [SK03, PPG04], although designed to alleviate singularities in contact computation for rigid bodies, can be regarded as a specific type of layered deformable model. They are designed to capture the two-way coupling of the global motion of bodies in contact, but use simple deformation models, such as spring-damper networks [SK03] or Boussinesq's approximation [PPG04].

2.1.6 Regular Grids and Texture-based Approaches

Texture-based representations have been used for animating surface deformations. Stam [Sta03] has introduced a technique for simulating flows on the parametric domain of subdivision surfaces. The image-based techniques by Sumner et al. [SOH99] and Wrotek et

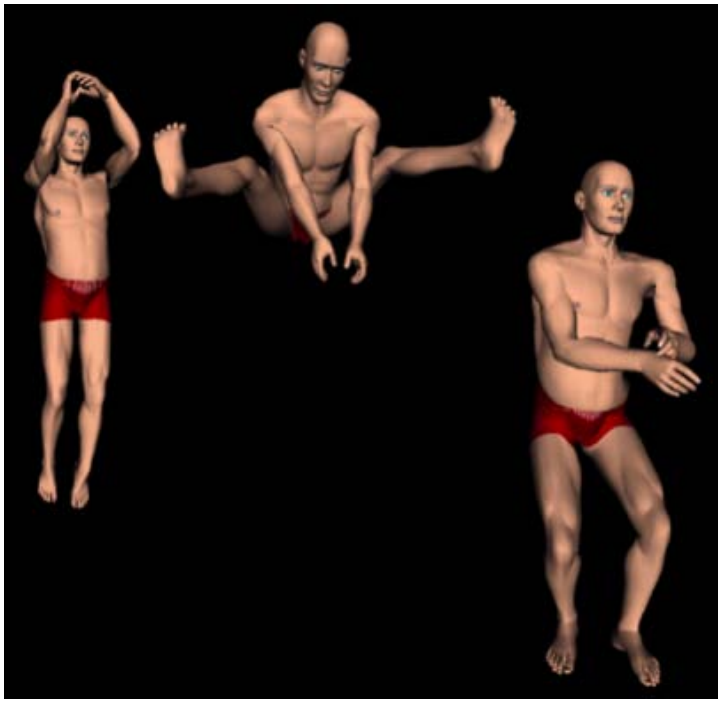


Figure 2.8: DyRT. The technique by James and Pai [JP02], animates deformations excited by global deformation modes using *Dynamic Response Textures*.

al. [WRM05] animate surface deformations induced by contact, but do not handle the effect of deformation forces on global motion of the objects (Fig. 2.7). The technique by James and Pai [JP02], on the other hand, animates deformations excited by global deformation modes (Fig. 2.8), but does not focus on contact-induced deformations. These methods unfortunately do not capture the coupling between global motion of free-floating objects and contact forces, and deformations in the way e.g. Terzopoulos' work does [TPBF87].

2.2 Character Animation and Simulation

Animation and simulation of deformable articulated characters is a problem that has been investigated using procedural, example-based, or physically-based approaches. The animation of a character's skeleton can be separate from the motion of its deformable surface. They can be modeled and animated by an artist or automated by physical or

physically-inspired simulation. In this overview, I have surveyed the body of work that is most related to the work in this dissertation.

2.2.1 Procedural Methods

A fundamental technique used in current character animation is to drive the deformation of the surface via an underlying skeleton. In the context of three-dimensional animation, this technique was introduced by Komatsu *et al.* [Kom88] and [MTLT88]. Given the animation of the skeleton, the deformation of the skin surface is computed by linear blending of bone transformations. This method is called *skeletal-subspace deformation* (SSD). This technique, also known as linear blend skinning, cannot capture complex deformations and typically has problems deforming skin near joints due to collapsing geometry (i.e. pinching), because the deformation is restricted to the subspace of the affine transformation of the joints. A few of these problems are illustrated in Figure 2.9. Different methods have been proposed to address these problems by using example-based deformations [LCF00], adding eigenbases of deformations in pose space [KJP02], inserting additional joints tuned from examples [MG03], employing blending of transformations instead of weights [KZ05], or the use of dual quaternions [KCŽC07], among others. Recent techniques have extended skinning to mesh deformations [JT05] or motion capture data without a predefined skeleton [PH06].

2.2.2 Example-based Methods

Another key technique in character animation is the representation of shape deformations using a set of meaningful parameters rather than directly manipulating control vertices. This idea was introduced by Parke in his work on facial animation [Par82]. He separates control parameters into two categories. Conformation parameters capture aspects of the face that vary from person to person, such as nose length. Expression parameters control deformations relating to the emotional state and activity of the face,

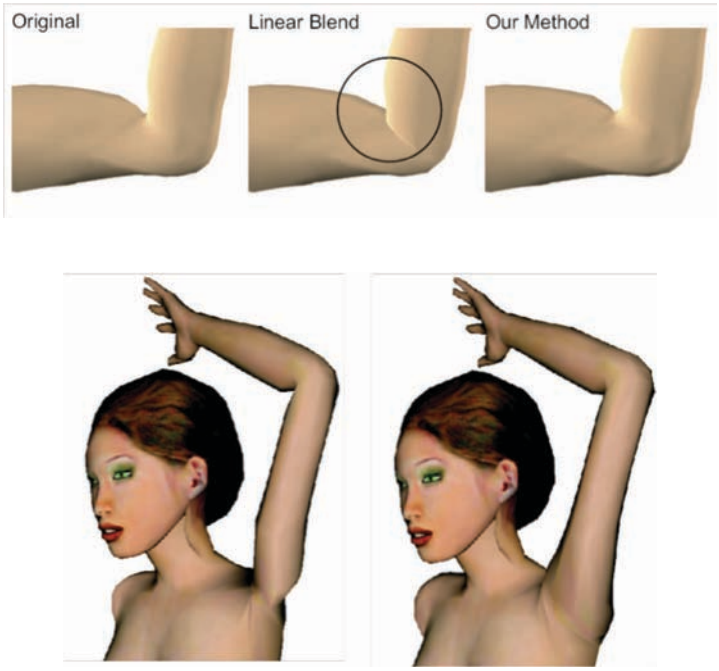


Figure 2.9: Skinning Problems. Linear blend skinning has problems deforming skin near joints due to collapsing geometry (i.e. pinching). Dual quaternions have been proposed to address this problem [KCŽC07].

such as smiling. The deformations associated with each parameter are built on a set of primitive deformation operations including procedural construction, interpolation, rotation, scaling, and positional offset.

Lewis *et al.* [LCF00], and Sloan *et al.* [SIC01] were the first to provide hybrid methods to improve the methods above in order to enable shape interpolation for skeleton-driven characters. By factoring out the nonlinear warping due to skeletal joints, they are able to interpolate character shapes associated with particular skeletal configurations. The surface of the character can be hand-modeled in any pose of the skeleton, and the given surfaces are then interpolated to provide a surface for any other pose. When dealing with large pose-spaces that have many example poses, PSD becomes memory inefficient due to the large database of surface displacements. PSD can be extended to support per-vertex weighted pose-space deformation (WPSD) [KM04, RLN06], largely reducing the number of required example poses. The EigenSkin method [KJP02] also

provides a way to reduce per-vertex displacement memory footprint by computing an error-optimal set of eigenbases for approximating the original deformation model and is optimized using graphics processors. Other recent methods [WSLG07, WPP07] learn example-based corrections on sparse points and assume that these corrections can be smoothly interpolated.

2.2.3 Physically-based Methods

Skeletal Dynamics In the area of *rigid* articulated physical simulation, several linear-time algorithms exist for simulating articulated skeletons without closed loops, either with articulated body inertias [Fea87] or with Lagrange multipliers [Bar96]. The Lagrange multipliers framework can be naturally extended to formulate contact constraints.

Recently, a few researchers have shown how to handle both unilateral and bilateral constraints. For example, Cline and Pai [CP03] emphasize handling rigid body contact constraints using post-stabilization, whereas Erleben [Erl04] combines joint constraints, joint limits, and joint motors with rigid contact constraints in a velocity-based linear complementarity formulation with shock propagation. Weinstein et al. [WTF06] propose an iterative solution for handling joint and contact constraints for rigid, articulated bodies in a single framework. Accurate contact handling for an articulated body with k bones and m contacts has $O(km)$ complexity [Bar96]. Obviously, this multiplicative complexity poses a problem for collision intensive scenes with complex characters. Physically-based simulation of an articulated skeleton can also be combined with motion capture data to generate plausible blending of motion capture segments [ZMCF05].

Skin Deformations Skeletal deformations can also be used to impose constraints on a control lattice for FEM simulation of dynamic deformations [CGC+02a, GW05]. Recently, Capell et al. [CBC+05] extended their framework to include rigging force fields, self-collision handling, and linearization of deformations in pose space.

Anatomical Modeling In the simulation of human characters, accurate anatomy-based representations (e.g., [DCKY02]) can be used for modeling material properties. These representations, however, are computationally expensive, and the goal of this thesis is to develop methods that can interactively capture surface deformation effects and global deformation in a plausible way with less focus on internal behavior.

2.3 Contact Handling

Contact handling in physical simulation usually consists of two steps, *collision detection* and *computation of contact response*. Often they are performed independently, with the result of collision detection being used as input to the contact response module. On the other hand, in reality they are tightly intertwined: collision detection has a strong influence on the continuity of contact response and hence on the stability of numerical integration.

2.3.1 Collision Detection

Collision detection is the first step in contact handling between two bodies. Collision detection has received much attention in robotics, computational geometry, and computer graphics. For more information on collision detection between rigid objects, please refer to surveys on the topic [LM04]. For a survey of recent techniques for deformable bodies, please refer to a more recent survey [TKH+05].

The existing work on collision detection has tackled many different types of models: 2-manifold polyhedral models, polygon soups, and curved surfaces are just a few examples. In this dissertation, the main interest lies in collision detection between two *deformable* objects. Such objects change at each frame, making the use of precomputed acceleration structures much harder. The simulation framework of this thesis takes a two-step approach, exploiting low-resolution 2-manifold rigid proxies in the first step,

and detecting interference of high-resolution deformable polygon soups in the second step. Therefore, it is interesting to look at the previous work that is related to each one of those two steps.

Proximity Queries Between Convex Polyhedra The property of convexity has been exploited in algorithms with sublinear cost for detecting interference or computing the closest distance between two polyhedra. Gilbert *et al.* [GJK88] exploit Minkowski sums in order to design a convex optimization algorithm (known as GJK) for computing the separation distance between convex polyhedra, with linear-time performance in practice. Lin and Canny [LC91, Lin93] designed an algorithm for computing separation distance by tracking the closest features between convex polyhedra. They exploit motion coherence and Voronoi marching to achieve an algorithm that runs in nearly constant time per frame. Later, hierarchical convex representations have been exploited along with temporal coherence in order to accelerate queries in dynamic scenes [GHZ99, EL00, Lin93].

Hierarchical Collision Detection Collision detection between general models achieves large speedups by using hierarchical culling or spatial partitioning techniques that restrict the primitive-level tests. Over the last decade, bounding volume hierarchies (BVH) have proved successful in the acceleration of collision detection for dynamic scenes of rigid bodies. For an extensive description and analysis of the use of BVHs for collision detection, please refer to Gottschalk's PhD dissertation [Got00]. The optimal choice of type of bounding volume depends on the application. Some of the common bounding volumes (BVs), sorted approximately according to increasing query time, are: spheres [Qui94, Hub95], axis-aligned bounding boxes (AABB)[BKSS90], oriented bounding boxes (OBB)[GLM96], k-discrete-orientation polytopes (k-DOP) [KHMS98], convex hulls[EL01], and swept sphere volumes (SSV)[LGLM00]. BVHs of rigid bodies can be computed as a preprocessing step, but deformable models require a bottom-up update

of the BVs after each deformation. Recently, James and Pai[JP04] have presented the BD-tree, a variant of the sphere-tree data structure[Qui94] that can be updated in a fast top-down manner if the deformations are described by a small number of parameters.

Surface interference for deformable polygon soups If compared to collision detection approaches for rigid bodies, there are various aspects that complicate the problem for deformable objects. First, in order to realistically simulate interactions between deformable objects, all contact points including those due to *self-collisions* have to be considered. Then, as mentioned before, efficient collision detection algorithms are accelerated by spatial data structures including bounding-volume hierarchies, distance fields, or alternative ways of spatial partitioning. Such object representations are commonly built in a *pre-processing* stage and perform very well for rigid objects. However, in the case of deforming objects these pre-processed data structures have to be updated frequently. Collision detection algorithms for deformable objects also have to consider that extra information such as penetration depth may be required for realistic contact response.

Image-space Collision Detection with Graphics Processors (GPUs) Due to the rapidly growing trend for parallel architectures [GRLM03] and its immense potential for high-performance general-purpose computation, many researchers have exploited the parallel processing power of GPUs for collision detection in the last few years. Rasterization hardware enables high performance of image-based collision detection algorithms. Hoff *et al.* [IZLM01] presented an algorithm for estimating penetration depth between deformable polygons using distance fields computed on graphics hardware. Others have formulated collision detection queries as visibility problems. Lombardo *et al.* [LCN99] intersected a complex object against a simpler one using the view frustum and clipping planes, and they detected intersecting triangles by exploiting OpenGL capabilities. More recently, Govindaraju's CULLIDE algorithm [GRLM03] performs series of visibil-

ity queries and achieves fast culling of non-intersecting primitives in N-body problems with nonrigid motion. Otaduy *et al.* [OJSL04] use low-resolution geometric representations (or *proxies*) with texture images that encode surface details. They exploit GPUs to compute directional penetration very efficiently, which enables interactive haptic texture rendering of complex models. More recently, Greß *et al.* [GGK06] use GPU-based collision detection for parameterized deformable surfaces. They are able to render and check collisions for deformable models consisting of several thousands of trimmed NURBS patches in real-time.

2.3.2 Contact Response

Contact response is typically applied in one of two ways: using penalty methods or enforcing constraints.

Penalty Methods The major advantage of penalty methods [WVS90, BW98, HFS03] is their ease of implementation, but they rely on the existence of interpenetrations to produce collision response. Interpenetrations may be alleviated by using stiff penalty forces along with implicit integration methods, but this approach results in a coupling of the motion equations of colliding bodies, and penalty methods lose their original simplicity.

Constraint-based Methods Based on the Signorini problem [KO88], the contact response between deformable bodies can be formulated as a Linear Complementarity Problem (LCP) on contact forces and displacements [SK03, PPG04, DAK04]. Duriez et al. [DAK04] formulated the LCP along with the time-discretization of FEM based dynamic deformation equations. LCP-based solution to contact problems has long been applied for rigid bodies, but with deformable bodies the dimension of the LCP may grow by orders of magnitude when contact areas are large. Pauly et al. [PPG04] suggest

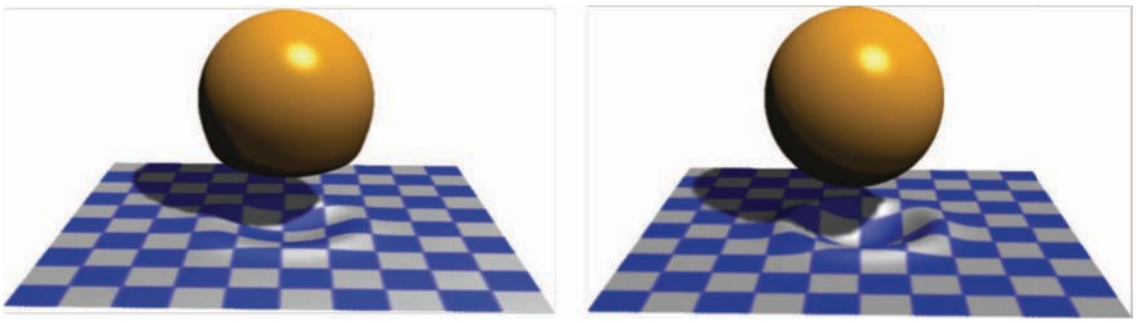


Figure 2.10: Constraint-based Contact Handling. Using only a small subset of the contact points in a constraint-based method accelerates contact handling, but limits the resolution of deformations and produces smoothened deformations (see Fig. 2.10).

using only a small subset of the contact points in the LCP, which limits the resolution of deformations and produces smoothened deformations (see Fig. 2.10).

With deformable bodies, contact lasts a small duration, and the classic inequality non-penetration constraints may be substituted by equality constraints on the colliding particles without affecting the body's momentum much [BW92, BFA02]. A large family of methods formulate contact constraints and apply contact impulses on each colliding object independently [ZC99, BFA02, CW05, KEP05]. Pre-impact velocities are computed by performing a collision-free integration step of the motion equations. Then, colliding particles are detected, and impulses are applied to compute post-impact velocities that satisfy the contact constraints. Finally, positions are corrected as well. Previous methods, however, apply impulses explicitly, and require small time steps to avoid the inversion of the deformable mesh, as they rely on accurate accumulation of elastic energy during the compression phase [ITF04, PPG04]. In order to increase the fidelity and *responsiveness* of collision detection, others have used implicit time integration of the constrained motion equations. Cirak *et al.* [CW05] formulated the deformable constraint problem using Lagrange multipliers, into an algorithm that is unfortunately computationally expensive and poorly parallelizable.

Contact Response for Procedural Characters For procedural character animation methods such as linear blend skinning (SSD) [MTLT88], the surface deformation is fairly restricted because the skeletal pose fully defines the surface deformation. Such methods based on the skeletal configuration and possibly other parameters [LCF00] cannot capture the reaction of the character to collisions, as illustrated in Fig. 1.1.

Contact Response for Layered Models Terzopoulos and Witkin [TW88] suggested an explicit integration of rigid motion and contact forces. Unfortunately, this approach is only conditionally stable for appropriate time steps and it requires small simulation time steps for the correct propagation of strain due to collision compression [PPG04]. Thus, as explained in Sections 1.1.3 and 3.4. I propose implicit integration of contact forces to achieve stable and responsive contact handling (see also Figure 1.7).

2.4 Control of Deformations

In addition to strictly forward simulation of skin dynamics, there has been a fair body of work related to control of skin deformations. That work is motivated by the importance of artistic freedom and intuitive control in a animation pipeline. In this section, I will give an overview of significant work related to *control* of surface deformation of both kinematic and dynamic characters. Similar to animation and forward simulation of skin dynamics in 2.2, I will categorize the work related to surface deformation control into data-driven, kinematic, combined and physically-based methods.

2.4.1 Data-driven methods

Purely data-driven methods are an attractive choice for control purposes, as the input shapes provide guide examples of desired deformations. In its most essential form, one simply interpolates between character poses in a large database [Mae06], providing

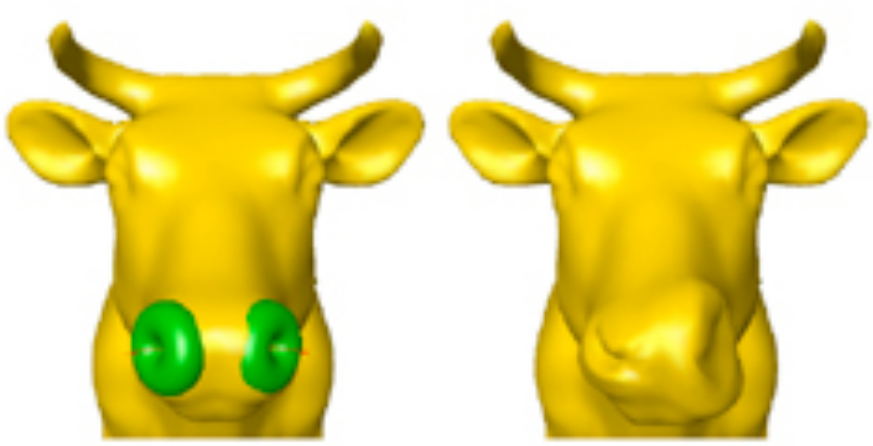


Figure 2.11: Physically Based Rigging. Capell *et al.* [CBC+05] use force field templates to control facial expressions. Here, two torus templates produce dilation and contraction of the nostrils.

ample control of skin deformation to animators. However, many poses are required in the database to achieve good results. Purely data-driven methods lack a kinematic model, making them of limited use for animation and dynamic simulation.

2.4.2 Kinematic methods

Purely kinematic approaches such as *skeletal-subspace deformation* (SSD) [MTLT88] model the deformation of the skin surface by linear blending of the animated bone transformations. In this technique, the surface deformation is restricted to the subspace of the affine transformation of the joints. Thus SSD is limited in the degree to which the animator can influence the deformation. Unlike shape interpolation and other data-driven methods, SSD does not permit direct sculpting or control. Instead, artists have to tweak vertex weights, giving SSD algorithms the reputation of being tedious to control.

2.4.3 Combined methods

The first work to add control to a kinematic approach is that of *pose-space deformations* [LCF00]. PSD is a hybrid method that combines SSD with shape morphing and

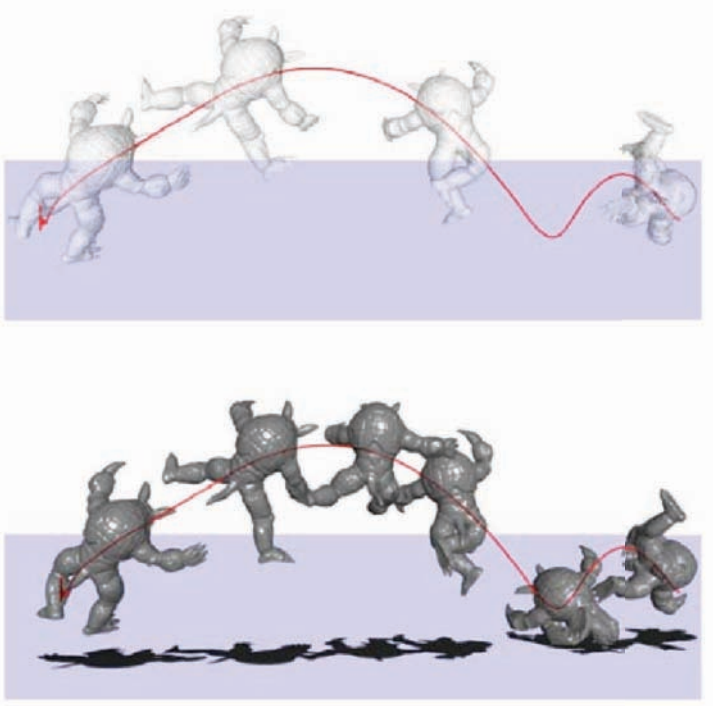


Figure 2.12: Directable Animation of Deformable Objects. Kondo *et al.* [KKA05] retarget elastic deformations with shape keyframes (above), which are used to compute the control result (below). Unfortunately, this technique is restricted to a given input animation.

employs scattered data interpolation to compute non-linear skin corrections in pose-space, resulting in a kinematic model that also has artist-sculpted poses. Pose space deformation and related example-based methods allow direct sculpting of geometric morph targets, but are purely kinematic approaches to (quasi-)static deformation, without reference to underlying forces or mass.

2.4.4 Physically-based methods

Finally, physically based methods can be used to increase the realism of a controlled animation. Such realism can be achieved by exploiting bio-mechanical models of skin tissue and musculature[ZCCD04, SNF05, SKP08]. To increase performance, some researchers have used simplified (quasi-)linear elastic models that cannot capture complex non-linear behavior such as muscle bulging. Physically based methods can only provide

control through the influence of forces. While methods that control global deformation modes have been around for a while [WW90], providing control of sculpted deformations for simulation of deformable models has only recently caught attention in graphics research. A method for physically based rigging was proposed by [CBC+05], using pose-dependent forces to guide the shape of the character. This approach does not support pose-dependent elastic properties and its performance is highly dependent on the resolution of the sculpted deformations. Given an input animation, shape keyframes can be used to retarget the elastic deformations [KKA05, AOW+08] or to enhance the surface deformations with physically simulated detail using subspace constraints [BMWG07]. The former provides good control of shapes but is restricted to a given input animation, while the latter achieves rich secondary surface detail but does not provide direct manipulation of the surface.

Chapter 3

Soft Body Simulation with Dynamic Deformation Textures

In this chapter, I present a novel and fast simulation framework for a specific class of deformable bodies in contact. The deformable bodies discussed in this chapter are assumed to have a rigid core enclosed in a layer of deformable skin. The simulation of the deformation in the skin layer is based on the projection of the three-dimensional deformation field onto a lower-dimensional space. The many DoFs arising from large contact regions and high-resolution geometry can be handled more efficiently handled in two-dimensional parametric atlases called *dynamic deformation textures*. In Chapter 4, this approach is generalized to support characters with an articulated core (i.e., a skeleton). The use of layered representations have been proposed before for simulating skeletal deformations [CHP89, Gas98, CBC+05], as many real-world solids retain their core structures under large deformations. In this chapter and in Chapter 4, I show that my layered model enables stable simulation of very appealing soft body effects for a wide range of objects. Examples include animated characters, human bodies, furniture, toys, footballs, tires, etc. I derive an implicit yet highly parallelizable solution to dynamic deformations using linear elasticity theory (with separation of rigid motion), continuum Lagrangian mechanics, FEM discretization, and constraint-based contact response with Lagrange multipliers.

Dynamic deformation textures are exploited at all levels of the simulation framework described in this chapter:

- Proximity queries of deformable bodies are computed in a two-stage algorithm directly on dynamic deformation textures, resulting in output-sensitive collision detection that is independent of the combinatorial complexity of the deforming meshes.

- Skin deformations are formulated using pose-space elasticity dynamics in the parametric domain of dynamic deformation textures. A highly parallelized and efficient implementation on the GPU is achieved by decoupling the motion equations.

- Contact response is also computed directly on the dynamic deformation textures. I present a robust, parallelizable formulation for computing constraint forces using implicit methods that exploits the structure of the motion equations to achieve highly stable simulation, while taking large time steps with inhomogeneous materials.

- Finally, dynamic deformation textures can be used directly for real-time shading and easily be implemented using SIMD architecture on commodity hardware.

3.1 Dynamic Deformation Textures

In this section, I present a layered representation of deformable objects, specifically tuned for objects that are amenable to low-distortion mapping or *unwrapping* to a 2-dimensional domain. An efficient extension to articulated characters is non-trivial, but a solution to that problem is presented in Chapter 4.

In the next few sections, I discuss the 2-dimensional parameterization of the deformation field, and I show how this can be discretized and stored in a texture atlas. I

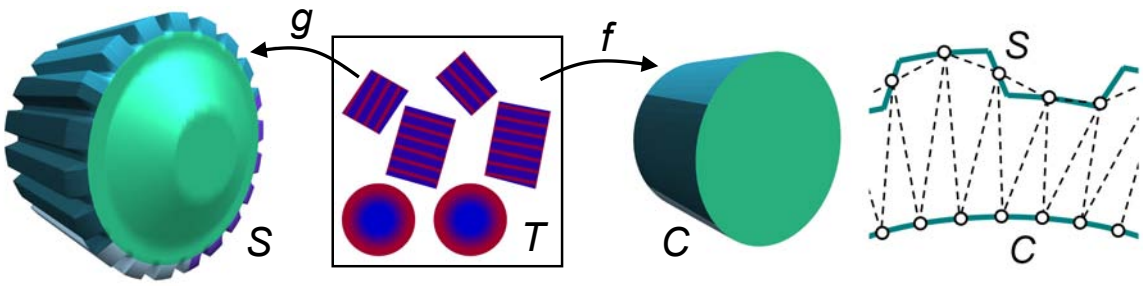


Figure 3.1: Deformable Object Representation. Deformable surface S (52K triangles) and core C (252 triangles) of a gear, showing the patch partitioning of its parameterization. The common color coding reflects the mapping $g \circ f^{-1} : C \quad S$. The dynamic deformation texture T (256×256) stores the displacement field values on the surface. On the right, $2D$ schematic figure showing a slice of the tetrahedral meshing of the deformable layer. The gear contains 28K simulation nodes on the surface and 161K tetrahedra, allowing the simulation of highly detailed deformations.

also introduce the generalized set of coordinates, which are partitioned into a rigid and a deformation coordinate set.

3.1.1 Parameterization of Layered Deformable Objects

Each deformable object is modeled as a *core* covered by a layer of (possibly heterogeneous) deformable material. For simplicity, this chapter describes the simulation framework of layered deformable objects assuming that the core consists of one single rigid body of low polygonal complexity. In Section 4.1, this approach is extended for articulated characters. It will be demonstrated in the remainder of this chapter that the layered representation enables modeling of many types of deformations in a elegant and unified manner:

1. Both large and small scale deformations over large regions of the object's surface.

2. Global deformations of skeletal nature.

3. Two-way dynamic coupling between the global motion of the object and the surface deformations produced during contact.

A body frame is attached to the core, with rotation $\mathbf{R}$ and position of the center of mass $\mathbf{c}$. The world position $\mathbf{x}$ of a point in the object can be expressed in terms of its body-frame position $\mathbf{u}$ as $\mathbf{x} = \mathbf{c} + \mathbf{R}\mathbf{u}$. In the deformable layer, $\mathbf{u}$ can be decomposed into a constant undeformed component $\mathbf{u}_o$ and a displacement $\mathbf{u}_s$, hence $\mathbf{u} = \mathbf{u}_o + \mathbf{u}_s$.

I proceed by expressing 3-dimensional deformations in a 2-dimensional parametric domain. For that purpose, I parameterize the surfaces of the core and the deformable object. A surface patch $C \subset \mathbb{R}^3$ of the core can be defined by a mapping f from a domain $T \subset \mathbb{R}^2$, $f : T \quad C$. I enforce a one-to-one correspondence between points on the surface of the deformable object and points on the surface of the core. Then a corresponding surface patch $S \subset \mathbb{R}^3$ of the deformable object can be defined by a mapping $g : T \quad S$, and the correspondence by $g \circ f^{-1}$. Based on the mapping g, the body-frame position $\mathbf{u}$ of the surface of the object and the displacement field $\mathbf{u}_s$ can be expressed as 2-dimensional functions $\mathbf{u}(s,t)$ and $\mathbf{u}_s(s,t)$.

The one-to-one correspondence can be achieved by appropriately modeling the core. One option is to parameterize the surface of the object in its rest position and decimate the surface while preserving the parameterization [COM98, SSGH01]. Another option is to parameterize the core and model the surface of the object by successive subdivision and addition of geometric detail [ZSS97]. The surfaces of the core and of the deformable object may be partitioned into multiple patches, and each patch parameterized independently.

3.1.2 Discretization and Generalized Coordinates

A regular sampling of the planar domain T can be regarded as a texture atlas, which I refer to as *dynamic deformation texture*. Moreover, the grid points (s,t) are referred to as texels. Each texel $(s,t) \in T$ maps to two corresponding points $f(s,t)$ and $g(s,t)$ on the surfaces of the core and the deformable object. The regular sampling of T and the correspondence of surface points define implicitly a meshing with one layer of tetrahedral

elements, as shown in Figure 3.1. By applying classical approximation methods such as FEM, the continuous displacement field $\mathbf{u}_s$ on the deformable layer can be approximated from the values at a finite set of nodes. Since $\mathbf{u}_s = 0$ at points on the core, $\mathbf{u}_s$ can be approximated entirely from the values $\mathbf{u}_s(s,t)$ at surface nodes. Effectively, each texel $(s,t) \in T$ maps to a simulation node $g(s,t)$ in the FEM discretization. Simulation variables defined per-node, such as velocities, forces, mass and stiffness values, etc. can also be stored in texture atlases. Note that the implicitly defined texture-based meshing is not consistent at patch boundaries, which require special yet simple treatment as discussed in Section 3.5.

The displacements of the surface nodes are packed in a vector of *elastic coordinates* $\mathbf{q}_s \in \mathbb{R}^n$. Together with the *core coordinates* $\mathbf{q}_c = \begin{matrix} \mathbf{c} \\ \theta \end{matrix} \in \mathbb{R}^7$, they form the generalized coordinates

$$\mathbf{q} = \begin{matrix} \mathbf{q}_c \\ \mathbf{q}_s \end{matrix} \tag{3.1}$$

This vector of generalized coordinates describes the state of a layered deformable object. I choose quaternions to represent the orientation θ. Given a (position-dependent) shape matrix $\mathbf{S}$, the displacement of a point in the deformable layer can be expressed in compact matrix form as $\mathbf{u}_s = \mathbf{S}\mathbf{q}_s$. Then the world-frame position of a material point can be written as $\mathbf{x} = \mathbf{c} + \mathbf{R}(\mathbf{u}_o + \mathbf{S}\mathbf{q}_s)$. With linear basis functions, $\mathbf{S}$ is linear in the barycentric coordinates for each mesh element [BNC96].

Given , the angular velocity of the core expressed in body-frame coordinates, I define a velocity state vector

$$\mathbf{v} = \begin{matrix} \mathbf{v}_c \\ \mathbf{v}_s \end{matrix}, \qquad \mathbf{v}_c = \begin{matrix} \mathbf{c} \end{matrix}, \qquad \mathbf{v}_s = \mathbf{q}_s. \tag{3.2}$$

As shown in Appendix A.1, the velocity state vector and the generalized coordinates are

related by $\mathbf{v} = \mathbf{P}\mathbf{q}$ and $\mathbf{q} = \mathbf{P}^{+}\mathbf{v}$, where $\mathbf{P}$ and $\mathbf{P}^{+}$ are matrices that encapsulate the relation between and the derivative of a quaternion.

3.2 Layered Dynamic Deformations

In this section, I first formulate the motion equations of single-core layered deformable objects based on Lagrangian mechanics, linear elasticity theory, and linear FEM discretization. For an extension to multi-core, articulated characters, please refer to Section 4.2.1. The motion equations are then discretized in time using implicit integration. I exploit the representation based on dynamic deformation textures and achieve an efficient, parallelizable set of equations, by separating the update of core and elastic velocities, without sacrificing responsive two-way coupling between the core and the deformable layer.

3.2.1 Equations of Motion

The motion equations of a deformable body can be derived from Lagrangian continuum mechanics [GPS02, Sha89]. Using linear elasticity theory and linear FEM, and by formulating the displacement field in the floating frame of reference, elastic forces are linear w.r.t. the elastic coordinates $\mathbf{q}_s$ and invariant under rigid motion of the core [Sha89, TW88]. I denote mass, damping, and stiffness matrices as $\mathbf{M}$, $\mathbf{D}$, and $\mathbf{K}$, generalized external forces as $\mathbf{Q}$, and a quadratic velocity vector that captures the inertial effects of centripetal and Coriolis forces as $\mathbf{Q}_v$. I obtain the ordinary differential motion equations:

$$\begin{aligned} \mathbf{M}\mathbf{v} &= \mathbf{Q} + \mathbf{Q}_v - \mathbf{K}\mathbf{q} - \mathbf{D}\mathbf{v} = \mathbf{F}, \\ \mathbf{q} &= \mathbf{P}^{+}\mathbf{v}. \end{aligned} \tag{3.3}$$

Figure 3.2: Simulation of Heterogeneous Materials. Efficient decoupled implicit integration enables fast simulation of a heterogeneous cylinder. Notice the large indentations and the resulting large contact area when the soft side collides with the ridged plane.

(A detailed description of the derivation can be found in [Sha89] and is also summarized in appendix B.5.) The mass and stiffness matrices present the following structure:

$$\mathbf{M} = \begin{matrix} \mathbf{M}_c & \mathbf{M}_{ce} \\ \mathbf{M}_{ce}^T & \mathbf{M}_e \end{matrix} \quad \text{and} \quad \mathbf{K} = \begin{matrix} 0 & 0 \\ 0 & \mathbf{K}_s \end{matrix} . \tag{3.4}$$

$\mathbf{M}_c \in \mathbb{R}^{6\times 6}$ and $\mathbf{M}_{ce} \in \mathbb{R}^{6\times n}$ are dense, and I approximate $\mathbf{M}_e \in \mathbb{R}^{n\times n}$ with a diagonal form by applying mass lumping. Raleigh damping $\mathbf{D}$ is used as follows:

$$\mathbf{D} = \begin{matrix} 0 & 0 \\ 0 & \mathbf{D}_s \end{matrix} \quad \text{with} \quad \mathbf{D}_s = \alpha\mathbf{M}_s + \beta\mathbf{K}_s, \tag{3.5}$$

such that the damping matrix $\mathbf{D}$ is a diagonal matrix acting only on the elastic coordinates. As shown by the structure of $\mathbf{K}$, the elastic forces depend only on the elastic coordinates $\mathbf{q}_s$. However, elastic forces affect the core coordinates as well, by inertial coupling through $\mathbf{M}_{ce}$. The submatrix $\mathbf{K}_s \in \mathbb{R}^{n\times n}$ is constant and sparse, with at most 21 non-zero terms per row, due to the regular texture-based meshing.

3.2.2 Efficient Decoupled Implicit Integration

I have opted for an implicit backward Euler discretization of the motion equations, enabling simulation of heterogeneous materials without letting the time step be governed by stiff regions. This approach enables simulation of very stiff materials and even heterogeneous materials, with large time steps, as shown in Fig. 3.2. Backward Euler discretization yields a nonlinear set of equations, which can be linearized using a first-order approximation of the force $\mathbf{F}(t + \Delta t)$ w.r.t. the state vectors $\mathbf{v}$ and $\mathbf{q}$. After algebraic manipulation, and assuming a constant mass matrix $\mathbf{M}$ during each time step, the equation for the velocity update $\Delta\mathbf{v}$ is:

$$\left(\mathbf{M} - \Delta t\frac{\mathbf{F}}{\mathbf{v}} - \Delta t^2\frac{\mathbf{F}}{\mathbf{q}}\mathbf{P}^+\right)\Delta\mathbf{v} = \Delta t\mathbf{F}(t) + \Delta t^2\frac{\mathbf{F}}{\mathbf{q}}\mathbf{P}^+\mathbf{v}(t). \tag{3.6}$$

I can define the *implicit mass matrix* $\mathbf{M}$ and *implicit force vector* $\mathbf{F}$ by gathering terms in (3.6):

$$\mathbf{M}\Delta\mathbf{v} = \Delta t\mathbf{F}(t). \tag{3.7}$$

Due to the linearity of (3.7), a collision-free velocity update can be computed first, and then the effect of contact forces (Section 3.4) is added to produce the constrained velocity update.

Unfortunately, the implicit mass matrix $\mathbf{M}$ does not lend to an efficient solution of the linear system (3.7). Instead, I propose an efficient solution that exploits the block structure of $\mathbf{M}$ and $\mathbf{F}$ in (3.4). I derive the following equations to decouple the update of core velocities $\mathbf{v}_c$ and elastic velocities $\mathbf{v}_s$:

$$\Delta\mathbf{v}_c = \mathbf{M}_{\text{cond}}^{-1}(\Delta t\mathbf{F}_c - \Delta t\mathbf{M}_{ce}\mathbf{M}_s^{-1}\mathbf{F}_s), \tag{3.8}$$

$$\Delta\mathbf{v}_s = \Delta t\mathbf{M}_s^{-1}\mathbf{F}_s - \mathbf{M}_s^{-1}\mathbf{M}_{ec}\Delta\mathbf{v}_c, \tag{3.9}$$

in which the condensed matrix $\mathbf{M}_{\text{cond}} = \mathbf{M}_c - \mathbf{M}_{ce}\mathbf{M}_s^{-1}\mathbf{M}_{ec} \in \mathbb{R}^{6\times 6}$. This differs from

other surface-oriented approaches [BNC96], where the size of the condensed matrix is governed by the number of surface nodes. The advantage of the decoupling lies in the structure of the systems to be solved. First, two linear systems are solved:

$$\mathbf{M}_s\mathbf{y} = \mathbf{F}_s \tag{3.10}$$

$$\mathbf{M}_s\mathbf{Y} = \mathbf{M}_{ec}. \tag{3.11}$$

Then $\mathbf{v}_c$ is updated by solving the condensed system (3.8). Finally, the elastic velocities $\mathbf{v}_s$ can be solved through (3.9) in a highly parallel manner.

The two $n \times n$ linear systems to be solved imply the matrix $\mathbf{M}_e$ which, omitting external forces, can be written as:

$$\mathbf{M}_e = \mathbf{M}_s - \Delta t\frac{\mathbf{Q}_{ve}}{\mathbf{v}_s} - \Delta t^2\frac{\mathbf{Q}_{ve}}{\mathbf{q}_s} + \Delta t\mathbf{D}_s + \Delta t^2\mathbf{K}_s. \tag{3.12}$$

I approximate the Jacobians of the quadratic velocity vector $\mathbf{Q}_{ve}$ by their diagonal forms. In this way, the matrix $\mathbf{M}_s$ is sparse, symmetric and positive definite; thus it can be solved efficiently using iterative methods. Moreover, as discussed in Section 3.5, the regularity of the matrix $\mathbf{K}_s$, due to my texture-based representation, enables a highly parallelizable implementation of the iterative solvers.

3.3 Texture-Based Collision Detection

Collision detection is the first step in resolving the dynamic behavior of soft objects in contact. In this section, I present an algorithm for collision detection that exploits dynamic deformation textures, in which contact constraints are detected in image space and then mapped to the texture-based simulation domain.

I propose to perform collision detection between two deformable objects A and B in two steps:

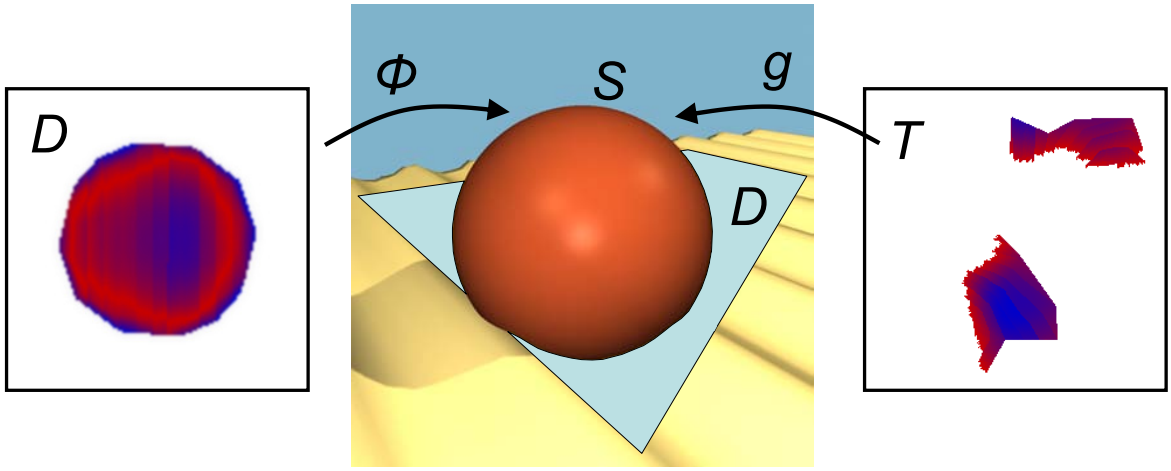


Figure 3.3: Texture-Based Collision Detection Process. Center: A sphere S collides with a textured terrain. Left: Contact plane D for texture-based collision detection, and mapping $\phi : D \quad S$. The contact plane shows the penetration depth. Right: Dynamic deformation texture T, and mapping $g : T \quad S$. The penetration depth is projected from D to T, and is available for collision response.

1. Identify contact regions with object-space techniques using low-resolution models of the objects.

2. Compute contact information in the contact regions using image-space techniques and high-resolution displacement fields.

A similar approach has been exploited for estimating the penetration depth value between rigid, high-resolution objects [OJSL04], whereas in my method, I perform collision handling of deformable objects and compute contact information for many colliding surface points.

In the object-space collision detection step, I identify patches of the core surfaces closer than a distance tolerance that bounds the high-resolution deformable surfaces. I employ existing acceleration methods based on convex hull hierarchies [EL01]. Given a contact region between core surface patches $C_A \subset \mathbb{R}^3$ and $C_B \subset \mathbb{R}^3$, I identify a contact plane $D \subset \mathbb{R}^2$. This is the plane passing between the contact points and oriented according to the contact normal. By orthonormal projection of C_A (and similarly for C_B) onto D, I define a mapping $h_A : D \quad C_A$. Due to the one-to-one correspondence

between patches on the surface core and patches on the deformable surface, a contact between core surface patches C_A and C_B defines a potential contact between corresponding deformable surface patches S_A and S_B.

Given a patch S on a high-resolution deformable surface, and the mappings g and f defined in Section 3.1.1, $\phi = g \circ f^{-1} \circ h$ defines a mapping $\phi : D \quad S$ from the contact plane D to the patch S, through the core patch C and the texture atlas T, as shown in Figure 3.3. Similarly to the sampling of the texture atlas T, the contact plane D is sampled in a regular grid. Then, each texel $(u, v) \in D$ maps to a point $\phi(u, v)$ on the high-resolution patch S.

For each texel $(u, v) \in D$, I perform high-resolution collision detection by testing the distance between points $\phi_A(u, v) \in S_A$ and $\phi_B(u, v) \in S_B$ along the normal direction of D. If the points are penetrating, I identify a contact constraint and compute the contact normal $\mathbf{n}$ as the average surface normal. I also approximate the penetration depth as $d = \mathbf{n}^T(\phi_B(u, v) - \phi_A(u, v))$ for applying constraint correction. This approximation is affected by texture distortion, but I have not found noticeable errors in my examples or benchmarks.

Contact constraint information can be transferred to a texture atlas T via the mapping $f^{-1} \circ h$ and made readily available for the computation of collision response at the simulation nodes, as shown in Figure 3.3. For accuracy of collision detection, it is convenient to sample the contact plane D at a higher density than the texture atlas T. As a result, multiple colliding points $(u, v) \in D$ may map to the same simulation node $(s, t) \in T$. In such cases, I keep only the constraint information from the colliding point with the largest penetration depth value.

3.4 Contact Resolution

After collision detection, the computation of dynamic response of colliding bodies is continued by formulating velocity constraints in the generalized coordinate setting using Lagrange multipliers. For enhanced two-way dynamic coupling between core and deformable layer under collisions, I propose a solution of collision response based on the implicit integration of constraint forces, and I present an efficient numerical solution.

Colliding surface nodes are prevented from penetrating other objects by the application of contact constraint forces. I first describe the handling of fixed, frictionless constraints, and then I extend the algorithm to moving constraints with friction. I define pre-impact velocities $\mathbf{v}^-$ computed by solving (3.7), post-impact velocities $\mathbf{v}^+$ and collision impulse $\delta\mathbf{v} = \mathbf{v}^+ - \mathbf{v}^-$. The contact constraints are expressed in the world-frame velocities of the colliding nodes, and must be transformed to the generalized coordinate setting by the kinematic relationship in Equation (A.5) in Appendix A.1. A planar constraint $\mathbf{n}$ acting at a node i produces an elastic collision impulse with coefficient of restitution ϵ governed by:

$$\mathbf{n}^T \left(\mathbf{x}_i^- + \frac{\delta\mathbf{x}_i}{1+\epsilon} \right) = \mathbf{j} \left(\mathbf{v}^- + \frac{\delta\mathbf{v}}{1+\epsilon} \right) = 0. \tag{3.13}$$

The generalized constraint normal is represented by the vector

$$\mathbf{j} = \mathbf{n}^T \mathbf{L}_i = \begin{bmatrix} \mathbf{n}^T & -\mathbf{n}^T \mathbf{R}\mathbf{u}_i & \mathbf{n}^T \mathbf{R}\mathbf{S}_i \end{bmatrix}, \tag{3.14}$$

where $\mathbf{S}_i$ indicates the position-dependent matrix $\mathbf{S}$ evaluated at node i. Note that $\mathbf{S}_i \mathbf{v}_s$ selects the i^{th} block component from $\mathbf{v}_s$. The velocity constraints can be jointly formulated with a generalized constraint matrix $\mathbf{J} \in \mathbb{R}^{m \times (6+n)}$, where m is the number of colliding surface nodes:

$$\mathbf{J} \left(\mathbf{v}^- + \frac{\delta\mathbf{v}}{1+\epsilon} \right) = 0. \tag{3.15}$$

In order to compute the collision impulse, a constraint force vector $\mathbf{J}^T\lambda$ is added to the external forces $\mathbf{Q}$ in (4.13). Here, λ is a vector of Lagrange multipliers. Typically, the collision impulse is solved by explicitly integrating the constraint forces, which is equivalent to applying an instantaneous change of momentum to the surface nodes at the end of each time step. Unfortunately, this approach requires small simulation time steps for the correct propagation of pressure waves induced by collision response [PPG04]. With explicit integration and large time steps, the elastic deformation forces are unable to counteract the momentum of the core upon collision, and the core may penetrate the constraints. Figure 1.7 illustrates this problem.

I propose the computation of the collision impulse through implicit integration which, as shown in Figure 1.7, produces a robust and responsive reaction of the core with large time steps. Due to linearity of (3.7) w.r.t. the vector of forces $\mathbf{F}$, one can compute the collision impulse separately by solving:

$$\mathbf{M}\delta\mathbf{v} = \Delta t\mathbf{J}^T\lambda. \tag{3.16}$$

From (3.15) and (3.16), the following equation is obtained:

$$\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\lambda = -\frac{1+\epsilon}{\Delta t}\mathbf{J}\mathbf{v}^-. \tag{3.17}$$

After solving this equation for λ, the constraint forces $\mathbf{J}^T\lambda$ can be computed. The efficient decoupled implicit velocity update described in Section 3.2.2 is then performed to compute the post-impact velocities $\mathbf{v}^+$.

3.4.1 Efficient Decoupled Contact Resolution

The matrix $\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T$ is dense, and the computation of λ through (3.17) is computationally expensive. Instead, I propose to decouple (3.17) by exploiting the structure

of $\mathbf{J} = \begin{bmatrix} \mathbf{J}_c & \mathbf{J}_s \end{bmatrix}$, which can easily be derived from the individual node velocity constraints (3.13). $\mathbf{J}_c \in \mathbb{R}^{m\times 6}$ is dense, and $\mathbf{J}_s \in \mathbb{R}^{m\times n}$ presents one non-zero 1×3 block per row. Equation (3.17) can be rewritten as:

$$\left(\mathbf{J}_s\mathbf{M}_s^{-1}\mathbf{J}_s^T + \mathbf{U}\mathbf{M}_{\text{cond}}^{-1}\mathbf{V}^T\right)\lambda = -\tfrac{1+\epsilon}{\Delta t}\mathbf{J}\mathbf{v}^-, \tag{3.18}$$
$$\mathbf{U} = \mathbf{J}_c - \mathbf{J}_s\mathbf{M}_s^{-1}\mathbf{M}_{ec}, \qquad \mathbf{V} = \mathbf{J}_c - \mathbf{J}_s\mathbf{M}_s^{-1}\mathbf{M}_{ce}^T.$$

I account for the rank-6 matrix $\mathbf{U}\mathbf{M}_{\text{cond}}^{-1}\mathbf{V}^T$ by applying a Sherman-Morrison-Woodbury update [GL96] to the solution of the full-rank linear system given by $\mathbf{J}_s\mathbf{M}_s^{-1}\mathbf{J}_s^T$. For the solution of the full rank system, I approximate $\mathbf{M}_s$ by considering only 3×3 block diagonal terms of the stiffness matrix $\mathbf{K}_s$. This approximation has the effect of discarding the implicit integration of inter-node elastic forces in the computation of the collision force. Note that the approximation still captures the two-way coupling of elastic forces between the core and the deformable layer, thereby preserving the responsiveness of the core's motion to collisions. Due to the block diagonal approximation of $\mathbf{M}_s$ and the structure of $\mathbf{J}_s$, where each constraint only affects one simulation node, the matrix $\mathbf{J}_s\mathbf{M}_s^{-1}\mathbf{J}_s^T$ is diagonal, and can be trivially inverted.

Moving Constraints

If a node i collides against a moving constraint $\mathbf{n}$, I estimate the world-frame velocity $\mathbf{x}_o$ of the constraint at the time of maximum compression, and I rewrite the elastic collision equation (3.13) as:

$$\mathbf{n}^T\left(\mathbf{x}_i^- + \frac{\delta\mathbf{x}_i}{1+\epsilon} - \mathbf{x}_o\right) = 0, \tag{3.19}$$

To estimate $\mathbf{x}_o$, I rigidify the colliding bodies and compute the normal velocity at the point of contact under a perfectly inelastic collision [Mir96].

Friction

I compute frictional response based on Coulomb's model, with friction coefficient μ_i for each colliding node i. Based on the kinematic relationship (A.5), pre-impact velocities $\mathbf{v}^-$, frictionless impulsive response $\delta\mathbf{v}$, a constraint normal $\mathbf{n}$, and pre-impact tangential velocity $\mathbf{x}_i^t = \mathbf{L}_i\mathbf{v}^- - \left(\mathbf{n}^T\mathbf{L}_i\mathbf{v}^-\right)\mathbf{n}$, I compute a maximally dissipating friction impulse $\delta\mathbf{x}_i^t$ for node i similar to [BFA02] as:

$$\delta\mathbf{x}_i^t = -\mu\mathbf{x}_i^t, \qquad \mu = \min\left(1, \frac{\mu_i \ \mathbf{L}_i\delta\mathbf{v}}{\mathbf{x}_i^t}\right). \tag{3.20}$$

I conclude by applying a friction impulse to the elastic velocity of the colliding node as $\delta\mathbf{q}_i^t = \mathbf{R}^T\delta\mathbf{x}_i^t$.

Constraint Correction

After the computation of collision response, I perform a position update with the newly computed velocities. With a new texture-based collision detection step, I detect possible colliding nodes and their penetration depth d. For a colliding pair of nodes i and j, I estimate local stiffness k_i and k_j, and I determine the constraint position correction of node i to be

$$\delta\mathbf{x}_i = \frac{-k_j}{k_i + k_j} d\mathbf{n}. \tag{3.21}$$

Then, I correct the body-frame displacement of node i as

$$\delta\mathbf{q}_i = \mathbf{R}^T\delta\mathbf{x}_i. \tag{3.22}$$

3.5 Algorithm and Parallel Implementation

The implicit formulation of the dynamic motion equations and collision response yields linear systems of equations with dense coupling between the core and elastic velocities.

It is possible to formulate the velocity update and collision response in a highly parallelizable manner. Figure 3.4 is a schematic overview of the GPU algorithm for simulating and rendering deformable objects in contact using dynamic deformation textures maps. Figure 3.5 is an more detailed outline of the algorithm. Let s denote the operations that are performed on small-sized systems (i.e., computations of core variables, and low resolution collision detection). The remaining operations are all executed in a parallel manner on a large number of simulation nodes. Specifically, T refers to operations to be executed on all simulation nodes in the dynamic deformation texture T, D refers to operations to be executed on texels of the contact plane D, and T_D refers to operations to be executed on the colliding nodes. As highlighted in Figure 3.5, all operations to be executed on simulation nodes (indicated by T, T_D and D) can be implemented with parallelizable computation stencils on the GPU, as indicated in Figure 3.4 with purple diamond boxes. Moreover, due to the regular meshing of the deformable layer produced by dynamic deformation textures, the computation stencils are uniform across all nodes, hence they are amenable to implementation on a streaming processor such as the GPU. In Section 3.5.3, I will illustrate this concept for representing and computing sparse matrix multiplications in step 2 of Figure 3.5.

I exploit image-based computations also on the GPU for collision detection. Because the dynamically deforming surface is updated in texture memory directly, its state is available to the collision detection module without requiring an expensive update from the CPU host. The computations of per-texel penetration depth and contact normal are performed by orthonormal projection of the geometry, as described in Section 3.5.4.

Finally, after computing collision response of steps 6-15 and updating the position D2T in texture memory, the state of the surface is readily available for rendering. Section 3.5.5 describes how the deforming mesh is drawn to the screen using our D2T model representation.

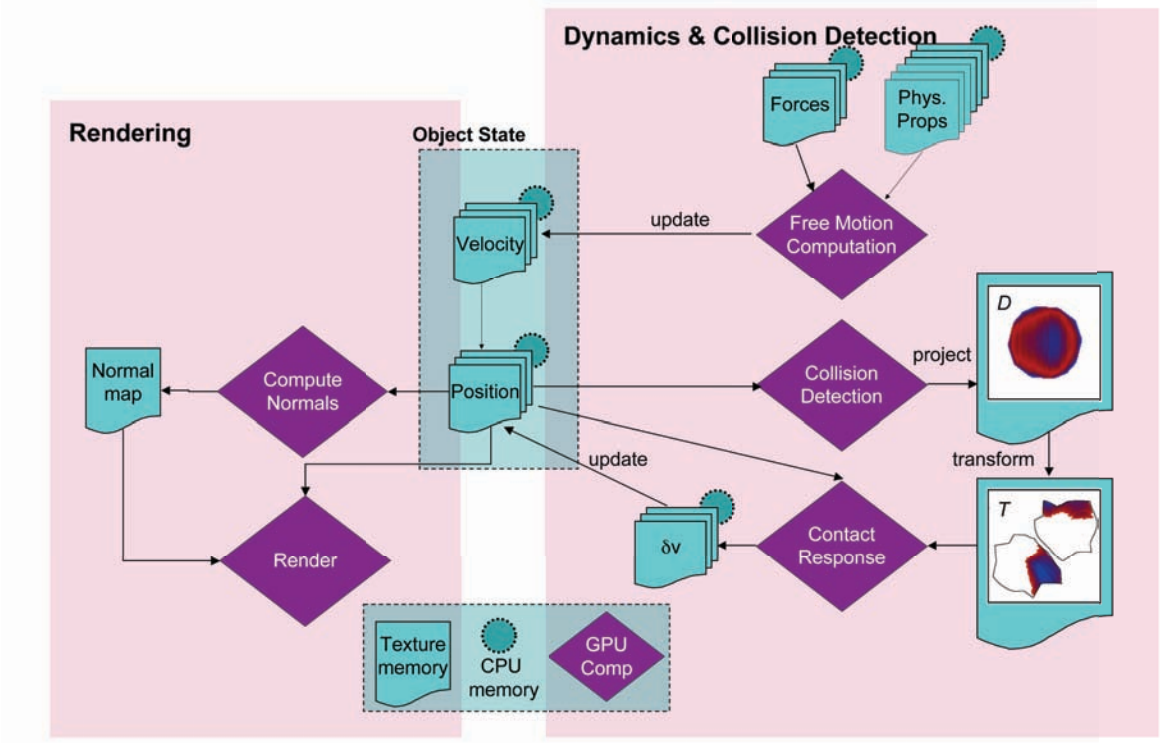


Figure 3.4: GPU Algorithm Overview.

3.5.1 Dynamic Deformation Textures

I encode the state of the deformable surface in *dynamic deformation textures* or D2Ts. A D2T consists of a texture atlas, with potentially multiple patches (Fig. 3.1), in which each texel (s, t) that falls within the patches implicitly represents a vertex on the surface. These texels are also referred to as *valid* texels. Each texel $(s, t) \in T$ maps to two corresponding points $f(s, t)$ and $g(s, t)$ on the surfaces of the core and the deformable object as indicated in Fig. 3.1. The regular sampling of T and the correspondence of surface points define implicitly a meshing of one layer of tetrahedral elements, as shown in Figure 3.6. By applying classical approximation methods such as FEM , the deformation field in the deformable layer can be approximated from the values at a finite set of nodes. Since there is never any deformation at points on the core, the deformation field can be approximated entirely from the values at surface nodes. Effectively, each texel $(s, t) \in T$ maps to a simulation node $g(s, t)$ in the FEM discretization. Simulation

	COLLISION-FREE UPDATE	
1.	Evaluate forces	T
2.	Solve the sparse linear systems $\mathbf{M}_s\mathbf{y} = \mathbf{F}_s$ and $\mathbf{M}_s\mathbf{Y} = \mathbf{M}_{ec}$ (Section 3.2.2), using a Conjugate Gradient solver [GL96]	T
3.	Update core velocities $\mathbf{v}_c^-$ using the condensed formulation (3.8)	s
4.	Update elastic velocities $\mathbf{v}_s^-$ using the new core velocities as in (3.9)	T
5.	Perform a position update $\mathbf{q}^- = \mathbf{q}(t) + \Delta t\mathbf{P}^+\mathbf{v}^-$	T
	COLLISION DETECTION	
6.	Execute low-resolution collision detection	s
7.	Execute high-resolution collision detection	D
8.	Map contact information to the dynamic deformation textures	T
	COLLISION RESPONSE	
9.	Invert the block-diagonalized full-rank matrix $\mathbf{J}_s\mathbf{M}_s^{-1}\mathbf{J}_s^T$ (Section 3.4.1)	T_D
10.	Solve for λ in (3.18) using the Sherman-Morrison-Woodbury formula	T_D
11.	Repeat steps 3 and 4 to obtain the collision impulse $\delta\mathbf{v}$, based on (3.16)	
12.	Compute friction impulse	T_D
13.	Perform a position update $\mathbf{q}(t + \Delta t) = \mathbf{q}^- + \Delta t\mathbf{P}^+(\delta\mathbf{v})$	T
	CONSTRAINT CORRECTION	
14.	Repeat collision detection steps 6 to 8	
15.	Apply constraint correction	T_D

Figure 3.5: Summary of the Simulation Algorithm

variables defined per-node, such as velocities, forces, mass and stiffness values, etc. can also be stored in the D2T texture atlases. Note that the implicitly defined texture-based meshing is not consistent at patch boundaries, which requires special yet simple treatment as discussed in Section 3.5.3.

In a preprocessing step, I tesselate the mesh from the vertex connectivity that is implicitly defined by the texel grid in the D2T texture (see Fig. 3.6). The implicitly defined triangle strips are encoded in a vertex index list I_M. Additional triangle strips are constructed to patch or *zipper* [TL94] the mesh at the cuts along the patch boundaries.

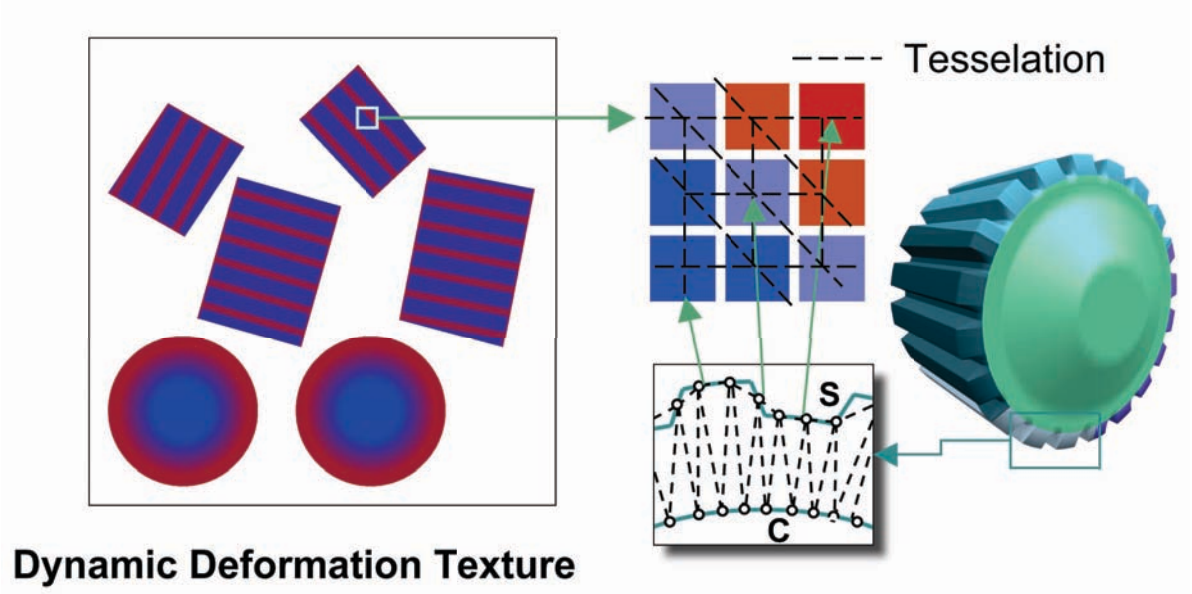


Figure 3.6: Dynamic deformation texture representation and implicit tesselation.

3.5.2 Basic Rendering Blocks

In this section, I define a few basic blocks that are used to render the deforming mesh to the screen and into the collision and simulation domains. Note that the representation of a deformable mesh is carefully chosen such that expensive GPU readback or host upload are avoided at all times. Therefore, the mesh topology is stored in a static index buffer on the GPU and all surface vertex position data is stored in texture memory, while the surface deformation simulation is computed using fragment programs on the GPU. The blocks are illustrated schematically in Fig. 3.7

UpdateMesh (UM) This block is used to update a dynamic vertex buffer V_M with the deformed surface vertex positions after each time step in the simulation. One approach to render the deforming surface given the dynamic deformation texture T on Vertex Shader 3.0 hardware, is to fetch the positions from T in the vertex shader. Each vertex can then be displaced according to the current position stored in T. Unfortunately, the less powerful vertex processing pipe and slow vertex-stage texture fetches of non-unified GPU architectures can make this approach a bottleneck, especially because

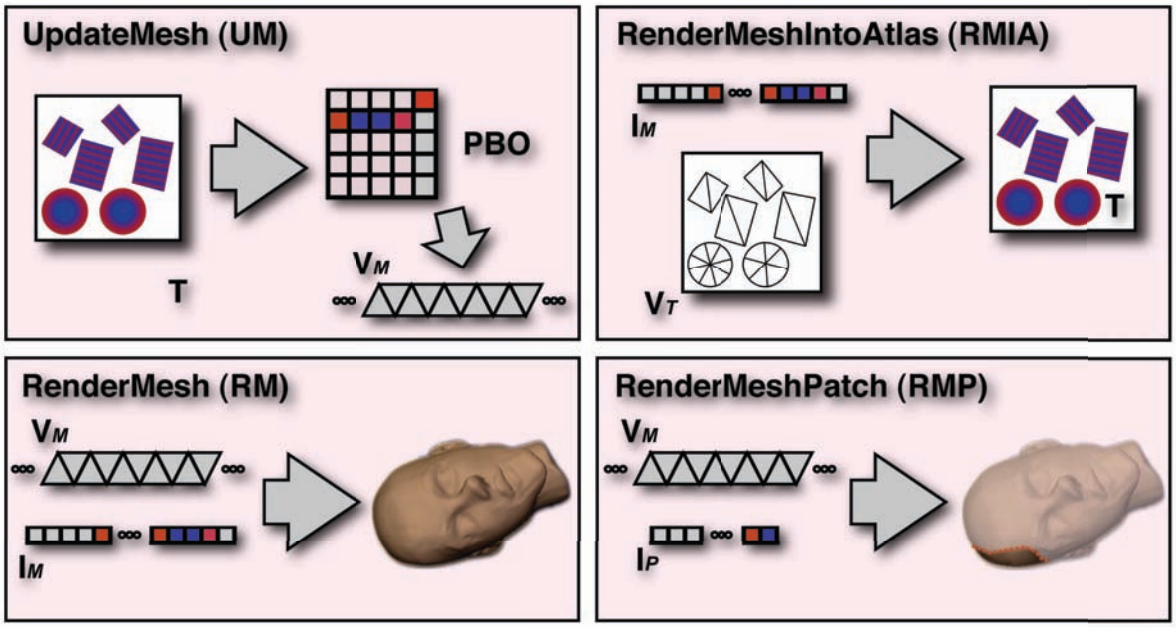


Figure 3.7: Rendering Blocks.

the UpdateMesh block will be used multiple times for the same snapshot in time. It would be wasteful to repeat the displacement in the vertex shader for collision detection, shadow map generation and multiple final render passes.

Therefore, the OpenGL PBO (Pixel buffer object) extension is used to copy the D2T texture T to a pixel buffer object that can later be interpreted by the OpenGL API as a vertex buffer object V_M (see Code Snippet E.1). This technique is often referred to as the PBO/VBO approach to render-to-vertex-array. This data copy is efficient because it is between two GPU memory areas: there is no data copy to or from the host. Note that in this approach not all memory locations in the PBO contain valid vertex data, because not all texels in T are valid (Sec. 3.5.1). The vertex indices in I_M are assigned such that they index into the correct location of the PBO. I store the triangle list in the static index buffer I_M; thus the vertices are rendered without any vertex bandwidth overhead with an indexed draw call (`glDrawElements()` for the OpenGL API).

RenderMesh (RM) This block is the encapsulation of the vertex processing stage on the GPU, when rendering a deformable mesh. Given the index buffer I_M and the dynamic vertex buffer V_M, the deforming geometry can be rendered efficiently with a single indexed draw call.

RenderMeshPatch (RMP) This block is identical to the RenderMesh block, except that the input index list I_P is not static. In this case, only a subset of the mesh's triangles is rendered by sending the vertex index list at each frame. As it is only a limited number of triangles, this is not a significant overhead.

RenderMeshIntoAtlas (RMIA) and RenderPatchIntoAtlas (RPIA) In many simulation parts of the algorithm, it is required to render values defined on the surface of the mesh into the D2T texture atlas. This can easily be achieved by the RenderMeshIntoAtlas block. The D2T texture coordinates are stored as positions a separate (static) the vertex buffer V_T. Therefore, through the use of the identity matrix as the model-view-projection matrix, the surface values are rasterized into the D2T texture atlas. The same operation can also be performed for a subset of the mesh triangles. This block is called RenderPatchIntoAtlas.

3.5.3 Simulation of Surface Deformations

As mentioned in Sec. 3.5.1, I perform dynamic simulation of the surface deformable object in the domain of the *dynamic deformation texture* (D2T). The goal of the dynamic simulation part of the algorithm is to compute the global motion of objects (i.e. the rigid motion of the core C) and to compute how the surface S deforms under influence of forces and accelerations. In practice, this can be done very efficiently exploiting the parallellism on the GPU in fragment programs while rendering the results directly to the dynamically changing D2T position texture which can then be used for collision detection and rendering. The only information communicated between CPU and GPU

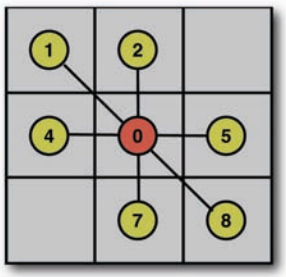


Figure 3.8: A texel in the D2T defines a simulation node. The figure shows its neighborhood in the FEM mesh. Its 6 neighbors and itself give rise to 7 non-zero blocks per block row in the stiffness matrix, as shown in Fig. 3.9.

are a few small state changes, typically 6-tuples or 3×3 matrices. These state changes are required for for updates that are related to the rigid transformation modes of C and for transferring forces and accelerations that are due to dynamic coupling between the deformable surface and the core.

This section will only touch on a few concepts and simple shaders that are being used to map step 2 in Algorithm 3.5 to the deformation simulation to the GPU pipeline. In reality, our implementation of all dynamics steps in Figure 3.5 consists of 50-100 different shaders that compute the different steps in the dynamics equations and contact handling.

Velocity and position updates At the core of the dynamics simulation of a mesh with n vertices, a large linear system $\mathbf{Ax} = \mathbf{b}$ (Equation (3.10)) has to be solved at each time step to compute the velocity at the next time step, where $\mathbf{x}$ and $\mathbf{b}$ are vectors of size n. The matrix $\mathbf{A}$ is a symmetric, positive definite sparse block matrix, where the non-zero blocks are 3×3 matrices (Fig. 3.9). Such a system can be solved with any variant of the conjugate gradients (CG) solver [She94]. The conjugate gradients method is an iterative solver and a very important building block of CG are sparse matrix multiplications of the form $\mathbf{y} = \mathbf{Ax}$.

In the remainder of this section, I will explain how $\mathbf{A}$ is stored and how these sparse matrix multiplies are performed in a fragment program.

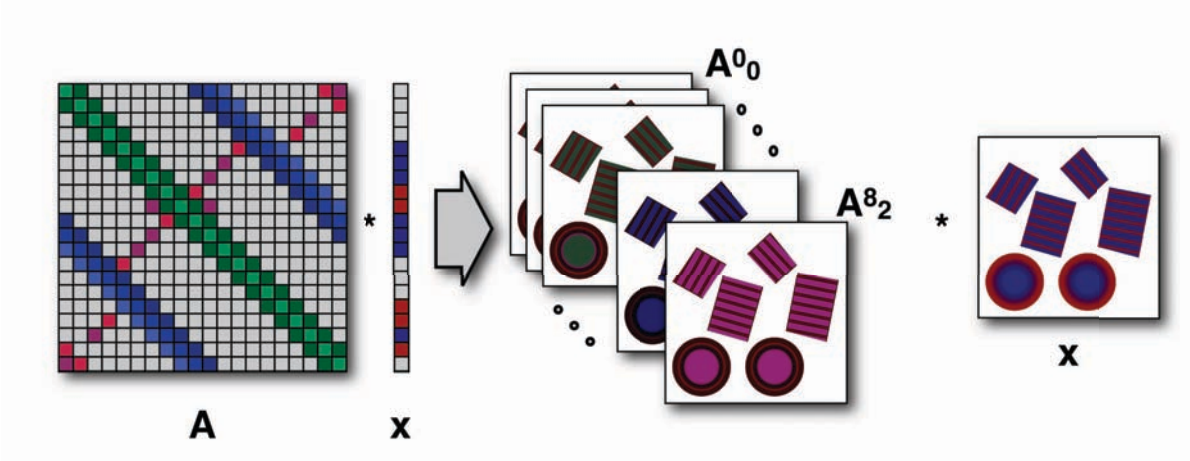


Figure 3.9: Sparse matrix multiplies on the GPU using D2T representation. The matrix **A** has 7 non-zero blocks per block row (left), which can be represented by 21 RGB textures that use the D2T atlas mapping (right).

Sparse Matrix Representation and Multiplication The vectors $\mathbf{x}$ and $\mathbf{y} \in \mathbb{R}^{3n}$ both define vector values (3-tuples) at each vertex. We already know from Section 3.5.1 that we can store those values at *valid* texels in the D2T texture atlas. We can also map $\mathbf{A}$ to the D2T atlas as follows. Each block row of $\mathbf{A}$ defines seven 3×3 blocks, one for each neighbor of a given vertex (or texel in the D2T) as shown in Fig. 3.8. Hence, we can store $\mathbf{A}$ in 21 RGB textures where each texture stores a 3×1 row of a 3×3 block (Fig. 3.9). Due to the limited number of texture samplers that can be bound to a fragment program within a pass, the actual sparse matrix multiplication has to be performed in two passes. Mathematically, this corresponds to the following transformation: $\mathbf{Ax} = \begin{bmatrix} \mathbf{A}_l & \mathbf{A}_r \end{bmatrix} \mathbf{x} = \mathbf{A}_l\mathbf{x} + \mathbf{A}_r\mathbf{x}$. In the second pass, the result of $\mathbf{A}_l\mathbf{x}$ is passed in from the first pass. Code Snippet E.2 shows the setup and invocation of the passes, while Fragment Programs E.1 and E.2 show the implementation in the fragment processor. Note that if $\mathbf{x}$ is a $n \times 3$ matrix instead of a vector of size n, the result is a $n \times 3$ matrix. This can still be achieved in 2 passes by rendering to multiple render targets simultaneously, storing 3 columns instead of 1.

This approach of matrix multiplies is very efficient on parallel streaming processors such as current GPUs, because there is no branching or pointer chasing involved.

Moreover, my mapping of the sparse matrix to the D2T atlas exploits the GPU texture caching architecture in two ways. First, due to tile based rendering, neighboring values fetched from $\mathbf{x}$ and $\mathbf{A}$ in one fragment are conveniently pulled into cache for neighboring fragments in the same tile. Second, fetching a value pulls in other neighboring values from $\mathbf{x}$ and $\mathbf{A}$ that are required in the same fragment program for free.

Patch Boundary Handling In the previous section, it was neglected that, at patch boundaries in the D2T, not all neighboring texels are valid texels. One solution could be to flag boundary texels in some way and use branching that is available in current GPU hardware, but this is not very efficient because the boundaries are not coherent fragment blocks. Better approaches are to rasterize and handle the boundary texels separately with a separate fragment program [Har05] or to guarantee that all neighbors are valid. I have taken the latter approach. I adapt a method by Stam [Sta03] for providing accessible data in an 8-neighborhood to all nodes located on patch boundaries. Before every sparse matrix multiplication step in the algorithm, I fill a $\overline{2}$-texel-width region on patch boundaries by sampling values on the adjacent patches. In practice, for each deformable model and D2T atlas, I maintain a list of thin quads that have texture coordinates assigned that map to locations of neighboring surface points across boundaries in the D2T texture atlas.

3.5.4 Texture-Based Collision Detection

I employ a GPU-accelerated image-space algorithm because it exploits the surface position data that is stored and simulated in fast texture memory. Therefore, the transfer of large amounts of mesh position data between CPU and GPU is avoided. Such data transfer could easily become a bottleneck for our system otherwise.

As proposed in Section 3.3, collision detection between two deformable objects A and B is performed in two steps:

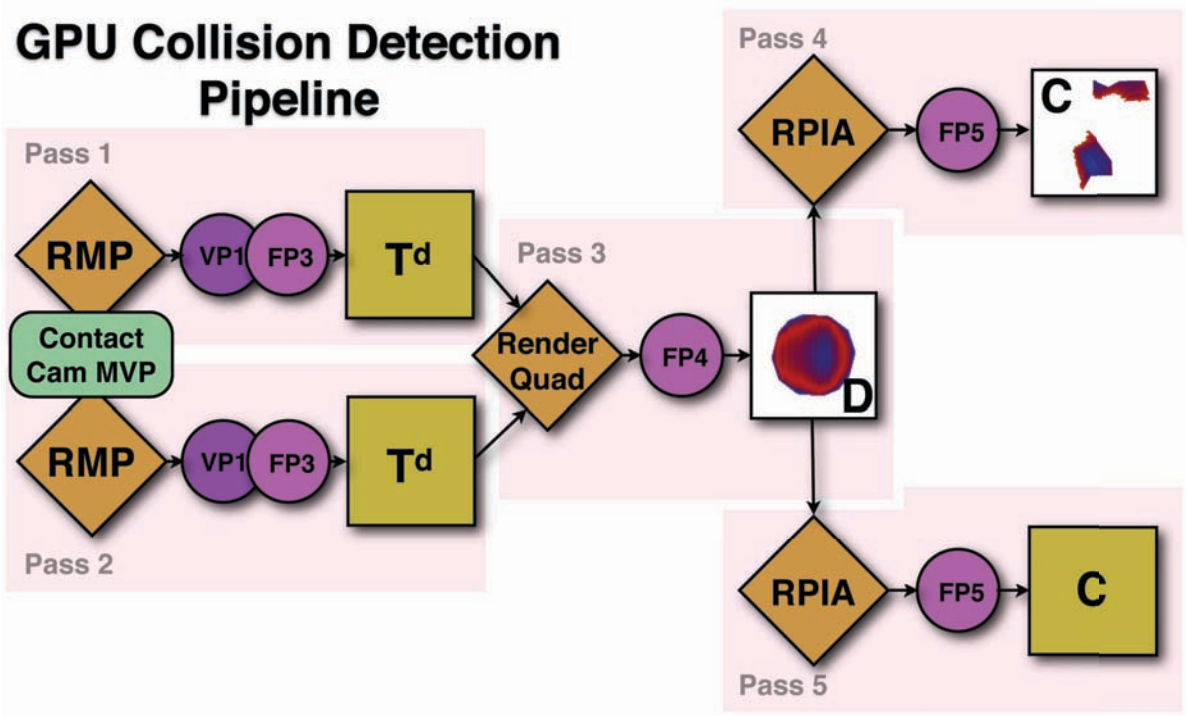


Figure 3.10: Schematic overview of the pipeline of my GPU-based collision detection algorithm, composed out of 5 passes.

1. Identify contact regions with object-space techniques using low-resolution proxies of the objects.

2. Compute contact information in the contact regions using image-space techniques and high-resolution displacement fields.

The second step in my algorithm is accelerated by the GPU. This stage utilizes the RenderMeshPatch block (Sec. 3.5.2) The draw call is restricted to the triangles that form the potentially colliding surface patch. My image-based algorithm consists of three substeps that are implemented by five rendering passes per pair of potentially colliding surface patches (Fig. 3.10)

In the first two passes, I perform a projection step for each potentially colliding surface patch. I set up an orthographic projection which we call the *contact camera*. The contact camera is carefully positioned such that it looks along the normal of the contact plane D and such that the projections C_A and C_B capture the full extent of

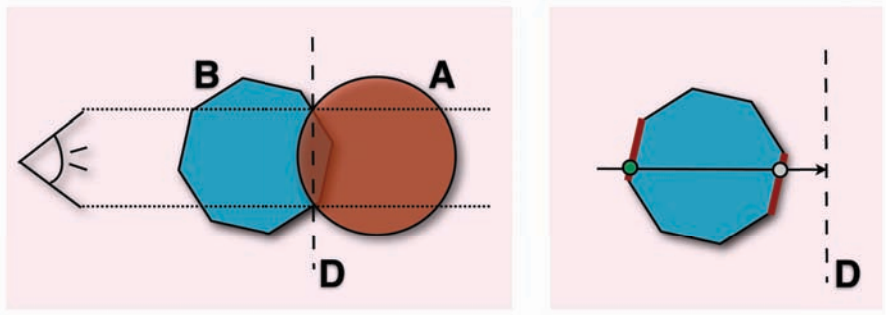


Figure 3.11: Left: The contact camera is set up with an orthogonal projection perpendicular to the contact plane D. **Right:** Multiple surface points may map to the same location on D. When texels in the D2T are tagged as colliding, a check is required which triangle (of the two red triangles) the rasterized fragment belongs to, in order to avoid tagging the green surface point as colliding.

the contact area of a pair of potentially colliding surface patches S_A and S_B (Fig. 3.11). Vertex Program E.1 and Fragment Program E.3 are used to rasterize the distance from the eye directly into textures T_A^d and T_B^d. Note that I enable front-facing triangles while rasterizing S_A into T_A^d and back-facing triangles while rasterizing S_B into T_B^d. In the third pass, I capture the areas of interpenetrating surface patches by constructing texture D from projections T_A^d and T_B^d. For each texel $(u, v) \in D$, I perform high-resolution collision detection by testing the distance between points $C_A(u, v) \in S_A$ and $C_B(u, v) \in S_B$ along the normal direction of D. If the points are penetrating, I identify a contact constraint and I compute the contact normal $\mathbf{n}$ as the average surface normal. In practice, as shown in the middle of Figure 3.10, I render a full-screen quad of the size of T_A^d and T_B^d into D, while Fragment Program E.4 computes the difference in distances. Positive values indicate penetration in the projection as indicated by the red regions on the left in Fig. 3.3. Note that I also write the triangle ID of the current fragment to D. These IDs are used in the next pass to check whether a rasterized texel of the D2T is originating from the triangle whose fragments were rasterized into D and not from a triangle that maps to the same texel in D (see Fig. 3.11).

Recall that the deformation of the sphere is stored in the two-dimensional texture atlas T called *dynamic deformation texture* (D2T). This texture atlas is shown on the

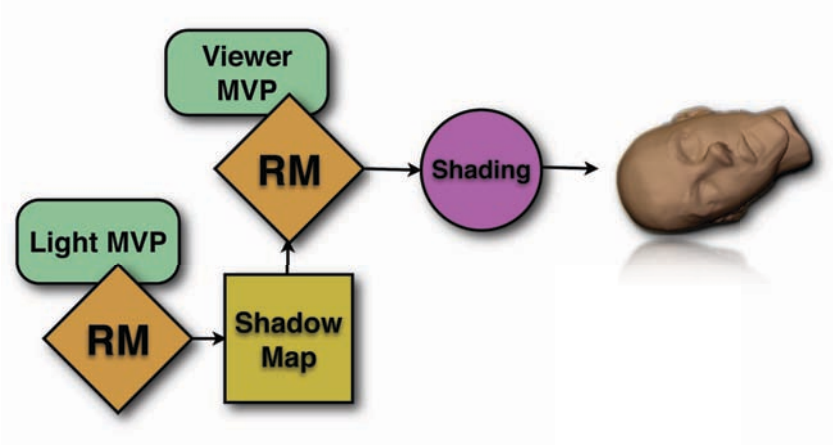


Figure 3.12: Rendering pipeline. Note that the RenderMesh (RM) block utilizes the vertex stream with normals generated as in Fig. 3.13.

right in Fig. 3.3. Dynamic contact response is computed in this domain. Therefore, the collision information in texture D has to be transferred to the dynamic deformation texture T via a mapping that is the combination of the inverse of the orthogonal contact projection with the D2T texture atlas mapping. In practice, this step is performed by the two last passes of our algorithm. These passes render each potentially surface geometry again using the RenderMeshIntoAtlas block (Section 3.5.2) into the D2T domain. I set up the texture matrix to perform the correct mapping while fetching values from texture D. The required texture matrix set up is completely analogous to the typical setup for traditional shadow mapping. Here, the contact camera model-view-projection matrix takes the place of the light's model-view-projection matrix. Snippet E.3 shows the code that is used for this setup. Fragment Program E.5 shows the pixel shader code of the last two passes, one shader per object.

3.5.5 Rendering

Using the RenderMesh block defined in Sec. 3.5.2, rendering a deformable mesh represented by D2T position textures and the additional data structures described in Sec. 3.5.1 and Sec. 3.5.2 is relatively straight forward (see Fig. 3.12). A standard frag-

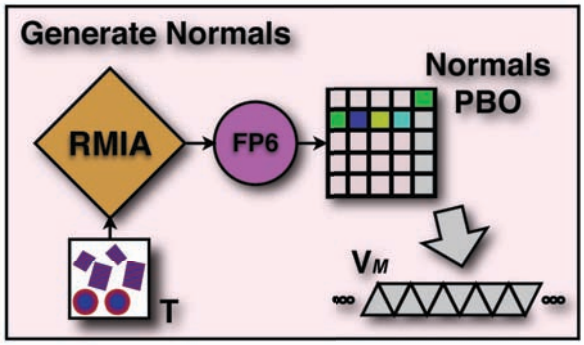


Figure 3.13: Normal Generation block. A normals PBO is generated and then copied to the normal vertex buffer.

ment program that computes per-pixel shading is plugged into the pipeline and the RenderMesh block can also be used to generate a standard shadow map.

The only missing piece of information are the vertex normals. As the geometry is deforming, normals have to be recomputed at each frame (or each few frames). There are two approaches possible. On Shader Model 4.0 (DirectX10) compatible hardware, the normals can be computed in a geometry shader provided that an appropriate triangle adjacency list is sent to the GPU. Alternatively, on older hardware, one can generate a normal map using the D2T texture atlas. This process is illustrated in Fig. 3.13 along with Fragment Program E.6. Here, as for sparse matrix multiplication in Sec. 3.5.3, the input D2T texture has to be augmented with replicated position information along the patch boundaries. This ensures that each D2T texel neighborhood is valid and can be sampled to approximate the corresponding vertex normal. The normals vertex buffer can be updated with the normal map using the PBO technique that was also used when updating the position vertex buffer in Sec. 3.5.2.

3.6 Benchmarks

The experiments described in this section were performed on a 3.4 GHz Pentium-4 processor PC with a Nvidia GeForce 7800GTX graphics card. Table 3.1 lists the statistics of

Figure 3.14: Deformations of High-Resolution Geometry. Left: Two deformable pumpkins are dropped on top of each other. Right: Detail view of the rich deformations produced on the top pumpkin during contact.

the used models. In all cases, the size of the dynamic deformation textures was 256×256 texels. Such high resolution enables the simulation of rich deformations, as shown most clearly in the pumpkins (Figure 3.14), gear (Figure 3.15), and head (Figure 3.17) models. With our constraint-based collision response approach, impacts produce highly-detailed indentations such as the ones suffered when the pumpkins are dropped on each other, or when the fist punches the head on the eyebrow. The gear model demonstrates rich dynamic deformations of surface features larger than 30% of the object radius.

Model	Tire	Cylinder	Pumpkin	Gear	Head
Nodes	31K	21K	30K	29K	40K
Tetrahedra	162K	161K	183K	173K	240K

Table 3.1: Models and Statistics.

The use of sound physically-based techniques for modeling contact and deformations leads to highly plausible rolling and tumbling motion in combination with surface deformations, as can be observed in the tires (Figure 3.16) and gear (Figure 3.15) scenes. With

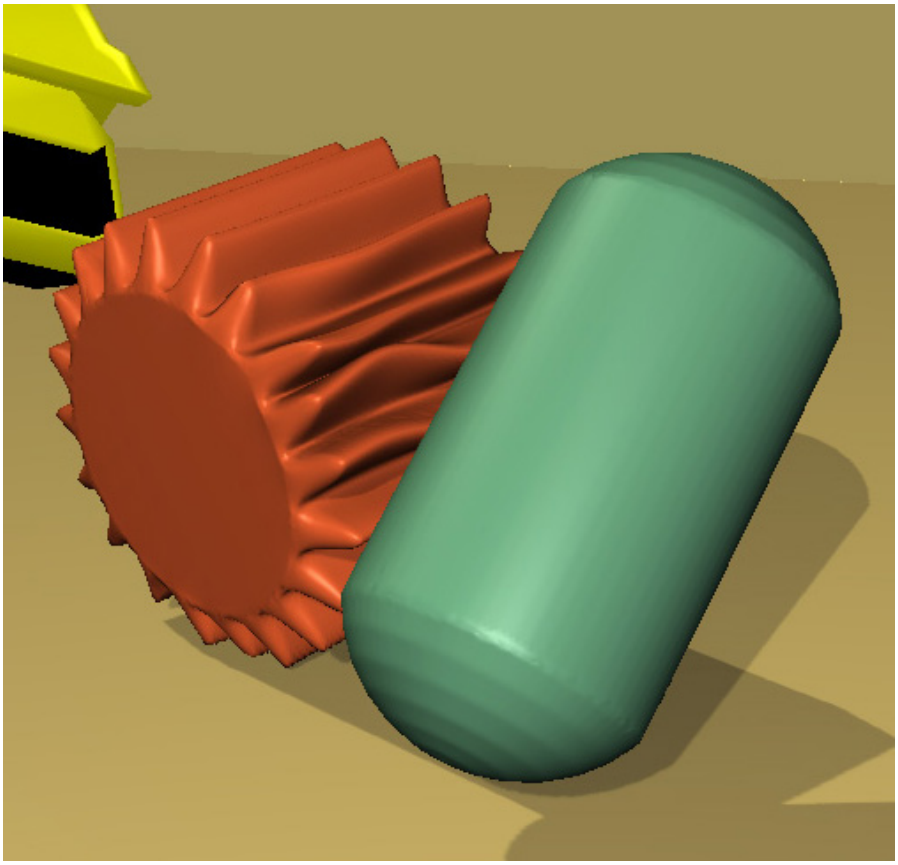


Figure 3.15: Deformations of High-Resolution Geometry. A dropped cylinder produces rich dynamic deformations on the ridges of a gear.

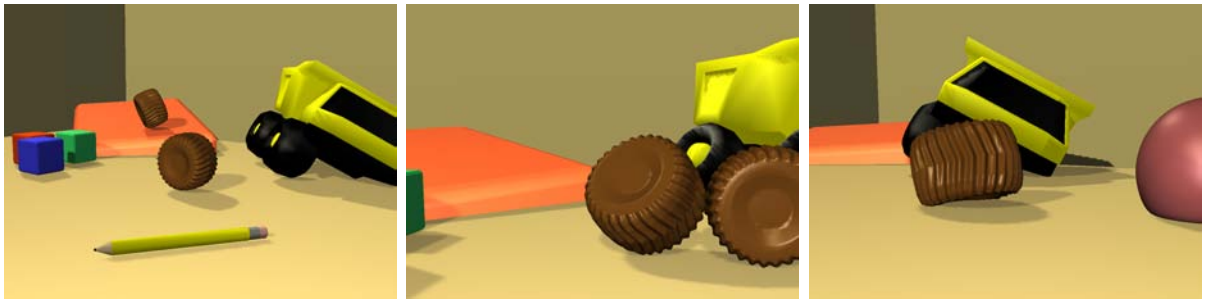


Figure 3.16: Deformable Objects Roll and Collide in the Playground.

my deformable object representation and simulation algorithm, I achieve those effects on high-resolution objects in an efficient manner. The computational cost is dominated by the iterative solver, and its convergence depends mostly on the stiffness of the regions in contact. As a reference, the simulation of the rolling heterogeneous cylinder depicted in Figure 3.2 runs at an average of $1-2$ fps when the stiff part (Young modulus 60KN/m^2)

is in contact with the ridged terrain. On the other hand, the same simulation runs at an average of 6 fps when the soft part (Young modulus 3KN/m^2) is in contact, with up to 2600 simultaneously colliding nodes. This translates to a throughput of approximately 1M tetrahedra and 120K surface simulation nodes simulated per second. In the rest of the experiments, similar average performance is observed: 2 seconds/frame for the simulation of the tires (Figure 3.16), 1 fps for the punch (Figure 3.17) and cylinder-with-gear (Figure 3.15) scenes, and 2 fps for the dropped head (Figure 3.17).

3.7 Comparisons and Discussion

Method	DoFs	Contact	Performance
BNC96	surface	explicit	11K nodes/sec
ZC99	volume	explicit	303K els./sec
PPG04	surface	LCP	2K contacts/sec
MG04	volume	explicit	63K els./sec
D2T	surface	implicit Lagrange mult.	120K nodes./sec 1M els./sec 15K contacts/sec

Table 3.2: Approximate Performance Data Benchmark. Extrapolated performance data from [BNC96], [ZC99], [PPG04], [MG04] shown with mine, **D2T**.

I have chosen a few related techniques as a basis for benchmarking the overall performance of the algorithm. It is, however, very difficult to compare the various techniques, as their primary goals are often different. The algorithm in this chapter complements the prior work by offering an efficient, robust contact handling method for colliding deformable bodies with *large contact regions and high-resolution surface geometry*, but cannot simulate arbitrary large deformations. In Chapter 4, I present a method that does not have this restriction, providing a solution for soft articulated characters. I have extrapolated performance data using Moore's Law (performance increases 2x every 18 months). As indicated in Table 3.2, the performance of my approach (D2T) (up to 15K contacts/sec, 1M tets/sec, and 120K nodes/sec for moderately soft objects), is compara-

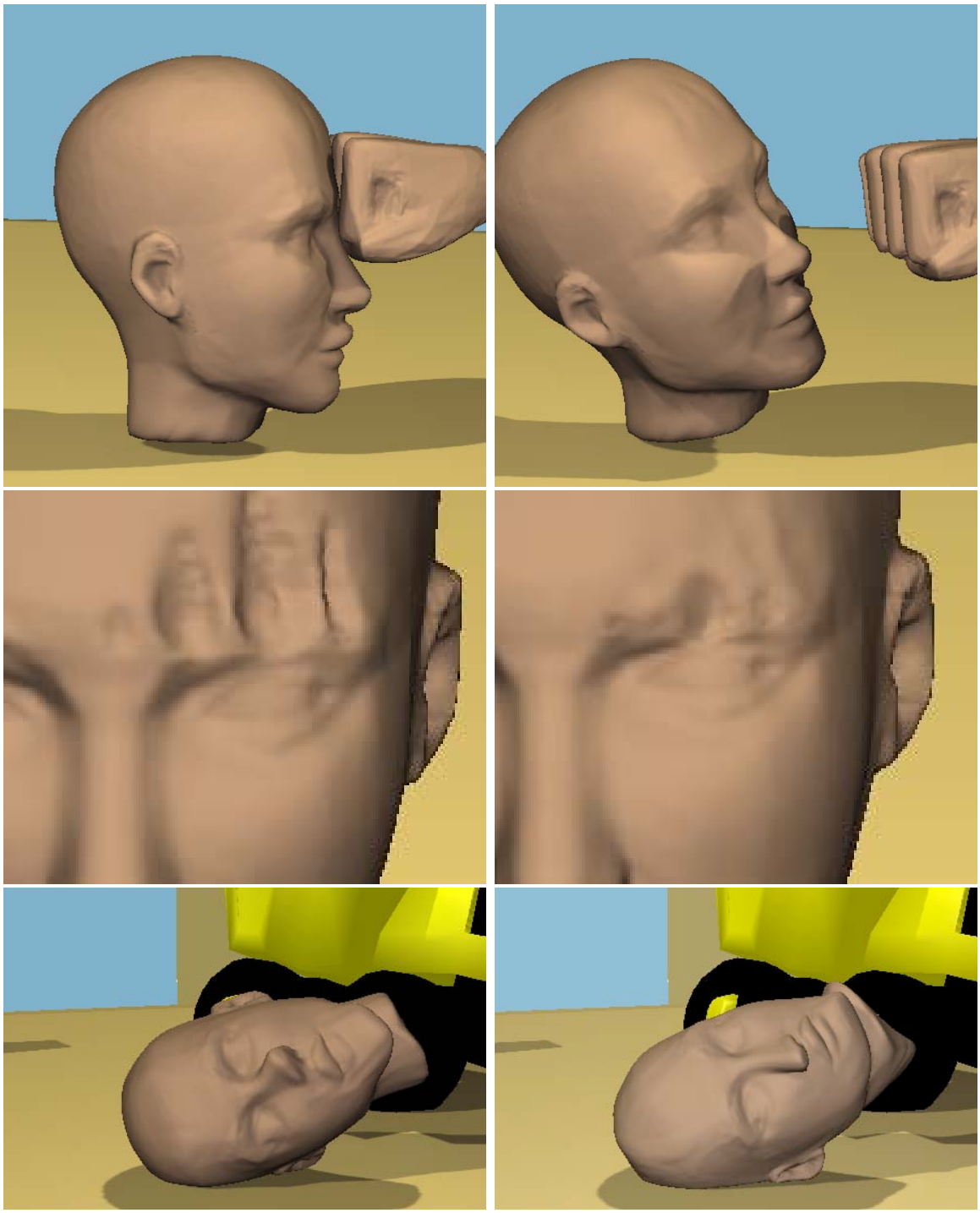


Figure 3.17: Deformations of a Virtual Head. Top: A fist hits a deformable head (attached by springs in the neck area), producing both local deformations and global motion. Middle: Detail of the deformations produced near the eyebrow by the impact. Bottom: A softer head is dropped on the floor, resulting in larger deformations.

ble to the performance of techniques that use explicit integration (e.g. [ZC99]), without their time-step restrictions. My approach is considerably faster than other methods that enable large time steps, both those that focus on the surface deformation (such as [BNC96]), and efficient co-rotational methods that compute deformations within the entire volume (such as [MG04]). My approach can also handle many more contact points than novel quasi-rigid dynamics algorithms using LCP [PPG04] while also producing richer deformations. Though computationally very efficient, this method cannot achieve a performance comparable to model reduction techniques that precompute data-driven models and build efficient low-rank approximations of deformed shapes [BJ05]. On the other hand, my approach does not require lengthy pre-computation of dynamics and achieves rich high-resolution deformations with both local and global support.

3.8 Advantages and Summary

In summary, my approach offers the following advantages:

- With the reformulation of the 3-dimensional elastoplastic deformations and collision processing on 2-dimensional dynamic deformation textures, the resulting system achieves fast and robust simulations of contacts between deformable bodies with *rich, high-resolution* surface geometry.

- Using a two-stage collision detection algorithm, the proximity queries are *scalable* and *output-sensitive*, i.e. the performance of the queries does not directly depend on the complexity of the surface meshes.

- By decoupling the *parallel* update of surface displacements from the update of the core DoFs, my efficient implicit formulation enables fast, stable simulations of heterogeneous materials under large time steps.

- The constraint-based collision response, using Lagrange multipliers and approximate implicit integration of elastic forces, provides *fast and responsive contact handling*, alleviating time-step restrictions of previous impulsive methods.

- The surface detail attributes stored in *dynamic deformation textures* can also be used directly for high-quality real-time shaders.

My mathematical formulation of dynamic simulation and contact processing, along with the use of dynamic deformation textures, is especially well suited for realization on commodity SIMD or parallel architectures, such as graphics processing units (GPU), Cell processors, and physics processing units (PPU). I have demonstrated the implementation of dynamic deformation textures on parallel processors, achieving fast simulation of complex scenarios with detailed deformations and thousands of simultaneous collisions.

3.9 Limitations and Future Work

The use of a layered representation obviously poses some limitations on the type of deformations that can be modeled. Nevertheless, it is possible to capture large deformations of as much as 30-40% of the object's radius successfully. This layered representation can be extended to articulated, flexible bodies that undergo skeletal deformations, by augmenting the generalized coordinate set of the core representation to include multibody systems. This is demonstrated in Chapter 4. Due to the issues regarding boundary patches discussed in Section 3.5, it may be harder to port the texture representation to articulated models. Such models have rather complicated texture atlases. As the atlas complexity goes up, so does the patch boundary complexity. In my experiments, the convergence rate of the conjugate gradients solver decreases as the boundary complexity goes up.

Chapter 4

Articulated Soft Character Simulation with Fast Contact Handling

In this chapter, I present a method for simulating soft characters with contact constraints in a unified framework for real-time modeling of character deformations due to contact with the interplay between skeletal deformations and surface deformations, for objects with thousands of deformable surface vertices.

The method proposed in Chapter 3 simulates objects with a rigid core and a highly detailed deformable skin, but does not support *global* deformations of the core, such as the bending of an arm, or twisting of a shoulder joint. These effects are essential to realistic animation of virtual characters such as humans and animals. In this chapter, I present a method that supports characters with an *articulated* core, such as a skeleton with bones and joints. Building on the concepts of Section 3.2, I extend *layered representations* to soft *articulated* characters, which is essentially an integration of articulated body dynamics and skinning with displacement corrections (see Section 4.1). While this representation cannot capture *general* global deformations, it is nevertheless well suited for representing skeletal and surface deformations. One of the challenges for modeling soft articulated characters that has not been well investigated previously is the interplay of skeletal motion and surface contact and the resulting two-way coupling effects. Another major issue is the enforcement of contact constraints on soft articulated bodies

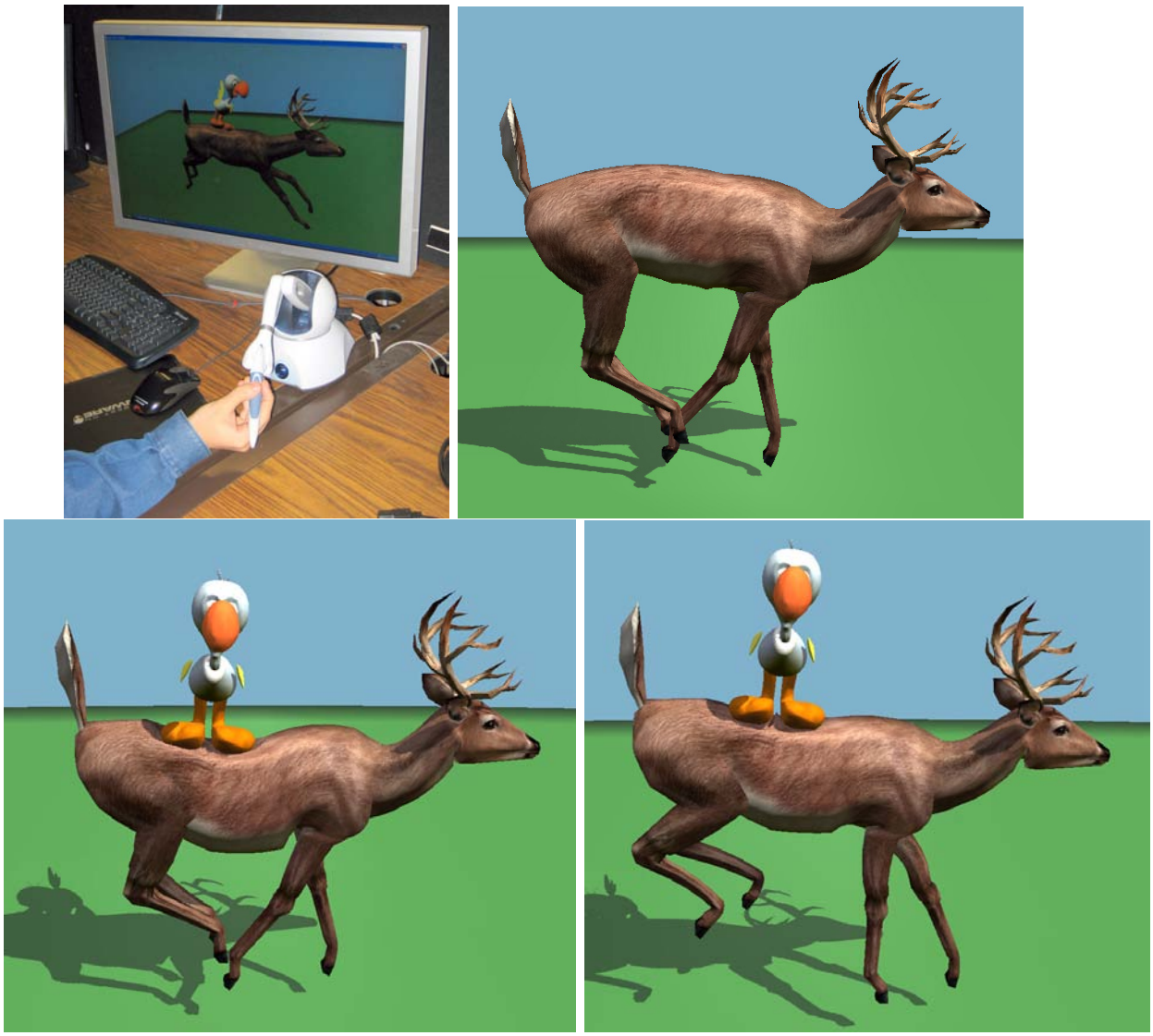


Figure 4.1: Interactive Deformation of an Articulated Deer. The deer, consisting of 34 bones and 2 755 deformable surface vertices is being deformed interactively (almost 10 fps on average) by a rigid bird model. The interplay between small-scale contact deformations and the skeletal contact response is successfully captured (below). The interactivity of my approach is demonstrated on the top left picture, where the bird is controlled in real time by a 3-DoF haptic controller.

with many degrees of freedom. In this chapter, I also extend the image-space collision detection algorithm of Section 3.3 to support articulated characters. My extended algorithm performs fast *hierarchical* collision queries between deformable characters whose surface is computed by displacements from (weighted) rigid bones, and it overcomes the

computational bottlenecks due to contacts.

My algorithm overcomes the computational challenges of soft character simulation by robustly decoupling skeleton and skin computations using careful approximations of Schur complements (Section 4.3), and efficiently performing collision queries by exploiting the layered representation. With this approach, this simulation framework can simultaneously handle large contact areas, produce rich surface deformations, and capture the collision response of a character's skeleton.

4.1 Layered Articulated Soft Characters

In this section I describe the formulation of deformations in the pose-space of an articulated character. I define the set of generalized coordinates composed of bone transforms and skin vertex deformations, and discuss the FEM discretization of the deformation field.

4.1.1 Pose-Space Deformation

Given a skeletal-subspace deformation model with k bones, the deformed position $\mathbf{x}$ of a material point is defined based on the position $\mathbf{u}$ in pose space $\mathsf{T}_{o\,i}$ and bone transformations T_i as

$$\mathbf{x} = \sum_{i=1}^{k} w_i \mathsf{T}_i \mathbf{u}_i = \sum_{i=1}^{k} w_i \mathsf{T}_i \mathsf{T}_{o\,i}^{-1} \quad \mathbf{u}. \tag{4.1}$$

I choose to express deformation and elastic energy in pose-space (also known as the bind pose of the articulated mesh) before applying the skin transformations (see Figure 4.2). This approach has been proposed previously for geometric deformation and displacement corrections [JT05, KJP02, LCF00]. Pose-space offers a local coordinate frame on which we can measure elastic energy using a linear strain tensor without

suffering from geometric non-linearities [MDM+02].

The deformed position in pose space $\mathbf{u}$ can be decomposed into a constant undeformed component $\mathbf{u}_o$ and an elastic skin displacement $\mathbf{u}_s$, hence

$$\mathbf{u} = \mathbf{u}_o + \mathbf{u}_s. \tag{4.2}$$

The bone transforms are chosen to be rigid transforms. The bone-frame position $\mathbf{u}_i$ can be then be defined:

$$\mathbf{u}_i = \mathsf{T}_{o\,i}^{-1}\mathbf{u} = \mathbf{c}_{o\,i} + \mathbf{R}_{o\,i}\mathbf{u}. \tag{4.3}$$

with $\mathbf{c}_{o\,i}$ a displacement and $\mathbf{R}_{o\,i}$ a rotation matrix. The constant transformations $\mathbf{c}_{o\,i}$ and $\mathbf{R}_{o\,i}$ transform world-space surface positions in rest state to each bone's reference system. It is also possible to make Eqn. 4.1 more explicit for rigid transforms:

$$\mathsf{T}_i\mathbf{u}_i = \mathbf{c}_i + \mathbf{R}_i\mathbf{u}_i, \tag{4.4}$$

with $\mathbf{c}_i$ a displacement and $\mathbf{R}_i$ a rotation matrix. Note that the blend weights w_i are assumed to obey the affine constraint $\sum_i w_i = 1$.

4.1.2 Discretization and Meshing

The layered model described here can be regarded as a generalization of the model in Section 3.2 to account for articulated motion. In this representation, the degrees of freedom (DoFs) are determined by the DoFs of the bone transforms T and the DoFs of the deformable layer. The deformation field $\mathbf{u}_s$ in the deformable layer is discretized using linear FEM. The deformation field $\mathbf{u}_s$ can then be approximated by n discrete node values accumulated in a vector $\mathbf{q}_s \in \mathbb{R}^{3n}$ through the (position-dependent) shape matrix $\mathbf{S}$ and expressed compactly as $\mathbf{u}_s = \mathbf{S}\mathbf{q}_s$.

An advantage of this method is that we can generate dynamic models from skinned

meshes that have been created with popular 3D authoring software. A volumetric mesh of the deformable layer can be generated with any method that preserves the original outer surface vertices, either by defining two enclosing surfaces [HGB06] or by generating a layer of tetrahedral elements inwards from the outer surface [EDS05]. The mesh blend weights can simply be reused for the physical model as defined in Eqn. (4.1).

By replacing the deformation field in Eqn. (4.1) with its discretized version, the expression for the position of a vertex is obtained:

$$\mathbf{x} = \sum_{i=1}^{k} w_i(\mathbf{c}_i + \mathbf{R}_i(\mathbf{c}_{o\,i} + \mathbf{R}_{o\,i}(\mathbf{u}_o + \mathbf{S}\mathbf{q}_s))). \tag{4.5}$$

The DoFs of our model can be packed together in the generalized state vector

$$\mathbf{q} = \begin{matrix} \mathbf{q}_c \\ \mathbf{q}_s \end{matrix}, \tag{4.6}$$

where

$$\mathbf{q}_c = \begin{bmatrix} \mathbf{c}_1^T & \theta_1^T & \dots & \mathbf{c}_k^T & \theta_k^T \end{bmatrix}^T \in \mathbb{R}^{7k} \qquad \text{for } k \text{ bones.} \tag{4.7}$$

We chose quaternions to represent the orientations θ.

The velocity state vector $\mathbf{v}$ follows from the time differentiation of Eqn. (4.5). As shown in Appendix A.2, the world-frame velocity $\mathbf{x}$ of a material point can be approximated as $\mathbf{x} = \mathbf{L}_W\mathbf{v}$. Here, the velocity state vector is

$$\mathbf{v} = \begin{matrix} \mathbf{v}_c \\ \mathbf{v}_s \end{matrix}, \tag{4.8}$$

with

$$\mathbf{v}_c = \begin{bmatrix} \mathbf{c}_1^T & {}^T_1 & \dots & \mathbf{c}_k^T & {}^T_k \end{bmatrix}^T \in \mathbb{R}^{6k}. \tag{4.9}$$

The angular bone velocities are expressed in the bone's local frame. The velocity state

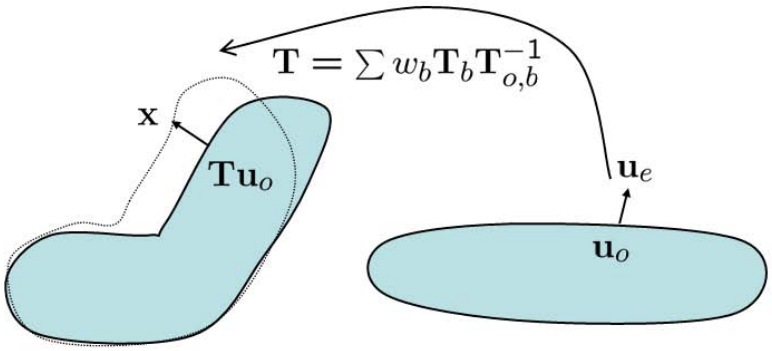


Figure 4.2: Pose space deformation. Elastic deformations $\mathbf{u}_s$ of the skin are defined in bind-pose-space.

vector and generalized coordinate vector are related by $\mathbf{v} = \mathbf{P}\mathbf{q}$ and $\mathbf{q} = \mathbf{P}^+\mathbf{v}$, with $\mathbf{P}$ and $\mathbf{P}^+$ matrices that transform angular velocities to time derivatives of quaternions in $\mathbf{q}$. Note that the discretized deformation model in Eqn. (4.5) is identical to the deformation model in Section 3.2 for the case of a single bone ($k = 1$).

4.2 Layered Dynamics with Contact Constraints

In this section I formulate the constrained dynamic simulation problem for soft characters. I model both joint and contact constraints with the method of Lagrange multipliers, and I use implicit backward Euler integration with linearization of forces.

4.2.1 Coupled Layered Dynamics in Free Motion

I formulate the dynamic motion equations of soft articulated characters using Lagrangian continuum mechanics [GPS02, Sha89]. Using linear elasticity theory and linear FEM and by formulating the displacement field of the soft layer in pose space, the elastic forces can be regarded as linear with respect to the displacements $\mathbf{u}_s$, and as invariant to the rigid bone transformations T in Eqn. (4.1) [Sha89, TW88].

The elastic energy given by pose-space displacements and linear elasticity yields the usual sparse block $\mathbf{K}_s$ of the stiffness matrix $\mathbf{K}$ that affects only DoFs of the soft skin layer $\mathbf{q}_s$, not the skeleton:

$$\mathbf{K} = \begin{matrix} \mathbf{K}_b & 0 \\ 0 & \mathbf{K}_s \end{matrix} . \tag{4.10}$$

However, note that, as shown in Section 4.3.5, the skeletal response of surface contact forces is still naturally captured. On the other hand, pose-space strain (as defined in Section 4.1.1) does not model pose-dependent strain energy. As an example, this is apparent for large bending of an elbow. Due to compression of tissue in the elbow region, one expects a reactive force that prevents the elbow to bend further. This is not captured by pose-space strain. Instead, I propose to capture this effect partially in the skeleton dynamics by adding a joint stiffness term between connected bones in the skeleton. This approach leads to off-diagonal non-zero blocks in the skeleton stiffness block $\mathbf{K}_c$. The derivation of the joint stiffness terms is given in Appendix C .

The kinetic energy depends on both skeleton and skin velocities, and it captures the interplay of articulated motion and skin deformation. My pose-space linearized deformation model bears similarity with the one of Capell et al. [CBC+05], but I effectively capture inertial forces by directly considering pose space deformations in the Lagrangian formulation instead of using co-rotational methods [MDM+02].

From Lagrangian continuum mechanics (see Appendix A.2 and [Sha89]), the inertia

matrix $\mathbf{M}$ can be derived:

$$\mathbf{M} = \quad \rho \mathbf{L}_W^T \mathbf{L}_W dV. \tag{4.11}$$

This matrix has the following structure:

$$\mathbf{M} = \begin{matrix} \mathbf{M}_b & \mathbf{M}_{bs} \\ \mathbf{M}_{bs}^T & \mathbf{M}_s \end{matrix}, \tag{4.12}$$

with $\mathbf{M}_b \in \mathbb{R}^{6k \times 6k}$ the inertia of bones and $\mathbf{M}_s \in \mathbb{R}^{3n \times 3n}$ the inertia of the skinned surface. Due to skinning blend-weights, $\mathbf{M}_b$ computed from the full Lagrangian would present off-diagonal blocks. However, the inertial coupling between bones is dominated by joint constraints, hence I compute a block-diagonal $\mathbf{M}_c$, where for each bone I compute the inertia by associating approximate link geometry (See Figure 4.3). Note that this approximation requires that the bone coordinate frames are located at the center of mass of the approximate link geometries. The dense bands $\mathbf{M}_{bs}$ and $\mathbf{M}_{bs}^T$ are key for capturing the effect of surface contact forces on bone motion through inertial coupling.

I incorporate Raleigh damping $\mathbf{D}$ (see Section 3.2.1), generalized external forces $\mathbf{Q}$, and a quadratic velocity vector $\mathbf{Q}_v$ [Sha89] that represents the inertial effects of centripetal and Coriolis forces on the bones. A set of ordinary differential equations (ODEs) follows from assembling all terms in Lagrange's equation [Sha89]:

$$\begin{aligned} &\mathbf{M}\mathbf{v} = \mathbf{Q} + \mathbf{Q}_v - \mathbf{K}\mathbf{q} - \mathbf{D}\mathbf{v} = \mathbf{F} + \mathbf{J}_\mu^T \mu + \mathbf{J}_\lambda^T \lambda, \\ &\mathbf{q} = \mathbf{P}^+ \mathbf{v}. \end{aligned} \tag{4.13}$$

I explicitly separate joint constraint forces $\mathbf{J}_\mu^T \mu$ and contact forces $\mathbf{J}_\lambda^T \lambda$ from other forces $\mathbf{F}$. In the discrete formulation of joint and contact forces (see Sections 4.2.2 and 4.2.3), the Lagrange multipliers μ and λ will include the time discretization Δt, and can be regarded as impulses.

I have discretized the motion equations using implicit backward Euler with a first

order approximation of forces, as this method allows for stable and responsive contact response. Please refer to Section 3.3 for the rationale behind implicit integration of contact forces. The discretized motion equations have the form

$$\mathbf{M}\Delta\mathbf{v} = \Delta t\mathbf{F}, \tag{4.14}$$

with discrete-time mass matrix $\mathbf{M}$ and discrete-time force vector $\mathbf{F}$ defined as

$$\mathbf{M} = \mathbf{M} - \Delta t\frac{\mathbf{F}}{\mathbf{v}} - \Delta t^2\frac{\mathbf{F}}{\mathbf{q}}\mathbf{P}^+ \tag{4.15}$$

$$\mathbf{F} = \mathbf{F} + \Delta t\frac{\mathbf{F}}{\mathbf{q}}\mathbf{P}^+. \tag{4.16}$$

Given the separation of forces in Eqn. (4.13), and as performed in a similar manner in Chapter 3 and by others [Erl04, CW05], I decompose the dynamic update into three steps:

1. Computation of collision free velocities $\mathbf{v}^- = \mathbf{v}(t-\Delta t)+\Delta\mathbf{v}$ from the old velocities, using Eqn. (4.14).

2. Collision detection and identification of contact constraints.

3. Computation of collision response $\delta\mathbf{v}$ that yields constrained velocities $\mathbf{v}(t) = \mathbf{v}^- + \delta\mathbf{v}$.

To ensure enforcement of constraints on positions as well, I apply a final correction step that projects (possibly) penetrating vertices to the constraint surfaces.

4.2.2 Joint Constraints

I use the method of Lagrange multipliers to compute joint constraint forces [Bar96]. It is important to observe that the joint constraints do not influence the skin coordinates. Hence, $\mathbf{J}_\mu$ is of the form $\mathbf{J}_\mu = \left[\begin{array}{cc} \mathbf{J}_j & \mathbf{0} \end{array}\right]$, with $\mathbf{J}_j$ a sparse $(c \times 6k)$ block matrix, k the

number of bones, and c the total number of DoFs of the joints. The non-zero $(d \times 6)$ blocks in $\mathbf{J}_j$ are defined by each pair of bones connected with a d-DoF joint. Given the joint Jacobians, constraints on the collision-free velocities can be defined as follows:

$$\mathbf{J}_\mu \mathbf{v}^- = -\alpha \mathbf{g}(\mathbf{q}_c) \qquad -\mathbf{J}_j \Delta \mathbf{v}_c = \mathbf{J}_j \mathbf{v}_c + \alpha \mathbf{g}(\mathbf{q}_c). \tag{4.17}$$

In this equation, the term $-\alpha \mathbf{g}(\mathbf{q}_c)$ is a stabilization term to avoid position drift.

By combining the discrete motion equations and the joint constraints (4.17), it is possible to arrange the collision-free velocity update of the articulated skeleton in a large linear system:

$$\begin{array}{cccccc} \mathbf{M}_c & \mathbf{M}_{bs} & -\mathbf{J}_j^T & \Delta \mathbf{v}_c & & \Delta t \mathbf{F}_c \\ \mathbf{M}_{bs}^T & \mathbf{M}_s & \mathbf{0} & \Delta \mathbf{v}_s & = & \Delta t \mathbf{F}_s \\ -\mathbf{J}_j & \mathbf{0} & \mathbf{0} & \mu & & \mathbf{b}_\mu \end{array} \quad , \tag{4.18}$$

$$\text{with } \mathbf{b}_\mu = \mathbf{J}_j \mathbf{v}_c + \alpha \mathbf{g}(\mathbf{q}_c).$$

The system above is sparse, symmetric, and indefinite, with a rather dense band $\mathbf{M}_{bs}$, due to the inertial coupling between the skin and the skeleton. The size of this band, $O(kn)$ with k bones and n surface nodes, can be regarded as a lower bound on the cost for solving the system with direct solvers [BBK05], and the indefiniteness of the system suggests slow convergence of iterative solvers. In Section 4.3.2, I propose a solution combining matrix condensation and an approximation of skin forces that decouples the system and reduces the bilinear complexity.

4.2.3 Contact Constraints

I apply collision response by formulating velocity constraints on colliding surface nodes and solving them through the method of Lagrange multipliers. Instead of simply applying an impulse to the colliding nodes, I formulate the constraints on the implicit motion

equations, which guarantees that collision response effectively acts on the skeletal motion as well. This approach has been demonstrated in Section 3.3. My collision detection algorithm is described in Section 4.3.3. It identifies one contact constraint with normal $\mathbf{n}$ for each colliding surface node. Given the pre-impact velocity $\mathbf{x}_i^-$ of the colliding node, I solve for the node contact response $\delta\mathbf{x}_i$ by imposing a velocity constraint using the kinematic relationships (Eqn. (A.7)):

$$\mathbf{n}^T\left(\mathbf{x}_i^- + \delta\mathbf{x}_i\right) = \mathbf{n}^T\mathbf{L}_W^i\left(\mathbf{v}^- + \delta\mathbf{v}\right) = 0, \tag{4.19}$$

where $\mathbf{L}_W^i$ represents the position-dependent projection matrix $\mathbf{L}_W$ evaluated at node i (see Appendix A.2). It is easy to incorporate moving constraints, friction, and constraint correction. Moving constraints are handled by a velocity offset in the contact constraints in Eqn. (4.19). Similar to the approach of Chapter 3 and also of Bridson et al. [BFA02], friction and constraint correction can be handled for each node separately as a post-process to $\delta\mathbf{v}_s$, and to the resulting post-collision state $\mathbf{q}_s$ respectively.

The vector $\mathbf{j} = \mathbf{n}^T\mathbf{L}_W^i = \begin{bmatrix} \mathbf{n}^T & -\mathbf{n}^T\mathbf{R}\mathbf{u}_i & \mathbf{n}^T\mathbf{R}\mathbf{S}_i \end{bmatrix}$ represents the generalized constraint normal.

The generalized constraint normals can be stacked together in

$$\mathbf{J}_\lambda = \begin{bmatrix} \mathbf{J}_c & \mathbf{J}_s \end{bmatrix} \quad \in \mathbb{R}^{m\times(6k+3n)}, \tag{4.20}$$

where m is the number of colliding surface nodes, k is the number of bones, and n is the total number of surface nodes. The constraint equation is then:

$$\mathbf{J}_\lambda\left(\mathbf{v}^- + \delta\mathbf{v}\right) = 0 \qquad -\mathbf{J}_c\delta\mathbf{v}_c - \mathbf{J}_s\delta\mathbf{v}_s = \mathbf{J}_c\mathbf{v}_c^- + \mathbf{J}_s\mathbf{v}_s^-. \tag{4.21}$$

With constraints formulated through Lagrange multipliers, the complete system of equa-

tions for collision response is:

$$\begin{bmatrix} \mathbf{M}_b & \mathbf{M}_{bs} & -\mathbf{J}_j^T & -\mathbf{J}_c^T & \delta\mathbf{v}_c \\ \mathbf{M}_{bs}^T & \mathbf{M}_s & 0 & -\mathbf{J}_s^T & \delta\mathbf{v}_s \\ -\mathbf{J}_j & 0 & 0 & 0 & \mu \\ -\mathbf{J}_c & -\mathbf{J}_s & 0 & 0 & \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \mathbf{b}_\mu^- \\ \mathbf{b}_\lambda \end{bmatrix}, \tag{4.22}$$

$$\text{with } \mathbf{b}_\mu^- = \mathbf{J}_j\mathbf{v}_c^- + \alpha\mathbf{g}(\mathbf{q}_c^-) \text{ and } \mathbf{b}_\lambda = \mathbf{J}_c\mathbf{v}_c^- + \mathbf{J}_s\mathbf{v}_s^-.$$

The system above can be regarded as an augmented version of Eqn. (4.18), with the addition of contact constraints. This system could be solved by direct application of a method such as constraint anticipation [Bar96]. But, that would yield a computational cost of $O(mkn)$ at best, since the complete system described in Eqn. (4.18) should be solved for each contact. Instead, I propose a solution with a practical $O(m+k+n)$ cost in the next section.

4.3 Condensed Solution of Constraints

Two common constraints need to be resolved in the dynamics simulation of soft articulated characters, namely joint and contact constraints. One of the key contributions of this work is to reduce the best-case $O(mkn)$ complexity of the full solution to contact constraints, while preserving physically plausible global and local deformation effects. I achieve this result by combining Schur complement computation [GL96] (also referred to as matrix condensation [BNC96]), and careful approximations of implicit discretization. First, I present the condensation of skeleton dynamics that allows for $O(k+n)$ update of collision-free dynamics in practice. Then I present the condensation of contact constraints and anticipation of skeleton response that achieves $O(m+k+n)$ update of contact-consistent dynamics in practice.

4.3.1 Condensed Skeleton Dynamics

Joint constraints act only on bone coordinates, not on skin coordinates. This observation is exploited to split the velocity computation in Eqn. (4.18) and Eqn. (4.22), solving first for bone velocities (while accounting for forces on the skin). Computing the Schur complement of the skin inertia $\mathbf{M}_s$ yields a *condensed skeleton inertia* $\mathbf{M}_{\text{cond}}$:

$$\mathbf{M}_{\text{cond}} = \mathbf{M}_c - \mathbf{M}_{bs}\mathbf{M}_s^{-1}\mathbf{M}_{bs}^T \quad \in \mathbb{R}^{6k\times 6k}. \tag{4.23}$$

Unfortunately, computing $\mathbf{M}_{\text{cond}}$ requires solving $6k$ linear systems of size n, and the resulting matrix is dense.

Instead I compute an *approximate* condensed matrix

$$\mathbf{M}_{\text{cond}} = \mathbf{M}_c - \mathbf{M}_{bs}\mathbf{M}_s^{-1}\mathbf{M}_{bs}^T, \tag{4.24}$$

where $\mathbf{M}_s^{-1}$ is a fast approximate inverse of $\mathbf{M}_s$ that accounts only for block-diagonal terms. In this way the computation of $\mathbf{M}_{\text{cond}}$ has an $O(n+k)$ complexity. The approximation $\mathbf{M}_s$ amounts to discarding off-diagonal blocks of $\mathbf{K}$ (i.e., the Jacobians of elastic forces among skin nodes) in the implicit computation of velocities. Note that this approximation does not jeopardize the fulfillment of joint or contact constraints; it simply yields velocities that differ slightly from those of the full solution with Eqn. (4.15). Moreover, the full inverse $\mathbf{M}_s^{-1}$ is still employed in the computation of collision-free skin velocities in Eqn. (4.26). I have quantified the error $\|\mathbf{M}_{\text{cond}} - \mathbf{M}_{\text{cond}}\| / \|\mathbf{M}_{\text{cond}}\|$ (using the spectral norm), and it is below 10% in my simulations, with some variation depending on the average number of bone influences per vertex.

4.3.2 Solving Collision-Free Velocities

Applying condensed skeleton dynamics, the constrained system for bone velocities in Eqn. (4.18) can be rewritten as:

$$\begin{bmatrix} \mathbf{M}_{\text{cond}} & -\mathbf{J}_j^T \\ -\mathbf{J}_j & \mathbf{0} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{v}_c \\ \mu \end{bmatrix} = \begin{bmatrix} \mathbf{b}_c \\ \mathbf{b}_\mu \end{bmatrix}, \tag{4.25}$$

$$\text{with } \mathbf{b}_c = \Delta t \left(\mathbf{F}_c - \mathbf{M}_{bs}\mathbf{M}_s^{-1}\mathbf{F}_s \right) \text{ and } \mathbf{b}_\mu = \mathbf{J}_j\mathbf{v}_c + \alpha\mathbf{g}(\mathbf{q}_c).$$

The structure of Eqn. (4.25) is practically the same as one would obtain when solving a regular articulated body with implicit integration of joint stiffness. Equation (4.25) can be solved in $O(k + c)$ time, with c the number of DoFs constrained by the joints, for articulated structures without loops [Fea87, Bar96]. But $\mathbf{M}_{\text{cond}}$ has off-diagonal non-zero blocks for pairs of connected bones (due to joint stiffness), or for pairs of bones that influence a common skin patch (in the linear-blend skinning scheme). Hence I have opted for a more general indefinite symmetric sparse system solver with fill-reducing reordering [SG06]. Since the sparsity pattern of Eqn. (4.25) is fixed, the reordering and analysis (or symbolic factorization) can be precomputed. This approach reduces the runtime cost of solving this system to be linear in the number of bones and joints in my simulation. Therefore, it is not a bottleneck, as discussed in Section 4.4.

After solving Eqn. (4.25) and computing collision-free bone velocities $\Delta\mathbf{v}_c$, I solve for skin velocities $\Delta\mathbf{v}_s$ in Eqn. (4.18):

$$\mathbf{M}_s\Delta\mathbf{v}_s = \Delta t\mathbf{F}_s - \mathbf{M}_{bs}^T\Delta\mathbf{v}_c. \tag{4.26}$$

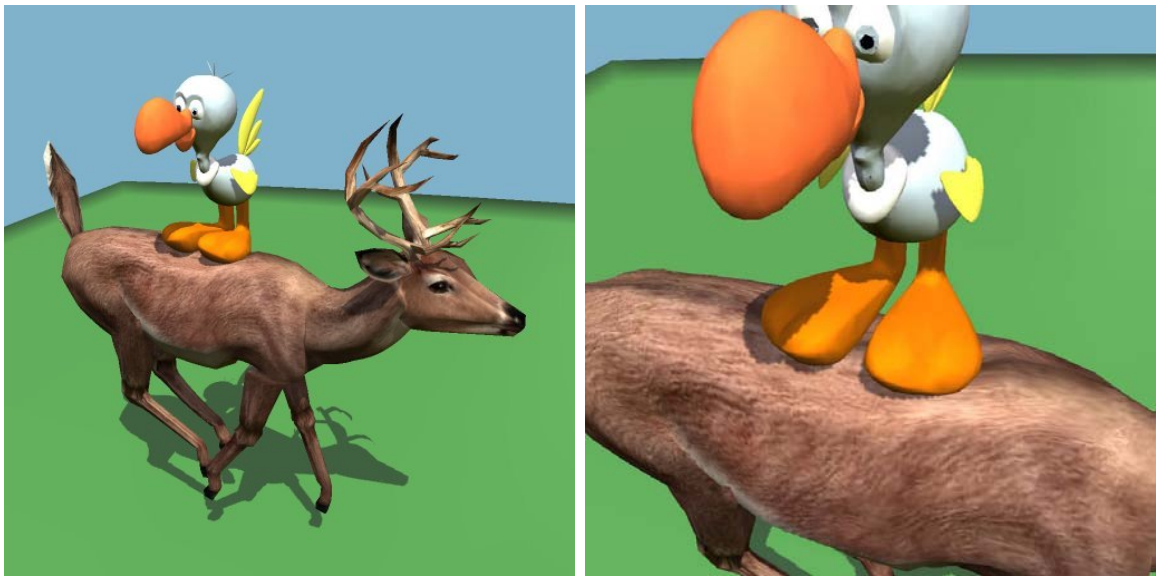
The matrix $\mathbf{M}_s$ is constant and symmetric positive-definite. Also, a fill-reducing sparse factorization of $\mathbf{M}_s$ is computed once for the entire simulation [SG06]. As discussed in Section 4.4, the performance of the solver is linear in my experiments, and better

than a diagonally preconditioned conjugate gradient method. In summary, by approximate condensation of skeleton dynamics, I was able to reduce the brute-force $O(nk)$ complexity to $O(k+n)$ in practice.

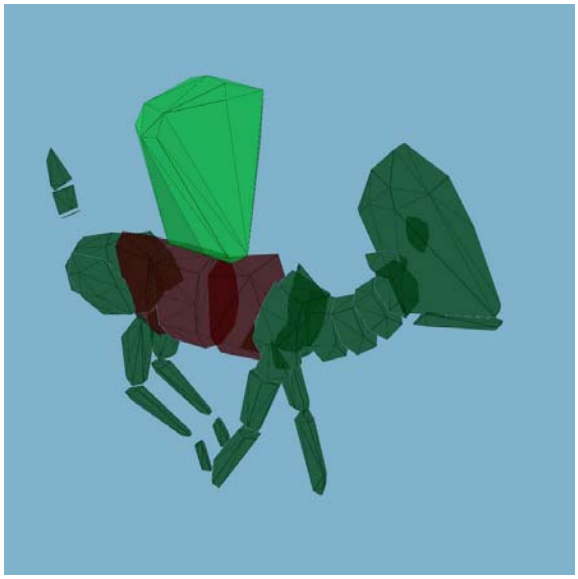
4.3.3 Hierarchical Pruning and Collision Queries

One of the essential components to efficiently solve contact constraints is a fast collision detection module. I extend the fast image-based algorithm that was presented in Section 3.3. In addition, the collision query algorithm for soft *articulated* characters described in this section also performs hierarchical pruning to eliminate large portions of the objects from collision queries, by exploiting the skeletal nature of the deformation. The worst-case cost of collision detection is $O(n)$ for a pair of tested objects with n surface nodes; but the actual cost depends only on the size of the contact area.

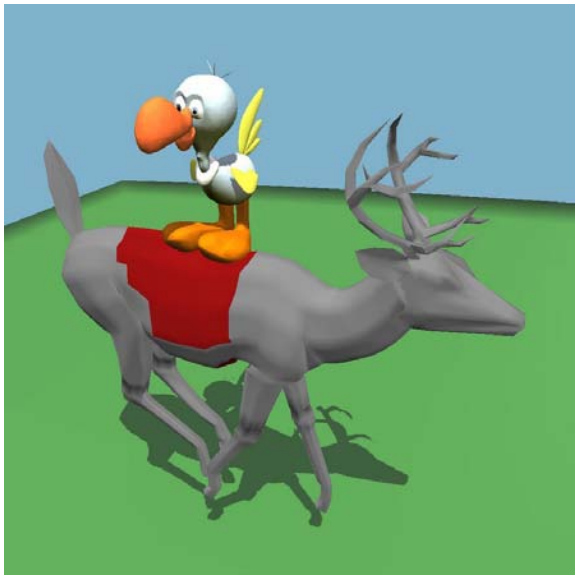
For the object-space collision detection step, I assign to each bone a low-resolution proxy, as shown in Figure 4.3. Specifically, each i^{th} bone is preprocessed to construct a low polygon-count approximate convex hull of the set of skin vertices v_i with blend weight $w_i = 0$. Typically, these convex proxies have a few tens of vertices. At runtime I use the maximum skin displacement $\mathbf{u}_s$ of all vertices associated to a bone to compute a conservative bound of the proxy. I identify potentially colliding bones using a fast collision detection algorithm for convex objects [EL00] that identifies low-resolution proxies within a user specified tolerance distance of each other. This phase, as well as the image-space collision detection phase, is the same as in Chapter 3.3. For articulated characters with many bones, the bone proxies can actually be organized in a bounding volume hierarchy to reduce the number of low-res comparisons. Additionally, the extended method can also handle self-collisions between surface patches of the same articulated body. However, the second step in our collision detection algorithm may report false positives for adjacent bone surface patches, especially for adjacent polygons in the mesh. This case is handled by a post-processing step to check for exact collision



(a) Contact between the bird and the deer, with skin deformations on the back of the deer.



(b) Proxies used for hierarchical pruning of collision queries, with potentially colliding proxies of the deer highlighted in red.



(c) Hierarchical Pruning: only triangles influenced by the potentially colliding bones (in red) are passed to my image-based collision detection algorithm.

Figure 4.3: Layered Representation and Collision Detection.

between adjacent polygons in object space.

4.3.4 Condensed Contact Constraints

When a collision is detected, I apply the same principle of condensation of skeleton dynamics as described in Section 4.3.1 to solve the collision reponse in Equations (4.22). However, this condensation is not sufficient to separate the computation of bone and skin response, since contact constraints act on the bones as well as on the skin, as shown in Eqn. (4.21). The solution is to compute *condensed contact constraints* $\mathbf{J}_{\text{cond}}$ with the approximated $\mathbf{M}_s^{-1}$ from Section 4.3.1:

$$\mathbf{J}_{\text{cond}} = \mathbf{J}_c - \mathbf{J}_s \mathbf{M}_s^{-1} \mathbf{M}_{bs}^T. \tag{4.27}$$

Algebraic manipulation of Eqn. (4.21) and Eqn. (4.22) yields the *condensed constraint equations*:

$$\mathbf{M}_\lambda \lambda + \mathbf{J}_{\text{cond}} \delta\mathbf{v}_c + \mathbf{b}_\lambda = 0, \tag{4.28}$$

$$\text{with } \mathbf{M}_\lambda = \mathbf{J}_s \mathbf{M}_s^{-1} \mathbf{J}_s^T. \tag{4.29}$$

Here, λ is the Lagrange multiplier vector that defines the contact impulses. In order to split the equations, I propose the *anticipation of skeleton response*, i.e., to single out λ:

$$\lambda = -\mathbf{M}_\lambda^{-1} \mathbf{J}_{\text{cond}} \delta\mathbf{v}_c - \mathbf{M}_\lambda^{-1} \mathbf{b}_\lambda. \tag{4.30}$$

Other existing methods for solving multi-body dynamics with joint and contact constraints propose the anticipation of contact constraints (i.e. first singling out skeleton response) and then solving for the contact impulses [Bar96]. However, for the layered soft character representation employed in this thesis, it pays off to exploit the use of equality contact constraints, the fact that each colliding surface node yields one constraint, and the approximation of skin force Jacobians. These techniques together make the matrix $\mathbf{M}_\lambda$ diagonal and trivial to invert. The result is a significant reduction of the

overall computational cost of expensive contact constraint anticipation.

4.3.5 Solving Collision Response

By application of condensed skeleton dynamics, condensed contact constraints, and anticipation of skeleton response, the following system of equations for computing *contact-consistent* articulated body dynamics is obtained:

$$\begin{matrix} \mathbf{M}_c^* & -\mathbf{J}_j^T & \delta\mathbf{v}_c \\ -\mathbf{J}_j & \mathbf{0} & \mu \end{matrix} = \begin{matrix} \mathbf{b}_c^* \\ \mathbf{b}_\mu^- \end{matrix}, \tag{4.31}$$

$$\text{with } \mathbf{M}_c^* = \mathbf{M}_{\text{cond}} + \mathbf{J}_{\text{cond}}^T \mathbf{M}_\lambda^{-1} \mathbf{J}_{\text{cond}} \text{ and } \mathbf{b}_c^* = -\mathbf{J}_{\text{cond}}^T \mathbf{M}_\lambda^{-1} \mathbf{b}_\lambda.$$

The matrix $\mathbf{M}_c^*$ has exactly the same structure as $\mathbf{M}_{\text{cond}}$; therefore, the same solver (and precomputed symbolic factorization) for articulated dynamics as discussed later in Section 4.3.2 can be used. It is remarkable that solving collision response with the proposed Eqn. (4.31) has the same cost as the collision-free solution.

Once the skeletal response $\delta\mathbf{v}_c$ is computed, one obtains the collision impulse λ as discussed in Section 4.3.4, and finally I solve for the skin response in Eqn. (4.22) as:

$$\delta\mathbf{v}_s = \mathbf{M}_s^{-1} \left(\mathbf{J}_s^T \lambda - \mathbf{M}_{bs}^T \delta\mathbf{v}_c \right). \tag{4.32}$$

The approximation of skin force Jacobians largely simplifies the solution of skin response. I have not encountered instabilities in the simulations due to this approximation in my experiments. My observation is that the use of full Jacobians in the collision-free update as described in Eqn. (4.26) guarantees stability, while the coupled response computed on the skeleton ensures the global reaction to collisions.

To summarize, the solution of bone velocities in Eqn. (4.25) and Eqn. (4.31) has a cost $O(k)$, the solution of skin velocities in Eqn. (4.26) and Eqn. (4.32) has a cost

$O(n)$, and the condensation and anticipation steps are sparse matrix multiplications with overall cost $O(m + k + n)$. Altogether, the solution of constrained dynamics for soft articulated characters in my simulation framework has a final cost $O(m + k + n)$. This is also observed in my experiments which are described in Section 4.4.

4.3.6 Run-time Algorithm

Figure 4.4 shows the outline of a single time step in my simulation algorithm. Each step starts with a collision-free update step, followed by contact response.

1. Compute free-motion bone velocities (Eqn. (4.25)).
2. Compute collision-free skin velocities $\mathbf{v}_s^-$ (Eqn. (4.26)).
3. Update collision-free positions.
4. Execute collision detection (See Section 4.3.3).
5. Formulate condensed constraint equations (Eqn. (4.28)).
6. Formulate contact-consistent articulated dynamics (Eqn. (4.31)) to compute skeletal contact response.
7. Compute contact impulses λ (Section 4.3.4) and solve for skin response (Eqn. (4.32)).
8. Update contact-consistent positions.
9. Collision detection and collision-free position reprojection.

Figure 4.4: Summary of the Simulation Algorithm

4.4 Results and Discussion

4.4.1 Benchmarks

My algorithm was tested on a variety of benchmarks, using the soft articulated characters listed in Table 4.1. All the benchmarks were simulated on a 3.4 GHz Pentium-4 processor PC with an NVidia GeForce 7800GTX graphics card. I have performed several experiments with a set of bendable tubes (see Figure 4.5) for validating the behavior with different material stiffness, friction values, number of bones, and moving

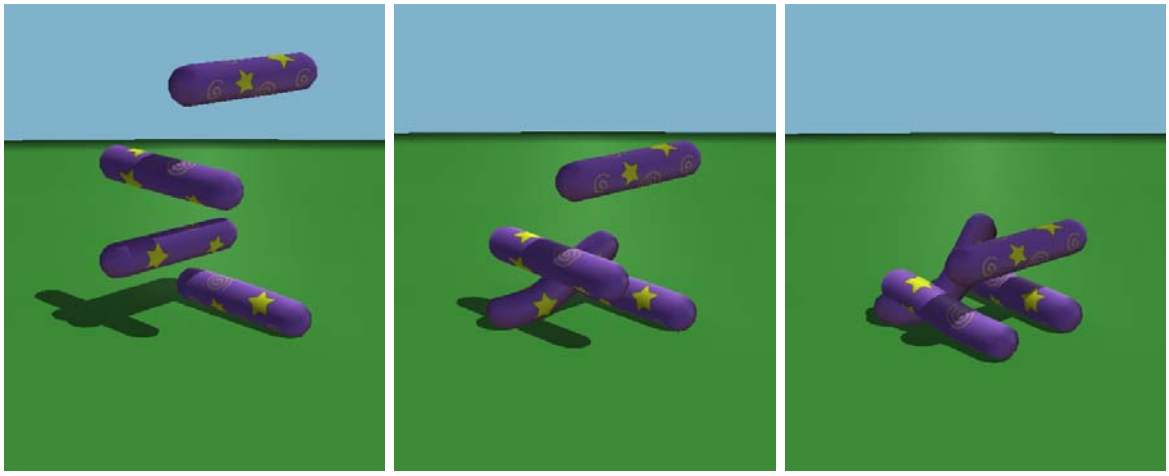


Figure 4.5: Contact between Deformable Tubes with Moving Constraints. The tubes consist of 3 links each and collide with each other in tangled configurations. The described algorithm can handle such situations seamlessly with a combination of local deformations and bone motion at 20 fps.

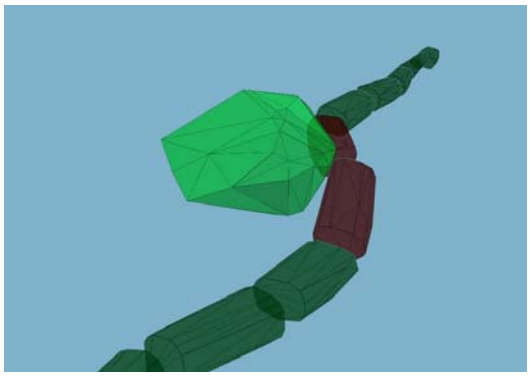
Model	**Nodes** n	**Bones** k	**Coll.** max(m)	**Collision-Free Update**			**Collision Response**			**Total Time**[1]	**Total Time**[2]
				Setup	**Bones**	**Skin**	**Setup**	**Bones**	**Skin**		
Deer	1 748	34	13	19.7	34.0	17.9	1.1	15.8	7.7	74.8	123.8
Deer	2 755	34	41	39.3	36.6	29.6	11.3	14.7	9.9	114.1	160.4
Deer	8 408	34	162	127.0	64.9	96.2	24.5	17.8	33.9	310.0	449.7
Snake	3 102	16	28	38.9	18.1	36.5	6.2	2.2	12.9	106.5	140.6
Tube	292	2	73	2.5	1.6	1.9	3.9	0.2	0.8	9.5	13.8
Tube	292	5	74	3.7	2.7	2.1	7.7	0.6	1.0	14.6	23.3

Table 4.1: Benchmark Statistics. The soft characters for the benchmarks are a deer model with three different skin resolutions, a snake model, and tubes with different numbers of bones. All timings (in msec.) are averages over the course of a simulation. The last two columns indicate the average time per frame in (1) non-colliding and (2) colliding situations.

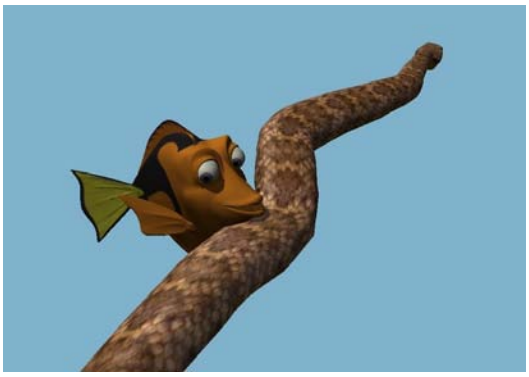
constraints. The deer model in Figure 4.1 was pre-animated and driven by applying additional control forces on its bones.

Table 4.1 also shows a breakdown of the average timings per frame, highlighting the time for collision-free dynamics update, collision detection, and collision response. The last two columns show the average total time per frame, for (1) non-colliding situations and (2) colliding situations. Note that, for the benchmark of the snake (16 bones and 3 102 skin nodes, shown in Figure 4.6), the simulation runs at 7 fps with collisions, and 10 fps when there are no collisions. For the benchmark of the deer (34 bones and 2 755

(a) A snake with 16 bones and 3102 surface vertices.



(b) Proxy geometry for the first phase of collision detection.



(c) Simulation sequence with a fish touching the snake, showing the global deformation of the snake.

Figure 4.6: Skeletal Deformations of a Soft Snake.

skin nodes), the simulation is also interactive (as shown in Figure 4.1) in the range of $6 - 9$ fps. For one tube with 5 bones, the simulation runs at 43 fps even with large colliding areas. The example shown in Figure 4.5 runs at $15 - 20$ fps.

For the deer model, the simulation was performed with different skin resolutions. As shown in the table, the simulation cost varies linearly w.r.t. the resolution of the skin n. Furthermore, notice how, for the two lower-resolution skins, the cost for the collision-free velocity update of the bones remains almost unchanged and independent of the skin resolution. This data shows that the cost at those resolutions is dominated by the number of bones and, more importantly, that the proposed method for solving

bone and skin velocities does not have a bilinear cost of $O(kn)$. Similarly, the number of collisions for the mid-resolution deer varies from 1 to 41, while the time for collision response on the bones remains practically unchanged during the simulation. This data again shows that the cost at low resolutions is dominated by the number of bones and is not bilinear $O(km)$ w.r.t. the number of contacts. Combining the observations for collision-free updates and collision response, it can be concluded that the simulations have a runtime complexity of $O(m+k+n)$ in practice. It is interesting to notice that, for the deer model with the highest resolution mesh, the dominating cost is the initialization of matrices for the computations, not the solution of the constrained systems.

4.4.2 Comparisons and Limitations

I have also compared the performance of the sparse system solver [SG06] with preconditioned conjugate gradient descent (PCG) to solve (4.26), on the deer model with 2 755 surface nodes. I used a diagonal preconditioner consisting of the diagonal part of $\mathbf{M}_s$. PCG is 4 times slower for solving the collision-free update of the skin, even after reaching 100 iterations without fully converging.

As demonstrated in the experiments, my method for simulating soft articulated characters handles contact constraints interactively while producing rich deformations on the skin. Due to its layered representation, it cannot be directly compared with methods that model global deformations using a volumetric representation. From the family of FEM-based methods, the one by Müller et al. [MDM$^+$02] is perhaps the closest relative to mine, as it also uses implicit integration and linear elasticity evaluated in a floating reference frame. In comparison with Müller's, my method is obviously restricted to skeletal global deformations, not arbitrary ones, and the elastic energy only accounts for skin-layer deformations, not deformations in the whole volume of the object. However, our method offers considerable benefits for fast collision handling. Due to the fast solver presented in Section 4.3, my method can efficiently compute global collision response

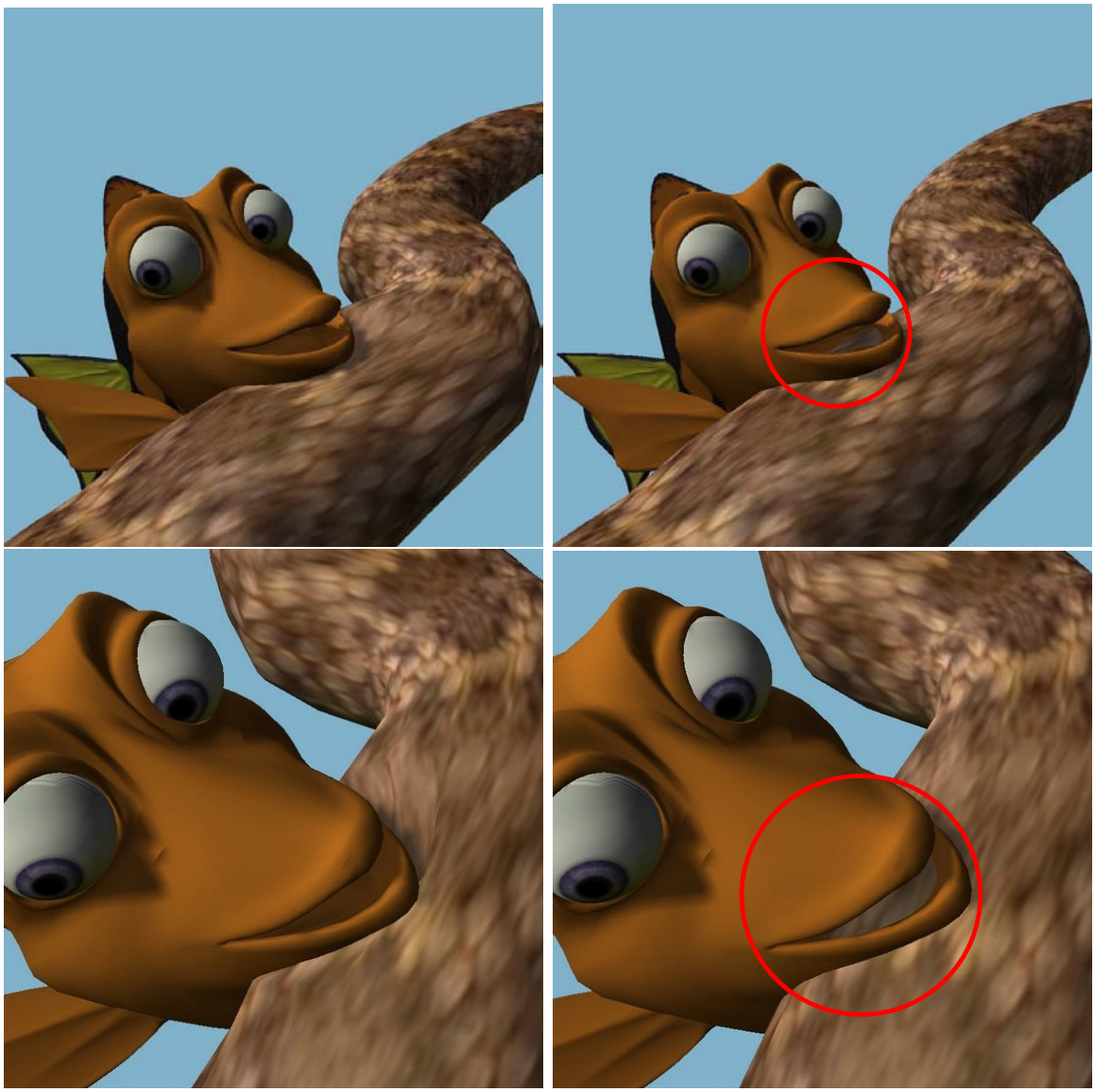


Figure 4.7: Contact Constraints. Left column: The fish touches the body of the snake, creating global response and skin deformations. Right column: Local skin deformations are turned off to show the importance of handling both global and surface response. Notice the highlighted interpenetrations, clearly visible through the fish's mouth.

(i.e., response of the skeleton) using matrix condensation. Furthermore, it can handle both local skin and global skeletal response with approximate implicit integration stably and robustly. Müller's method (and others) cannot exploit the decomposition of the deformation, and the constraint-based simulation would require the use of the full implicit system, in order to robustly compute global response. Perhaps for this reason,

methods such as Müller's are often combined with penalty-based collision response, not constraint-based, with the associated problem of object interpenetration as shown in Fig. 4.7.

4.5 Advantages and Summary

The method presented in this chapter has the following key advantages:

- A new formulation of elastic deformation in pose space, which is related to skin displacement corrections [KJP02, JT05] and FEM approaches in a floating frame of reference [TW88], augmented with a joint stiffness term to model pose-dependent deformations. With this formulation, the motion equations derived from Lagrangian mechanics naturally produce the desired *interplay between skin and skeleton.*

- *Efficient and scalable computation* of articulated-body dynamics with contact constraints and skin deformations, with a cost of $O(m + k + n)$ in practice, where m is the number of contacts, k the number of bones, and n the number of surface nodes. My method is based on the decoupling of skin and skeleton computations in otherwise coupled implicit equations, through careful and robust approximation of Schur complements.

- The presented model for dynamic soft articulated characters builds upon traditional mesh skinning paradigms and enables *easy integration* with existing skinning pipelines.

I have presented a novel method for simulating deformable characters with an articulated skeleton that allows fast handling of complex contact scenarios and plausible, coupled global and local deformations. This method models characters as skinned articulated bodies with a layered representation, and measures elastic deformations in pose

space. Central to the efficient simulation of contact-induced deformations is an implicit constraint-based collision handling approach that exploits the layered representation to enable efficient, approximate yet robust matrix condensation.

I have implemented my algorithm and tested its performance on several complex benchmarks. I was able to achieve simulation frame rates of 4 to 8 fps with multiple colliding articulated characters with a deformable skin layer, each consisting of up to $3,500$ vertices.

4.6 Limitations and Future Work

The deformation model described in this chapter expresses strain in pose space and does not capture pose-dependent strain near joints, as explained in more detail in Section 4.2.1. I have approximated pose-dependent strain energy with a user-tunable joint stiffness, but my model could be further extended with data-driven approaches [LCF00] to add e.g., bulging effects due to elbow flexion. In fact, such effects become possible with the method presented in Chapter 5, in which I present a method that complements pose-dependent strain-energy with artist-provided example shapes. Another possible extension for handling more complex volumetric deformations would be to adopt a deformation model based on a control lattice [CGC+02a], instead of a purely skeletal approach. This would require an extension of the contact handling algorithm so that collision response can be efficiently applied to both the control lattice and the skeleton. Similarly, it is worth exploring the addition of our efficient contact-induced deformations to non-skeletal skinning approaches [JT05].

Additionally, it would be interesting to explore ways to handle inequality constraints, as this would allow modeling more accurate contact forces and joint limits.

Finally, in the interest of performance it would be useful to investigate a fully parallelized solution to enable implementations that are accelerated for parallel hardware

architectures. This would open the door to more complex scenes with many deforming characters at interactive rates.

Chapter 5

Deformation Control with Dynamic Morph Targets

In this chapter, I present a method to control the behavior of elastic, deformable material in a dynamic simulation. I introduce *dynamic morph targets*, the equivalent in dynamic simulation to the geometric morph targets in (quasi-)static modeling, as a step towards bridging the gap between geometric example-based methods and physically based approaches. This chapter is a logical continuation of the development of a controllable elastic deformation model that was started in the previous chapters. The elastic deformation model presented in Chapters 3 and 4 for objects with a (possibly articulated) core and a layer of soft skin is leveraged in this chapter. Dynamic morph targets merely influence the coefficients of the dynamic equations and hence the discretization, integration and solution techniques that were presented in the previous chapters for simulation of combined global and local deformations can be employed in this chapter as well.

Dynamic morph targets define the pose-dependent physical state of soft objects, including surface deformation and elastic and inertial properties. My method easily integrates with current modeling and animation pipelines: at different poses, an artist simply provides a set of dynamic morph targets.

The deformable models presented in this chapter are computationally efficient at runtime through modal reduction and pose-space polynomial interpolation. These models

can therefore be plugged into existing dynamic simulation engines, either forming interactive, deformable content in real-time games or providing secondary dynamic effects for kinematically-driven characters in feature animation films.

5.1 Method

The goal of the method in this chapter is to simulate *controllable* non-linear deformations by interpolation of *dynamic morph targets* at runtime, kindred to geometric morph targets in static character modeling. In this section, I describe the concept of dynamic morph targets, and explain how they can be used to simulate a pose-dependent elastic model that is fast enough for real-time applications.

Dynamic morph targets rely on three key components:

- A pose-space method for interpolation of efficient elastic deformation models that allows the artist to author complex nonlinear deformation behavior (Sections 5.1.1 and 5.2).
- A compact way of interpolating skin geometry, elastic forces, and their derivatives, all in a unified manner using polynomial interpolation (Sections 5.1.2 and 5.1.3).
- The extension of the method to support modal reduction and therefore very efficient implementation that is linear in the number of coefficients of the force polynomial (Section 5.1.4).

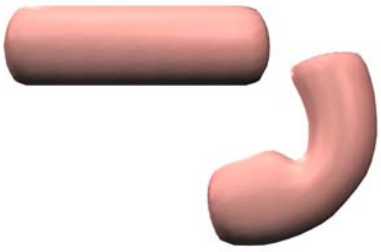
5.1.1 Dynamic Morph Targets

I define a `dynamic morph target' as an elastic deformation model at a certain skeletal pose of the character. More formally, dynamic morph targets are pairs of elastic models E_i and poses $\mathbf{s}_i$, i.e. they are pairs $E_i, \mathbf{s}_i$ of elastic models in pose space. Similar to

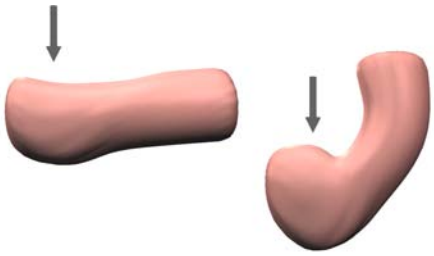
geometric morph targets, dynamic morph targets associate surface and volume deformation with character pose. In contrast to geometric morph targets, dynamic morph targets also define elastic properties at specific character poses. Elastic properties include stiffness and plasticity parameters. The combination of surface deformation and elastic properties defines the elastic model E_i. A pose is represented by a vector $\mathbf{s} \in \mathcal{S}$ where pose-space $\mathcal{S} \subset \mathbb{R}^k$. Note that the implementation in this dissertation uses skeletal pose, but the concept of pose can easily be extended beyond the skeletal sense; in fact any notion of state of a character can be used, such as emotional state, velocity state, contact state, or muscle activation.

Dynamic morph targets can easily be created in existing modeling packages. At the level of content creation, they are very similar to the creation of geometric morph targets. The modeler makes a set of m poses $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_m$ of the character and sculpts desired deformations that cannot be captured with traditional skinning methods [KZ05, MTLT88]. As shown in Fig. 5.1 and Fig. 5.2, the morph targets are defined in the skeletal bind pose, complying with most common modeling packages. In addition to surface modeling, the modeler uses vertex painting to assign pose-specific elastic properties to the dynamic morph targets directly from the modeling software. A modeler can therefore choose to make *the same* skin section stiff for one pose and flabby for another pose, e.g. to mimic contraction and relaxation of a muscle, or to exaggerate skin bulging.

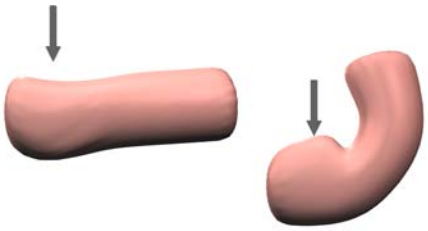
Once created, the dynamic morph targets are then fed through a preprocessing pipeline as described later in Section 5.2. The result is a fully dynamic character that can react to external forces just like other common deformable model methods. However, these characters also expose the non-linear deformations and elastic behavior as defined by the dynamic morph targets.



(a) Morph targets



(b) Simulation with all soft morph targets



(c) Simulation with stiffer bulged morph target

Figure 5.1: Concept of Dynamic Morph Targets: Simple cylinder geometry mimicking an elbow joint with bulging skin, for which two morph targets (out of a total of four) are given in (a). The skin of the bulged morph target was chosen to be stiffer to mimick muscle contraction. On the right, we show runtime snapshots of simulations using our pose-dependent elastic model, under influence of identical downward forces. (b) was generated with four **soft** morph targets, whereas (c) has **increasingly stiffer** morph targets, to mimic muscle contraction. The dynamic skin behavior is identical for the straight joint (a relaxed muscle), because the elastic properties of the first morph target are identical for (b) and (c). But, for the bent joint, the force clearly causes more skin deformation in (b). This undesirable behavior can be fixed to mimic muscle contraction by making the fourth morph target stiffer, as shown in in (c). This simple example shows a dynamic bulging effect that can only be achieved with dynamic morph targets.

5.1.2 Pose-dependent Elastic Model

Dynamic morph targets are used to build a pose-dependent elastic model $E(\mathbf{x}, \mathbf{s})$. For hyper-elastic materials, an elastic model can be defined as a material function $E(\mathbf{u}(\mathbf{x}))$ defining the internal elastic energy at material points $\mathbf{x}$ in function of the deformation $\mathbf{u}$. For my experiments and in correspondence with the space in which the morph targets

are defined, I choose to express elastic deformation in the skeletal bind pose as has been proposed in the past [LCF00, KJP02]. This is identical to the pose-space strain formulation presented in Chapter 4.1.1. On the other hand, it is certainly possible to use other formulations of elastic strain to define a pose-dependent model with dynamic morph targets. Traditionally, the elastic energy is a *pose-independent* material potential that causes internal elastic forces $R(\mathbf{u})$ in the material. I propose a *pose-dependent* elastic model by taking into account the dynamic morph targets $E_i, \mathbf{s}_i$ as example inputs. I use scattered data interpolation to derive an expression for the internal elastic forces $R(\mathbf{u}, \mathbf{s})$ anywhere in pose-space $\mathcal{S}$, given the expressions for the elastic forces $R_i(\mathbf{u})$ that are imposed by the dynamic morph targets at poses $\mathbf{s}_i$.

Equations of motion Without loss of generality, the finite element method is applied to discretize the partial differential equations of solid continuum mechanics in space, leading to the common Euler-Lagrange motion equations, where I substitute the pose-dependent elastic forces $\mathbf{R}(\mathbf{u}, \mathbf{s})$:

$$\mathbf{M}\mathbf{u} + \mathbf{D}(\mathbf{u}, \mathbf{u}, \mathbf{s}) + \mathbf{R}(\mathbf{u}, \mathbf{s}) = \mathbf{f}, \tag{5.1}$$

with $\mathbf{M}$ the mass matrix and $\mathbf{f}$ the external forces and the (local) Raleigh damping model $\mathbf{D}(\mathbf{u}, \mathbf{u}, \mathbf{s})$:

$$\mathbf{D}(\mathbf{u}, \mathbf{u}, \mathbf{s}) = \left(\alpha\mathbf{M} + \beta \frac{R(\mathbf{u}, \mathbf{s})}{\mathbf{u}}) \right) \mathbf{u}.$$

Polynomial elastic forces The computation of pose-dependent elastic forces $\mathbf{R}(\mathbf{u}, \mathbf{s})$ requires the interpolation of pose-specific forces $\mathbf{R}_i(\mathbf{u})$. However, since forces are a function of the time-varying deformation $\mathbf{u}$, they cannot simply be evaluated once and then interpolated at runtime. I have opted for elastic models for which $\mathbf{R}_i(\mathbf{u})$ can be expressed as a (multivariate) polynomial function of the degrees of freedom $\mathbf{u}$. Then, the interpolation of elastic models reduces to the interpolation of polynomial coefficients.

Common examples of such elastic models are the so-called `completely linear' FEM deformation model (with or without stiffness warping [MG04]), or the 'semi-non-linear' St.Venant-Kirchoff model (StVK) [BJ05, CBC+05]. I have simulated example characters with both linear and semi-non-linear elastic models. Interestingly, because deformation is expressed in the skeletal bind pose in my system, there was no noticeable quality difference between both elastic models in our experiments. Therefore, I have opted for the more efficient linear elasticity model to produce most of the images in this dissertation, unless otherwise noted (see Section 5.4). In Section 5.1.4 it is shown that both models are amenable to modal reduction for efficiency.

Each polynomial $\mathbf{R}_i(\mathbf{u})$ is associated with a dynamic morph target at pose $\mathbf{s}_i$ and is uniquely defined by its set of coefficients $a_{k\ i}$ which are collected in a vector $\mathbf{a}_i$. One can then determine the pose-dependent elastic force $\mathbf{R}(\mathbf{u}, \mathbf{s})$, which is also uniquely defined by its set of coefficients $\mathbf{a}(\mathbf{s})$. At an arbitrary pose $\mathbf{s}$ in pose-space, $\mathbf{a}(\mathbf{s})$ can be interpolated from the example coefficients $\mathbf{a}_i$. This is described in more detail in Section 5.1.3.

The interpolation of polynomial coefficients yields the interpolation of force values $\mathbf{R}(\mathbf{u}, \mathbf{s})$ for all possible deformation values $\mathbf{u}$. However, it also yields the interpolation of force derivatives, such as the stiffness matrix $\frac{\mathbf{R}(\mathbf{u}\,\mathbf{s})}{\mathbf{u}}$. One subtle detail remains for defining a complete interpolation of elastic models. The rest configuration at each input pose may be different, therefore the deformation $\mathbf{u}$ may not be consistent across poses. I first choose a certain pose as a reference, and express the deformation of all other poses by adding the difference between rest configurations, $\Delta\mathbf{u}$. The addition of this term simply modifies the coefficients of the force polynomials, which can then be safely interpolated, as all deformations are now expressed with respect to a consistent configuration.

5.1.3 Interpolating Force Polynomials in Pose Space

Conceptually, a *pose* can be described in many ways, as long as it provides a description of the state of a model or character, and a *metric* to measure distances between poses. In my implementation, I choose the joint configuration of an articulated character as the pose descriptor. I define the pose descriptor $\mathbf{s} \in \mathbb{R}^{6(k)}$, where k is the number of joints. Each joint contributes 6 components to $\mathbf{s}$, namely the 6-DoF representation of its parent-relative coordinate frame. The first three components represent the local joint translation, whereas the last three components represent the local joint rotation. The distance between two poses is defined to be the inner product of the difference of its descriptors.

As an articulated character moves between observed configurations, its elastic model should approximate the elastic models of the input poses. As mentioned in Section 5.1.2, the goal is to find a way to interpolate internal elastic forces $\mathbf{R}_i$. Obviously, as the character moves from one pose to the other, the internal forces change continuously but highly non-linearly. In other words, elastic forces form a non-linear smooth field in pose-space. Radial base functions [Pow87] (RBF) are a common choice for interpolating scattered data of an underlying smooth field that is non-linear. Moreover, as our goal is to have as few input models E_i as possible, RBFs are suited because they work well with sparse input data sets. RBFs extend easily to high dimensional domains, enabling the capture of multi-joint coupling effects.

As mentioned in Section 5.1.2, it is possible to determine the pose-dependent elastic forces $\mathbf{R}(\mathbf{u}, \mathbf{s})$ by computing the polynomial coefficient vector $\mathbf{a}(\mathbf{s})$. Using RBFs, the interpolated coefficient vector is computed at runtime as

$$\mathbf{a}(\mathbf{s}) = \sum_{j=1}^{m} \mathbf{w}_j \phi(\ \|\mathbf{s} - \mathbf{s}_j\|\ ) + \mathbf{Q}(\mathbf{s}) \tag{5.2}$$

where m is the number of dynamic morph targets. In my experiments, it was sufficient

to use a constant for the polynomial $\mathbf{Q}(\mathbf{s})$. I also employed the globally supported biharmonic RBF kernel $\phi(r) = r$, since its optimal fairness allows for smoother interpolation of sparsely scattered example poses as compared to locally supported kernels [CBC+01]. In our experience, locally supported kernels such as the Gaussian RBF kernel are harder to tune and are unable to extrapolate across dynamic morph targets that are far apart in pose-space. The smoothing term from [CBC+01] is also employed to achieve smoother behavior across large gaps between input poses.

The RBF weight vectors $\mathbf{w}_j$ are given by the solution of m linear systems (one for each input pose i):

$$\mathbf{a}(\mathbf{s}_i) = \mathbf{a}_i = \sum_{j=1}^{m} \mathbf{w}_j \phi(\ \|\mathbf{s}_i - \mathbf{s}_j\|\ ) + \mathbf{Q}(\mathbf{s}_i). \tag{5.3}$$

The combination of these m linear systems provides m vectorial equations with m vector unknowns $\mathbf{w}_j$, plus the additional constant unknown vector $\mathbf{Q}$ (in case of choosing a constant polynomial). The combined system is underdetermined (m+1 vector unknowns for m vectorial equations), but it can be solved by imposing orthogonality conditions on the weights $\mathbf{w}_i$ [CBC+01]. In order to avoid redundancy in the pose descriptors s_i, and to guarantee that Eqn. (5.3) is not singular, I perform a principal component analysis [GL96] on the set of input pose descriptors. By selecting modes with non-zero or large eigenvalues only, I reduce the dimension of $\mathbf{s}$ and define a mapping to the reduced pose descriptor $\bar{\mathbf{s}} = \mathbf{U}_s^T\mathbf{s}$. The pose descriptor $\mathbf{s}$ is replaced by $\bar{\mathbf{s}}$ both in the preprocessing stage to solve Eqn. (5.3) and at runtime for Eqn. (5.2). Modal reduction of the pose descriptors is very effective for robustness, but is also useful when our method is used for facilitating rigging. In highly complex areas of skin deformation such as the shoulder area, the skin is under influence of many bones for which the skin-bone relationships cannot easily be determined by a human rigger or technical director. My system can automatically deduce these relationships and reduce them to only a few

significant modes.

At runtime, given $\mathbf{a}(\mathbf{s})$ from Eqn. (5.2), I can compute the elastic forces $\mathbf{R}(\mathbf{u}, \mathbf{s})$ and their Jacobian $\frac{\mathbf{R}(\mathbf{u}\,\mathbf{s})}{\mathbf{u}}$ for implicit integration of Eqn. (5.1). Unfortunately, due to the large number of coefficients that need to be interpolated, the evaluation of Eqn. (5.2) is rather costly. The number of coefficients is proportional to the number of nodes n in the finite element mesh for linear elastic models, and $O(S^3 n)$ for StVK materials, where S is the average size of the neighborhood of a node). Instead, I propose a way to increase performance and to reduce the dependency on the resolution of the input geometry by reducing the number of degrees of freedom, while still maintaining the non-linear behavior defined by the morph targets.

5.1.4 Reduced Equations of Motion

I use a reduced model $\mathbf{u} = \mathbf{U}\mathbf{q}$ to enable dynamic simulation that is independent from the input resolution of the geometry. The ODE, Eqn. (5.1), is transformed into

$$\mathbf{q} + \mathbf{D}(\mathbf{q}, \mathbf{q}, \mathbf{s}) + \mathbf{R}(\mathbf{q}, \mathbf{s}) = \mathbf{f} \tag{5.4}$$

where $\mathbf{D}$, $\mathbf{R}$ and $\mathbf{f}$ are r-dimensional reduced forces,

$$\mathbf{D}(\mathbf{q}, \mathbf{q}, \mathbf{s}) = \mathbf{U}^T \mathbf{D}(\mathbf{U}\mathbf{q}, \mathbf{U}\mathbf{q}, \mathbf{s}) \tag{5.5a}$$

$$\mathbf{R}(\mathbf{q}, \mathbf{s}) = \mathbf{U}^T \mathbf{R}(\mathbf{U}\mathbf{q}, \mathbf{s}) \tag{5.5b}$$

$$\mathbf{f} = \mathbf{U}^T \mathbf{f}. \tag{5.5c}$$

Similarly, one can form the *dense* reduced tangent stiffness matrix,

$$\frac{\mathbf{R}(\mathbf{q}, \mathbf{s})}{\mathbf{q}} = \mathbf{U}^T \frac{\mathbf{R}(\mathbf{U}\mathbf{q}, \mathbf{s})}{\mathbf{q}} \mathbf{U} \in \mathbb{R}^{(r\,r)} \tag{5.6}$$

When applying model reduction to (multivariate) polynomial elastic forces, it can be shown that the reduced forces are still (multivariate) polynomial elastic forces. In particular, reduced `completely-linear' elastic forces are linear polynomials in terms of the reduced coordinates $\mathbf{q}$. Additionally, Barbic *et al.* [BJ05] have shown that reduced StVK internal forces and tangent stiffness matrices are multivariate cubic polynomials that can be evaluated in $\Theta(r^4)$ time, with r the number of reduced modes (typically 10-30), by simply evaluating polynomials in terms of the reduced coordinates $\mathbf{q}$:

$$\mathbf{R}(\mathbf{q}) = \mathbf{P}^i q_i + \mathbf{Q}^{ij} q_i q_j + \mathbf{S}^{ijk} q_i q_j q_k \tag{5.7a}$$

$$\frac{\mathbf{R}(\mathbf{q})}{q_l} = \mathbf{P}^l + (\mathbf{Q}^{li} + \mathbf{Q}^{il}) q_i + (\mathbf{S}^{lij} + \mathbf{S}^{ilj} + \mathbf{S}^{ijl}) q_i q_j, \tag{5.7b}$$

Note that Einstein summation convention was used in Eqn. (5.7). Here, $\mathbf{P}^i, \mathbf{Q}^{ij}, \mathbf{S}^{ijk} \in \mathbb{R}^r$ are constant vector polynomial coefficients. The polynomial coefficients can be precomputed, given the rest pose $\mathbf{s}_0$. For StVK materials, the algorithm is described in [BJ05]. As described at the end of Section 5.1.2, these polynomials must be expressed with respect to the common rest pose p_0. The details of finding the correct transformation are described in Appendix D. For linear materials, the $\mathbf{Q}^{ij}$ and $\mathbf{S}^{ijk}$ terms are all zero and the transformation is trivial.

One can now combine scattered polynomial interpolation from Section 5.1.3 with the reduced motion equations by concatenating the reduced coefficients into $\mathbf{a}$:

$$\mathbf{a} = [\ \mathbf{P}^i;\quad \mathbf{Q}^{ij};\quad \mathbf{S}^{ijk}\ ]. \tag{5.8}$$

Just as in Section 5.1.3, each dynamic morph target defines a set of coefficients $\mathbf{a}_i$ which can then be used to set up an interpolator for the pose-dependent coefficients $\mathbf{a}(\mathbf{s})$. This then yields all the necessary information to compute $\mathbf{R}(\mathbf{q}, \mathbf{s})$ in Eqn. (5.5). Note that, because the number of reduced modes r is typically many orders of magnitude smaller than the number of vertices of the mesh, the cost of evaluating Eqn. (5.2) is significantly

smaller than in the non-reduced case.

The construction of the reduction basis **U** will be discussed in Section 5.3. The reduced equations of motion Eqn. (5.4) can be solved using a reduced implicit Newmark Solver, employing the aforementioned internal forces and tangent stiffness matrices evaluated at each time step.

5.2 Model Construction and Kinematic Constraints

My deformation control framework integrates well with existing content creation pipelines provided by a wide range of popular modeling software. Dynamic morph targets are automatically generated from traditional geometric morph targets, and skeletal animation is supported at runtime through kinematic constraints.

Modeling As described in Section 5.1.1, an artist begins by modeling the base model surface and a skeleton with associated SSD skinning weights. He also defines a set of geometric morph targets *in the skeletal bind pose* of the model (on the left in Figures 5.2 and 5.1). Using vertex painting, he can then assign stiffness parameters such as Young's modulus and Poisson ratio to certain parts of the skin. This is where the preprocessing stage starts.

Preprocessing First, the base mesh is tetrahedralized and the surface node SSD skinning weights are propagated to internal nodes. This is done by solving a homogeneous Poisson problem for the internal node weights, where the known surface node weights are set up as boundary conditions. Then, for each morph target, a corresponding tetrahedral rest-pose mesh is defined (still in the skeletal bind pose). This can be done by displacing the surface nodes of the base tetrahedral mesh with the morph target's values. I then relax the internal nodes by performing a physical soft-body simulation, constraining the new surface positions and using the elastic model of the base mesh.

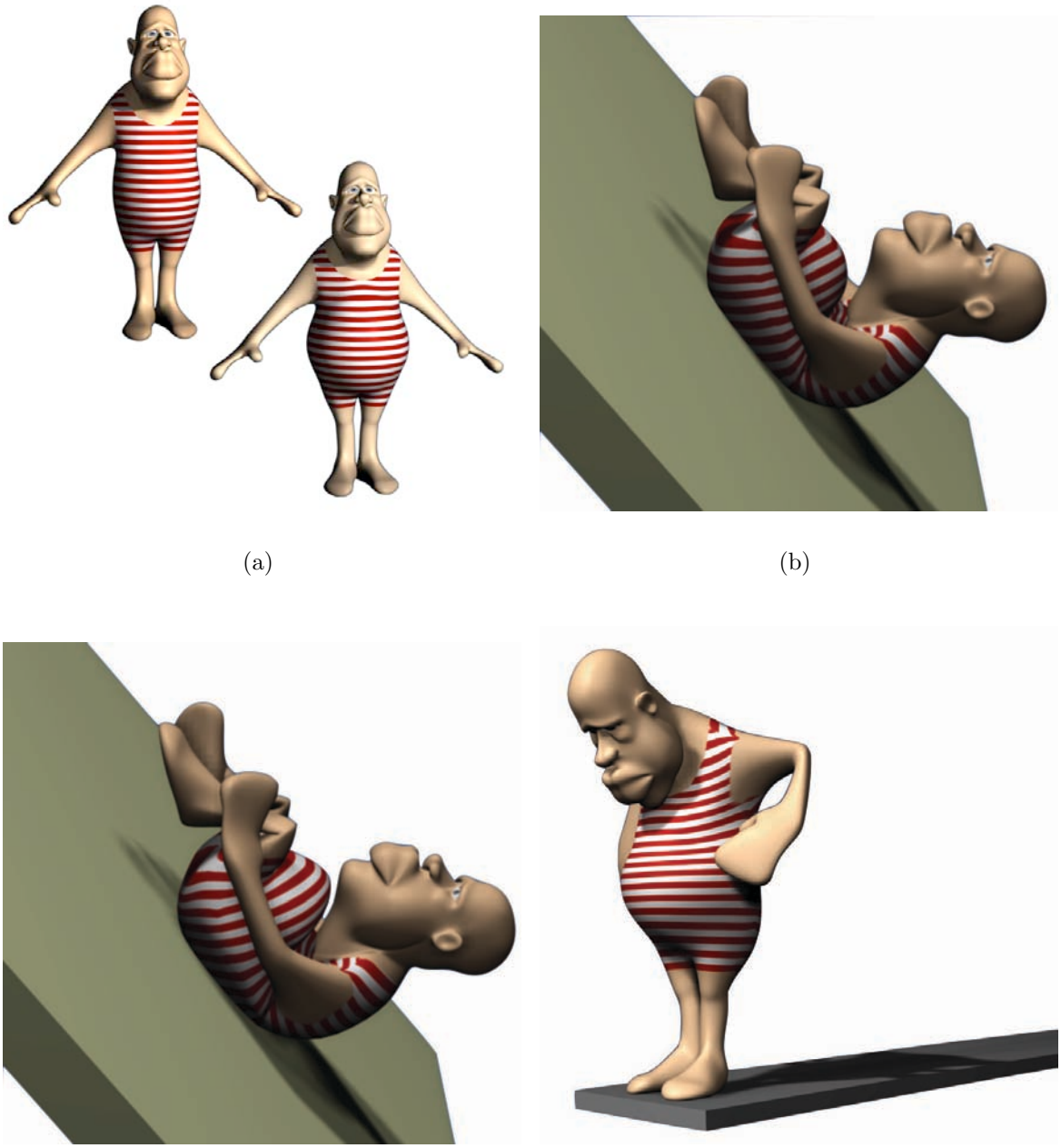
Once the new internal node positions are determined, the morph target's position offsets $\Delta\mathbf{u}$ from the base mesh are computed and the force polynomials $\mathbf{a}_i$ associated with $\mathbf{R}_i(\mathbf{u})$ are precomputed (see Eqn. (5.3), Sections 5.1.2 and 5.1.4). For reduced elastic models, a modal subspace is also constructed (Section 5.3).

Runtime skeletal constraints At runtime, the final positions of the material points are computed as the combination of linear blend skinning and elastic deformation computed in the skeletal bind pose. Inertial effects that are caused by the moving coordinate frames of the bones are accounted for by applying condensation techniques, in the same way as in Sections 3.2.2 and in 4.3.2 (Eqn. 4.26). Additionally, in order to emulate that skin is attached to bones, the material points that are attached to internal bones are constrained, at least conceptually. This can be achieved by removing the elastic degrees of freedom that are associated with corresponding internal mesh nodes. Hence, the positions of these points are then completely governed by the linear blend skinning transformations only. These degrees of freedom are removed a pre-process by identifying tetrahedral mesh elements that are intersected by skeletal bones. Degrees of freedom that are associated with these elements are removed (i.e. they are `fixed' in the pose-space), unless they lie on the model's boundary surface.

5.3 Reduced Modal Subspace Construction

In order to use a reduced model with dynamic morph targets, as described in Section 5.1.4, one needs to choose an appropriate reduced subspace. In this section, I describe what entails a `good' subspace, and I propose to construct a subspace that is *aware* of the morph targets.

In the reduced model, the displacement vector $\mathbf{u}$ is expressed as $\mathbf{u} = \mathbf{U}\mathbf{q}$, where $\mathbf{U} \in \mathbb{R}^{(3n\,r)}$ is the *displacement basis matrix*, and $\mathbf{q} \in \mathbb{R}^r$ is the vector of reduced displacement coordinates. Here, $\mathbf{U}$ is a time-independent matrix specifying a basis of



(a) (b)

(c) (d)

Figure 5.2: When under influence of dynamic events such as jumping from a diving board or bouncing off a wall, our method using morph targets produces deformations consistent with Herbert's morph targets defined in (a). The *fat* morph target is associated with a crunched pose to mimic a bulging belly (and only for that pose). As shown in (b), the simulation without dynamic morph targets does not show bulging, whereas our method shown in (c) does.

some r-dimensional ($r << 3n$) linear subspace of $\mathbb{R}^{3n}$. There is an infinite number of possible choices for this subspace and its basis, but ideally one would want a low-dimensional space that well-approximates the space of nonlinear deformations near the input poses.

For each of the dynamic morph targets, I employ linear modal analysis (LMA), which provides the best deformation basis for small deformations away from the rest configuration. Intuitively, modal basis vectors are directions into which the model can be pushed with the smallest possible increase in elastic strain energy. To find the modal basis vectors $\mathbf{U}_i$, I solve the following symmetric generalized eigenproblem (for a small number k of eigenvectors)

$$\mathbf{K}(\mathbf{x}_i^0)\mathbf{U}_i = \mathbf{M}\mathbf{U}_i\Lambda_i, \tag{5.9}$$

with Λ_p the diagonal eigenvalue matrix with the $k_i << 3n$ eigenvalues $0 < \lambda_1^i < \lambda_2^i < \ldots < \lambda_k^i$. The stiffness matrix $\mathbf{K} = \frac{\mathbf{R}(\mathbf{x})}{\mathbf{x}}$ is evaluated at $\mathbf{x}_i^0$, the rest configuration for input pose i, which defines a `goal' deformation for the input poses. Note that at this point, one could easily add modal derivatives, as in [BJ05].

In the next step, we have to combine the basis matrices $\mathbf{U}_i$ into a global basis matrix $\mathbf{U}$. We have taken into account three requirements when choosing the basis.

1. Avoid redundancy in the basis set, i.e. find an orthogonal set that is as compact as possible.

2. The characteristic deformations of all the morph targets have to be well represented.

3. The input deformations of each of the dynamic morph targets have to be well represented in the reduced space, otherwise the sculpted deformations cannot be simulated. In other words, the basis has to be *aware* of the morph targets.

The most straight forward approach is to combine all eigenvectors together as

$$\mathbf{U} = \left[\begin{array}{cccc} \mathbf{U}_1 & \mathbf{U}_2 & \dots & \mathbf{U}_m \end{array} \right],$$

and orthogonalize them (each set of eigenvectors $\mathbf{U}_i$ is orthogonal, but eigenvectors between different sets are not). This approach takes care of the second requirement, but results in a large set of eigenmodes with relatively small contributions for many eigenmodes, because all the common first deformation modes (such as stretch, shear, ...) are represented in each of the $\mathbf{U}_i$. Instead, similar to [BJ05], one can construct a low-dimensional motion subspace by applying mass-PCA. The derivatives are scaled according to the eigenvalues of the corresponding linear modes. Namely, the low-dimensional deformation basis is obtained by applying mass-PCA on the set of vectors

$$\frac{\lambda_1^i}{\lambda_j^i}\mathbf{U}_i^j \qquad i = 1, \dots, m; j = 1, \dots, k_i \ . \tag{5.10}$$

The first r principal modes are selected to form the basis $\mathbf{U}$. Scaling is necessary to put greater weight on the more important low-frequency modes, which would otherwise be masked by high-frequency modes. In case the eigenmodes in different poses show large variance, one can adapt the normalization factor such that the eigenvectors are normalized across morph targets, by replacing the scaling factor by $\frac{\lambda_1^i}{\lambda_k^i \lambda_j^i}$.

Finally, to make the basis aware of the morph targets, I add $m-1$ rest pose deformations $\mathbf{x}_p^0\ i = 2, \dots, m$ to the set $\mathbf{U}$ and re-mass-orthogonalize the set $\mathbf{U}_i \qquad \mathbf{x}_p^0$ into the final basis $\mathbf{U} \in \mathbb{R}^{3n \times r}$ with $r = (r\ + m - 1)$. Figure 5.3 illustrates the entire process.

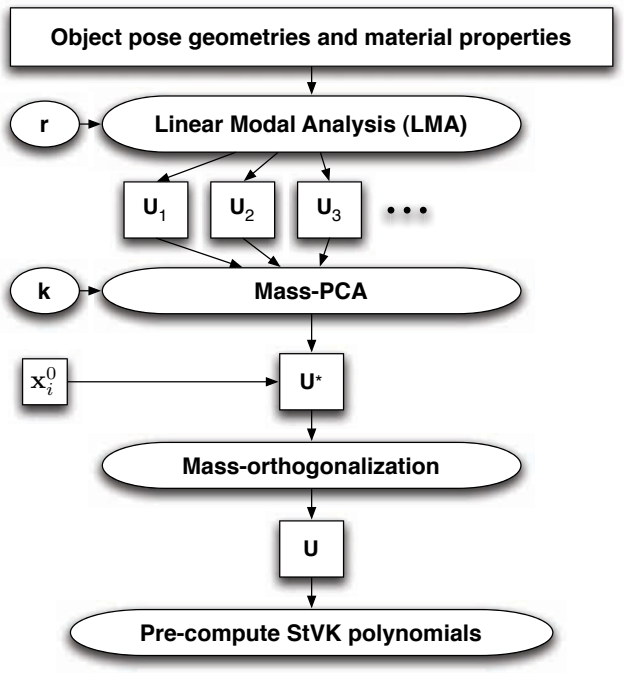


Figure 5.3: Construction of a morph target driven, mass-orthogonal reduction basis U: For each dynamic morph target i, during LMA the r smallest eigenmodes are selected to construct eigenbases $\mathbf{U}_i$. Mass-PCA combines and compacts the $\mathbf{U}_i$, retaining only k most significant modes. Finally, I explicitly add morph target deformations $\mathbf{x}_i^0$ to the eigenbasis and guarantee mass-orthogonality of the final basis $\mathbf{U}$.

5.4 Results

By using pose-space efficient polynomial interpolation to achieve pose-dependent behavior, I am able to demonstrate rich non-linear deformation effects at relatively small extra cost compared to simple simulation of linear or semi-non-linear materials. I have performed experiments with three different input models: a simple bulging cylinder with 4 bones (see Fig. 5.1), a shoulder model with 4 bones, and Herbert, the swimsuit model with 46 bones. For each model, I qualitatively compare simulations with different elastic models. I compare a single (pose-independent) elastic model with my pose-dependent elastic model that employs multiple dynamic morph targets, both with and without modal reduction.

Figure 5.4: Herbert jumps off on a diving board: Comparison of single (pose-independent) linear elasticity (left column), my method with dynamic morph targets (middle column), and my method with dynamic morph targets and modal reduction applied (right column). When balled up, Herbert's back (top) and belly (bottom) bulge in correspondence with his morph targets defined in Fig. 5.2. On the bottom left, Herbert's belly looks very flabby, as if he swallowed a brick. However, Herbert's `fetal pose' morph target 3 was authored with a stiff belly. My method (bottom right) shows the more desired behavior.

Herbert model For the Herbert simulations, only 3 morph targets were used, two of which are shown in Fig. 5.2. The first morph target is a skinny version of Herbert, in which his skin is very soft and flabby, the third target is a stiff, bulged Herbert in fetal position, while the second target has been chosen in between the first and the third. While the single elastic model shows little or no dynamic behavior, the pose-dependent

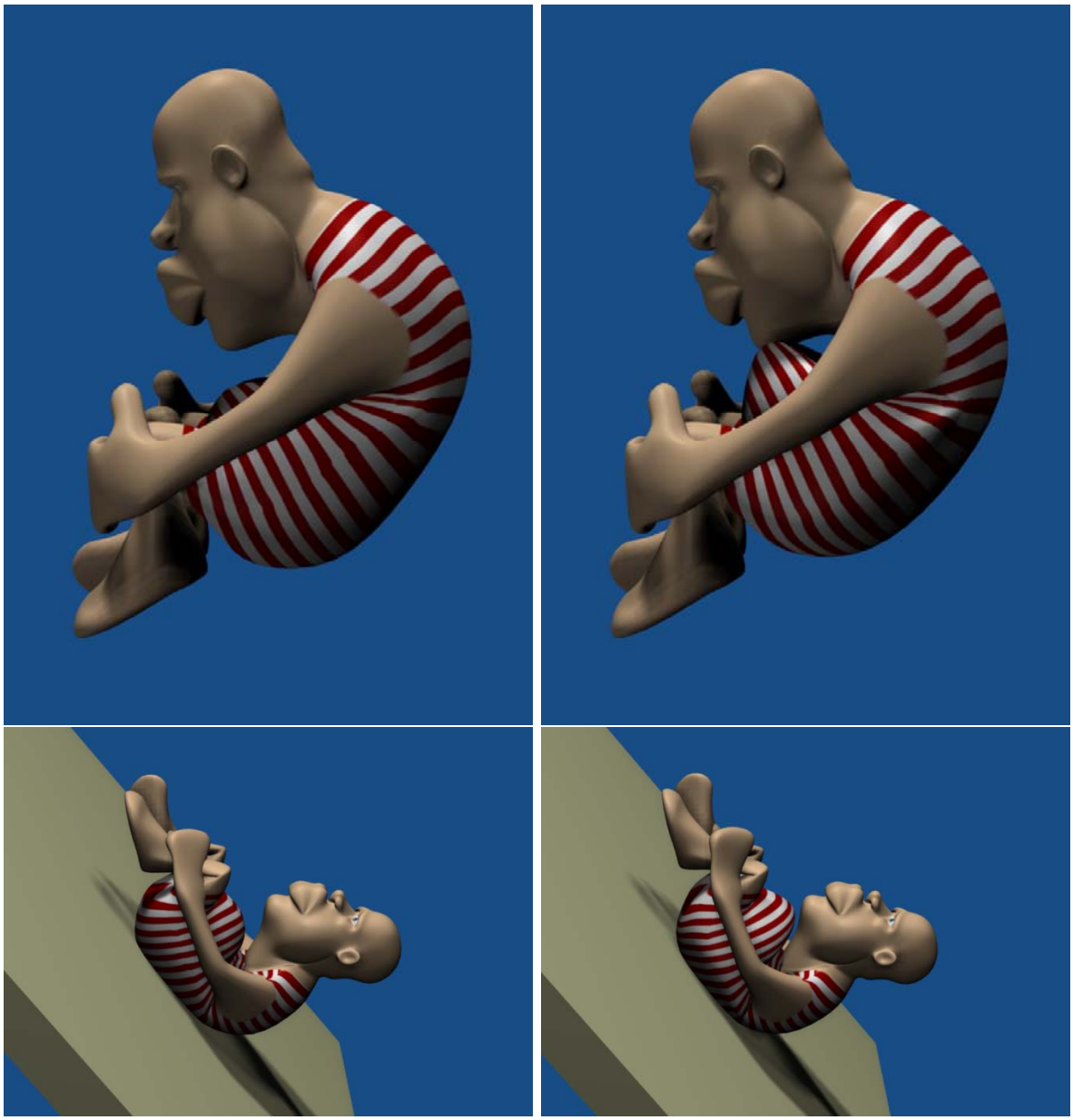


Figure 5.5: Herbert in flight: When under influence of kinematic events such as flipping and discontinuous velocity changes such as when hitting a wall, my method (right) still produces deformations consistent with Herbert's morph targets defined in Fig. 5.2.

elastic model adds a dramatic amount of realism due to the bulging behavior and inertial skin motion. The skinned Herbert model driven with a skeletal animation and my simulation framework adds inertial forces due to the bone's moving frames. As Herbert jumps off a diving board and flips through different poses, I show the advantage of the

pose-dependent model from an artistic viewpoint. With single elastic models, the belly is flabby and skinny throughout the entire simulation. Using the aforementioned morph targets for Herbert, an animator can impose a stiff, bulged belly in balled-up poses, and softer, skinny belly behavior in upright poses. Fig. 5.5 demonstrates the imposed behavior as Herbert's belly exposes bulging and non-flabby skin when he jumps from the diving board. Also, in Fig. 5.4, we show the use of reduced models in our method achieves the same quality of desired deformations as the computationally more expensive unreduced model.

Shoulder model My method also provides a physically-based approach to resolving regions affected by multiple joints, such as a shoulder rig. Our approach facilitates complex rigging: a set of skinning weights and a set of morph targets are sufficient to simulate complex co-articulation effects. There is no need for manual tweaking of the complex mapping of joint configuration to blending weights of geometric morph targets. In our shoulder example, we have 6 morph targets, shown in Fig. 5.6(a). This figure also shows the method's ability to simulate dynamic behavior at poses away from the morph target input poses. The input morph target set contains only one example of a folded elbow but two distinct folding scenarios are shown in the full simulation. Both folding scenarios show severe self-intersection in the single pose-independent model due to the effect of linear blend skinning. The presented pose-dependent model resolves both automatically. Another interesting co-articulation effect is the motion of the chest muscle as the arm makes a folding motion (see Fig. 5.7). Whereas the chest seems to collapse for single elastic models, it bulges more realistically with my method. The shoulder model has 4899 degrees of freedom. After modal reduction, it was possible to accelerate the simulation significantly by using only 19 eigenmodes (Table 5.1) with almost no visible effect on the simulation quality, as shown in Fig. 5.6.

Table 5.1: Model Statistics and Performance.

Model	Elasticity Type	# DOFS	# DMT DMTs	Preproc. (s)	Without DMTs (fps)	With DMTs (fps)
Bulging	Lin.	918	4	/	28	25
Cylinder	Red. Lin.	13	4	/	93	89
	Red. StVK	13	4	21.2	89	64
Herbert	Lin.	603	3	/	36	34
	Red. Lin.	12	3	/	96	92
	Red. StVK	12	3	16.8	90	58
Shoulder	Lin.	4899	6	/	3	3
	Red. Lin.	19	6	/	87	85
	Red. StVK	19	6	661	30	22

Performance In addition to qualitative comparisons, I have also compared simulation timings. All the experiments were performed on a Macbook Pro laptop with a single 2.4 GHz Intel Core 2 Duo processor, 2 GB of RAM and a NVidia GeForce 8600M GT graphics card. All rendering was done with the open-source Blender modeling package. Timing results are summarized in Table 5.1. All the techniques described in this chapter achieve real-time performance due to efficient pose-space interpolation of low-complexity linear elastic forces and modal reduction of either linear or semi-non-linear (StVK) forces. Comparing my method with the performance of single (pose-independent) elastic models, it is clear from Table 5.1 that my method has only a marginal extra cost, due to efficient polynomial interpolation of the dynamic morph target models. Finally, precomputation of the force polynomial coefficients in Eqn. (5.7) can be significant in case of StVK models, but never prohibitive. The precomputation times are shown for each specific experiment in Table 5.1.

Linear versus StVK Even though my method is general enough to handle any polynomial force model including the more expensive StVK model, my experiments show that the use of linear elastic models E_i for the dynamic morph targets yields very realistic, non-linear deformation effects due to polynomial force interpolation. I have included the StVK results in the timing results for completeness but did not notice bet-

ter visual quality. One of the key features of the StVK model is its rotation-invariance. In my skinning approach, where elastic deformation is expressed in skeleton's bind pose, rotation-invariance does not offer much advantage. Except for the simulation of the bulging cylinder, all images and videos show the use of linear dynamic morph target forces.

Comparison with other methods While geometric morph targets enable control of non-linear deformations, these deformations are purely static and cannot react to external forces in a physical simulation. Our dynamic morph targets add *dynamic behavior* to non-linear deformations such that external and inertial forces can be applied, as shown in Fig. 5.1. The method by Capell et al. [CBC+05] also enables deformations under influence of external forces, corresponding to the behavior in Fig. 5.1(b), but does not influence the underlying properties of the elastic material. As shown in Fig 5.1(c), our method can correct such undesirable behavior by setting elastic properties for each of the individual morph targets, effectively mimicking muscle contraction.

5.5 Advantages and Summary

In this chapter I have presented dynamic morph targets pose-dependent elastic models that allow an artist to easily author and control the geometry and elastic behavior of dynamic characters. The main advantages of my method over previous control approaches are three-fold: *quality of deformations, dynamic behavior and computational efficiency.* Although my method is physically based, expensive modeling of musculature or tendon influences is avoided, and instead I rely on physical constitutive models of deformable material to minimize skin pinching artifacts and I bypass complex rigging requirements that are common to purely geometric approaches. The use of such constitutive material models also enables response to external forces and inertial effects in dynamic simulations. Due to performance requirements, one is commonly restricted to

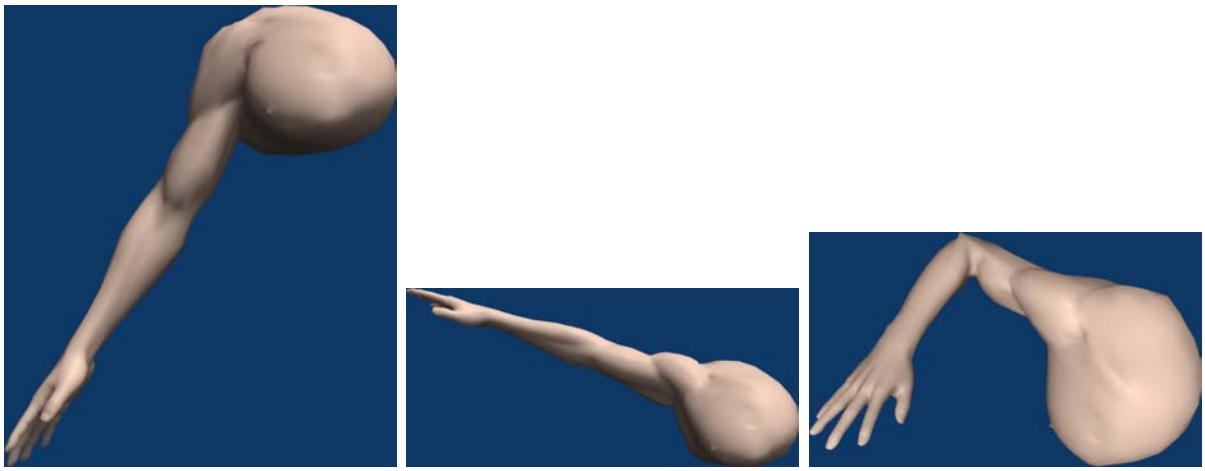
linear or quasi-linear models that cannot model pose-dependent effects such as bulging and wrinkling. Instead, I guide dynamic simulations by dynamic morph targets discrete pose-space examples of skin properties and deformations. Application of modal reduction to the basic framework in Sections 5.1.2 and 5.1.3 improves the runtime performance significantly, in order to guarantee real-time frame rates. Finally, my framework blends well with existing authoring pipelines, as described in Section 5.2.

5.6 Limitations and Future Work

The presentation of the algorithm in this chapter can be enhanced with additional features, especially with a component that was present in the last two chapters: deformations due to contact are not considered in this chapter. Even though the deformation models haven't changed, adding support for contact deformations can improve the ability to demonstrate the detailed deformations and dynamic effects that were present in the previous chapters. Dynamic morph targets merely change the coefficients and hence contact constraints could easily be added to the non-reduced dynamic model. However, because the skin stiffness matrix is not constant with dynamic morph targets, other acceleration techniques will be required. Additionally, support for contact constraints in reduced dynamic models could be provided by extending the method of [HSO03] to the pose-space interpolation setting of this chapter. Alternatively, an extension of the work by Bergou *et al.* [BMWG07] also provides possibilities for introducing low-dimensional constraints into the reduced dynamic system.

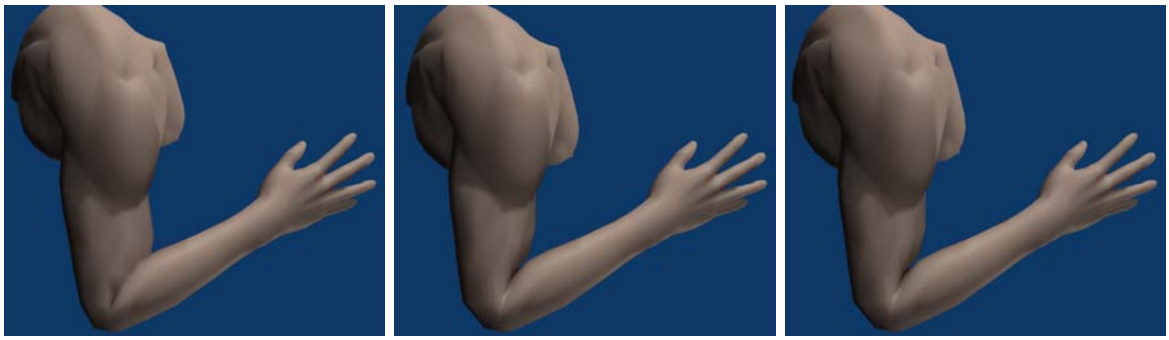
Another interesting area for future work is the extension of the pose-space deformation framework to weighted pose-space deformation [KM04], which allows for a smaller set of input poses. In my simulations, as shown in the provided figures, the relatively small number of input morph targets were sufficient without the use of weighted pose-space deformation. Lastly, so far I have used my method with artist-authored models,

but it would be interesting to exploit it for simulating complex nonlinear materials measured from reality. This would also serve well for quantitatively evaluating the approximation quality of the method, especially with respect to the added value of using StVK materials versus pose-space linear elastic materials.



(a) **Shoulder example morph targets:** A few skinned input poses with associated target deformations as provided by an artist (targets 1, 3 and 6). I used 6 morph targets in total, including one bent arm input with bicep bulging (target 6).

(b) **Bicep muscle bulge and self-intersection:** At runtime, a pose-independent force model clearly shows undesired and self-intersecting deformations when bending the arm (left column). On the other hand, my method in the middle column shows deformations consistent with morph target 6 (Fig. (a)), even after application of modal reduction for efficiency (right column).



(c) **Extrapolation to new shoulder poses:** My method also extrapolates to non-input poses.

Figure 5.6: Shoulder Rig Simulation

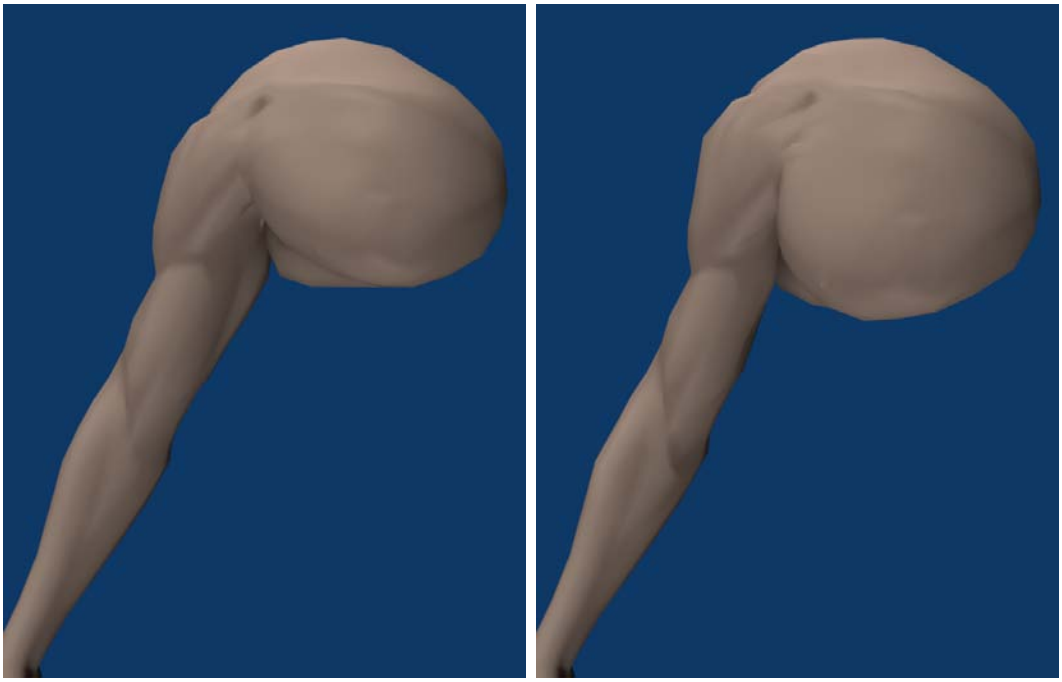


Figure 5.7: Chest flex at runtime: On the left, a pose-independent force model causes the chest to collapse as the arm of the character is lowered at runtime. On the right, with my method, the chest correctly deforms consistent with morph target 6 as shown in Fig. 5.6(a).

Chapter 6

Conclusion

In this dissertation, I have presented a unified framework for real-time modeling of soft, articulated characters with highly detailed skin. This framework combines character animation, dynamic elastic simulation, fast contact response and directable deformation. The methods I presented reduce the computational complexity by exploiting layered models and simplified physics-based models, thus enabling real-time performance. In this chapter, I summarize the main results of this dissertation and discuss potential future research directions.

6.1 Summary of Results

I have presented methods on the simulation of soft articulated bodies for computer graphics. These methods enable real-time animation, simulation and control of characters with contact and collision of very detailed skin, typically with thousands of deformable vertices. First, I propose the use of layered models, to concentrate computational resources on the area that is often the most interesting: the outer skin layer. Simultaneously, my system preserves the dynamic behavior of the entire volume by appropriate time and space discretization of continuum Lagrangian dynamics with implicit integration of contact forces. Thus, I guarantee realistic behavior in dynamic and interactive scenes, as well as stable and responsive contact handling. Correct contact

handling, including friction, is particularly hard for layered models because of the interplay between skeletal motion and soft surface contact. Due to the tight mathematical coupling between these two effects, I have proposed physically-inspired approximation to reduce the computational complexity of layered contact response, enabling real-time performance. For the simulation of non-articulated objects, I have proposed the use of dynamic deformation textures, a very efficient representation that maps well to parallel architectures such as GPUs. Dynamic deformation textures are exploited and implemented on the GPU for all stages of the algorithm presented in Chapter 3: in forward skin deformation dynamics, collision detection, contact response and rendering. The resulting implementation is able to process more than a million deformable elements per second and up to fifteen thousand contacts per second. This is comparable to the performance of techniques that use explicit integration (e.g. [ZC99]), without its time step restrictions and considerably faster than other methods that enable large time steps. My approach can also handle many more contact points than novel quasi-rigid dynamics algorithms using LCP [PPG04] while also producing richer deformations.

The system in Chapter 3 does not support global deformation modes such as bending, twisting or stretching. This limitation was addressed in Chapter 4, where I have extended the layered model of Chapter 3 to soft articulated characters, subject to skeletal animation and skeletal contact response. Naturally, surface collisions simultaneously cause skin deformations and skeletal motion. In my framework, I propose anticipation of skeleton response to enable fast coupled contact response, reducing the worst-case $O(mk)$ complexity to $O(m+k)$ in practice, for m contacts and k bones. For character meshes with n vertices, I have combined fast contact handling with an extension of non-articulated pose-space dynamics (Chapter 3) to articulated characters, effectively reducing the worst-case $O(nmk)$ complexity to $O(n+m+k)$ in practice. To the best of my knowledge, this system is the first to enable real-time simulation of soft articulated characters with frictional contact response that captures the interplay between skeletal

dynamics and skin deformation.

At the level of bones and joints, I have introduced two approaches for controlling the forward dynamics in Chapter 4. The skeletal deformation can be driven by skeletal animation and controlled by varying magnitudes of joint stiffness. But often users want to be able to control the shape of the character at a finer level, beyond simple skeletal deformation. Behavior such as muscle bulging and skin wrinkling are simply not possible with the pose-space linear dynamics proposed in Chapters 3 and 4. With dynamic morph targets, the data-driven approach presented in Chapter 5, control of the behavior of elastic, deformable material in a dynamic simulation is made possible simply by providing examples of desired shapes. Dynamic morph targets define the pose-dependent physical state of soft objects, including shape and elastic and inertial properties. Realistic animation of skin and muscular deformations is a complex and subtle phenomenon due to the tightly coupled interplay between bones and musculature governing the deformations. Most physical skin deformation models used in computer graphics today are not sophisticated enough to reproduce such complex behavior, or their computational complexity is too high to be practical for interactive computer graphics applications. Instead of increasing the complexity of the simulation model, I have proposed to give animators control over complex skin behaviors that are hard to capture in a physical model. This method is efficient at runtime through modal reduction and pose-space polynomial interpolation with radial basis functions. Likewise, it is much more practical than simulation of non-linear materials, both in implementation and runtime performance, while it nevertheless achieves rich, non-linear effects. In addition to physically-based contact response approximation in Chapters 3 and 4, non-linear elastic model approximation in Chapter 5 is the second type of model simplification applied in this dissertation. As a result, I was able to retain real-time performance for all methods presented in this dissertation.

6.2 Future Work

In the design of the methods presented in this dissertation, I have made a number of assumptions and have sometimes restricted the scope of my system for the sake of performance or ease of implementation. Some of these restrictions were relaxed in the course of my research, such as adding support for skeletal deformation modes in Chapter 4 or for rich non-linear effects such as bulging in Chapter 5. This section reiterates remaining limitations, offers potential solutions, examines ways to relax some assumptions, and explores ways to broaden the scope of my work with novel applications.

6.2.1 Limitations

As summarized in Section 6.1, the methods presented in Chapter 4 can be considered as a natural extension to resolve some limitations of the system presented in Chapter 3. Similarly, dynamic morph targets (Chapter 5) complement techniques in Chapers 3 and 4 to support rich non-linear effects and to account for the lack of control in the earlier methods. Nevertheless, there are still a number of improvements possible:

- Layered models, more specifically layered models with a single layer of deformable skin tissue on the surface of a body that are used throughout my work, do not preserve volume well. The reason does not necessarily lie in the linearized dynamics employed, because the dynamic formulation with FEM discretization has reasonably good volume preservation for small deformations. It lies in the fact that the deformations are often fairly large, to compensate for the lack of deformation degrees of freedom in the (non-)articulated core. Volume constraint forces can cause locking of competing forces, therefore an alternative approach, based on fluid dynamics, may be required [ISF07]. Alternatively, one could relax the assumptions on the core model, as proposed later in this section.

- The performance of dynamic deformation textures as presented in Chapter 3 is

partially dependent on the parameterization of the surface. The parameterization determines the amount and smoothness of the patch borders, which in turn influences the convergence of the Conjugate Gradient iterative solver. For my experiments, this was not a major issue, but it could be improved on. This implementation issue is due to the choice of mapping the degrees of freedom to a *regular* grid. As texture parameterization is a broad and very active field of research, there is certainly potential to find a parameterization that is tuned specifically to dynamic morph targets. Alternatively, one could lift the concept of dynamic deformation textures to a more abstract meaning. One could relax the assumption of a completely regular grid to a *semi-regular* grid. This is discussed later in this section.

- The image-space collision detection algorithm presented in Section 3.3 is approximate because of the distortion that is associated with back-projecting collision information from the contact domain to the dynamic deformation texture domain. This could be resolved by replacing this part of my collision detection pipeline with a more sophisticated primitive-level collision detection.

- I have employed velocity *equality* constraints to resolve deformable collisions. This works well for impact resolution and frictional contact effects such as rolling, but can lead to artifacts for resting contact. It would be interesting to explore ways to handle *inequality* constraints, as this would allow modeling more accurate contact forces as well as joint limits.

- Contact constraints for controllable models with dynamic morph targets were not presented in this thesis. Although the elastic models of Chapters 3 and 4 also govern the dynamics of models with DMTs, adding contact constraints efficiently requires new acceleration techniques (Section 5.6). Also, support for constraints with reduced models requires additional research. A good starting point is an

extension of the method of Hauser *et al.* [HSO03] to the pose-space interpolation setting. Another possibility is an extension of the work by Bergou *et al.* [BMWG07] to introduce low-dimensional constraints into the reduced dynamic system.

- In my simulations with dynamic morph targets in Section 5.4, the relatively small number of input morph targets were sufficient to achieve the desired effects. Nevertheless, for other applications, many more morph targets could be required to capture all desired deformations, especially for highly articulated characters. In that case, the extension of the pose-space deformation framework to weighted pose-space deformation [KM04], which allows for a smaller set of input poses could be useful.

6.2.2 Relaxing Design Assumptions

In the design of my framework, I have chosen to restrict the class of characters to objects with a rigid or articulated core covered with a single layer of deformable tissue. In addition, dynamic deformation textures were defined to be mappings to a fully regular grid for easy and efficient implementation on GPUs. In this section, I will briefly discuss ways to relax those assumptions to generalize my approach and deal with some of the limitations of Section 6.2.1.

Generalization of the Layered Model As mentioned before, the lack of degrees of freedom together with the lack of an *elastic deformation model in the core* has led to some unrealistic effects. The lack of degrees of freedom causes prevalent large deformations in the outer skin layer for which linear elastic laws aren't very accurate. In addition, the core was assumed either completely rigid, or articulated and complemented with simple joint constraints. With such a representation, material compression inside the core, usually around the joints, is not modeled at all. In Chapter 4, I proposed a solution by introducing joint stiffness. The main disadvantage of this approach is that the stiffness

has to be tuned manually for each joint to achieve both realistic behavior and stable simulation. Alternatively, many of these limitations could be relieved by generalizing the concept of the inner core. Future research could be geared towards augmenting the inner core with a low-dimensional elastic deformation model, and formalizing the dynamic interface between the core and the high-resolution outer tissue layer. With an appropriate choice of deformation model for the core, this approach could naturally provide the volume-preserving behavior that is missing in some of my methods, as well as inherent joint stiffness through material compression. One of the major challenges in this approach lies in the formulation of a contact handling algorithm that is stable and responsive as has been done in this dissertation. Either rigid or loose coupling between core and outer tissue layer are valid options, but in the end the resulting objects have to react realistically to surface contact, without restrictions to the material properties of either core or skin.

Improved Parallelization Dynamic deformation textures provide an extremely fast implementation of pose-space elastic dynamics of non-articulated objects on the GPU. The regular grid is very amenable to parallel implementation of iterative numerical solvers. Unfortunately, due to parameterization issues, mapping the surface of articulated characters to a regular grid is less straightforward. Unwrapping such surfaces inherently requires a considerable number of seams, resulting in many different patches in the parameterized domain and even more patch boundaries that have to be interfaced with boundary conditions. The boundary conditions negatively affect the dimensionality, conditioning of the system of equations and convergence rate of the numerical solvers. Alternatively, one could consider forgoing the parameterized domain, directly mapping surface degrees of freedom to a sparse linear system, as is commonly done. In fact, the sparse matrix is just another form of grid, although it can only be considered semi-regular because its sparsity pattern is not completely regular. A regular sparsity

pattern is the main reason for the extremely high efficiency of dynamic deformation textures on the GPU. Nevertheless, many techniques exist for re-ordering linear systems to increase the regularity of its sparsity pattern [Kar03, DGLN04]. In fact, in the articulated framework of Chapter 4, I have applied such strategy to increase the performance of the CPU-based solvers. Nevertheless, considering the increasing attention for massively parallel architectures and imminent appearance of many-core systems, the real challenge lies in parallelization of advanced solvers beyond simple Conjugate Gradient solvers, such as for example direct sparse solvers for semi- and indefinite systems of equations, as in Equation 4.31.

6.2.3 Beyond Current Applications

In addition to resolving limitations, the work in this dissertation also motivates exploring new and exciting applications. Here I briefly present ideas in the areas of real-time medical applications and active control.

Realistic Nonlinear Materials for Surgery Simulators For the experiments in Section 5.4, the dynamic morph target shapes were hand-modeled with the goal of giving artistic control to the outcome of the dynamic simulation. The resulting simulations show rich non-linear deformations achieved with my novel pose-dependent elastic model, such as muscle bulging. This effect could not be achieved with simple pose-independent linear forces, and incurs only marginal extra computational cost. The main goal in these experiments was to achieve more interesting dynamics with a good level of intuitive artistic control. On the other hand, I believe dynamic morph targets and the efficient pose-dependent model could provide accurate real-time simulation of nonlinear elastic behavior in medical applications, such as surgery simulators. The essential difference with the experiments in Section 5.4 would be the way the dynamic morph targets are produced: the set of example shapes and material properties could come

from either a highly accurate nonlinear off-line simulator, or from measurements of real physical tissue. It would be interesting to investigate the accuracy at which my pose-dependent elastic model is able to approximate complex non-linear physical materials in real time, given sufficient and appropriate morph targets. With sufficient accuracy, this method could be invaluable to real-time surgery simulators. Along with more input poses and hence more dynamic morph targets also comes the burden of performance degradation. An intelligent morph target selection scheme in combination with the extension to weighted pose-space deformation [KM04] will probably be required to retain real-time performance as well as the desired level of accuracy.

Active Control and Motion Synthesis In Chapter 5, I have proposed a method for directable dynamics of *passively* controlled systems. The control is passive, because dynamic morph targets only have indirect control over the output shape. Eventually, physical laws govern the final output shape. An exciting possibility for future research lies in directable *active* systems with deformable characters, in combination with the passive control of dynamic morph targets. Actively directed systems require a real-time controller which directs the physical simulation to follow a given input trajectory. A trajectory could be an actual physical path, but it could also represent a practical task, such as obstacle avoidance, grasping or balancing. Imagine for example a range of human characters with varying body structure and muscle mass, all trying to balance on a rocking platform with a unified controller. In order to generate realistic behavior, such controller needs to account for the dynamic properties, such as force generation in the muscles, as well as the elastic properties in the deformable tissue, as both will influence the behavior. Inverse dynamics [IC87] and constrained-Lagrangian inverse dynamics [BMWG07] can drive such a controller but use arbitrary forces that may cancel the natural dynamics and will look over-controlled. Compliant controllers such as proportional-derivative (PD) controllers [SNF05, WTF06], could also do the job but

depend on gains that are tuned either manually or with heuristics and may not perform well for many degrees of freedom. Optimal control minimizes the injection of fictional control forces by cooperating with the natural dynamics of the system [PW99, FP03, LHP05]. In other words, it can also cooperate with the dynamics of a pose-dependent system such as the one proposed in Chapter 5. The combination of such a controller with my dynamic morph targets could unite the benefits of both passive and active control in exciting new ways.

6.3 Conclusion

This dissertation has provided a strong incentive for using layered models in character animation systems. Layered models provide many benefits for real-time controllable simulation of realistic characters, provided that they are appropriately integrated with responsive contact handling and intuitive deformation control. Additionally, other types of layers could be added: for example, an animation layer to control high-level character traits or layers with planning and active control for motion synthesis. I believe that such layered models can lead to the creation of truly interactive, autonomous and persuasive characters.

Appendix A

Kinematic Relationships

A.1 Non-articulated (Single-Core) Objects

The angular velocity $\in \mathbb{R}^3$ can be expressed in terms of the derivative of a quaternion $\theta \in \mathbb{R}^4$ by the linear relationship $= \mathbf{G}\theta$ [Sha89]. Similarly, we can express the relationship between the velocity state vector $\mathbf{v}$ and the time-derivatives of the generalized coordinates $\mathbf{q}$ as:

$$\mathbf{v} = \mathbf{P}\mathbf{q}, \qquad \mathbf{q} = \mathbf{P}^{+}\mathbf{v}, \tag{A.1}$$

$$\mathbf{P} = \begin{bmatrix} \mathbf{I}_3 & 0 & 0 \\ 0 & \mathbf{G} & 0 \\ 0 & 0 & \mathbf{I}_n \end{bmatrix}, \qquad \mathbf{P}^{+} = \begin{bmatrix} \mathbf{I}_3 & 0 & 0 \\ 0 & \frac{1}{4}\mathbf{G}^T & 0 \\ 0 & 0 & \mathbf{I}_n \end{bmatrix},$$

where n is the number of elastic coordinates, and $\mathbf{P}\mathbf{P}^{+} = I$.

We can now derive the world-frame velocity of a material point in terms of the velocity state vector:

$$\mathbf{x} = \mathbf{c} + \mathbf{R}\mathbf{u} + \mathbf{R}\mathbf{u} \tag{A.2}$$

$$= \mathbf{c} + \mathbf{B}\theta + \mathbf{R}\mathbf{S}\mathbf{q}_e. \tag{A.3}$$

The matrix $\mathbf{B}$ is the Jacobian of the vector $\mathbf{R}\mathbf{u}$ w.r.t. θ, and it can be proven to be equal to $-\mathbf{R}\mathbf{u}\mathbf{G}$ [Sha89], where $\mathbf{u}$ is the skew-symmetric cross-product matrix obtained from $\mathbf{u}$. We can rewrite (A.3) in compact matrix form as a linear function of the time

derivative of the generalized coordinate set $\mathbf{q}$:

$$\mathbf{x} = \begin{bmatrix} \mathbf{I}_3 & -\mathbf{RuG} & \mathbf{RS} \end{bmatrix} \mathbf{q}. \tag{A.4}$$

Applying $\mathbf{q} = \mathbf{P}^{+}\mathbf{v}$, we can rewrite (A.4) and obtain:

$$\mathbf{x} = \begin{bmatrix} \mathbf{I}_3 & -\mathbf{Ru} & \mathbf{RS} \end{bmatrix} \mathbf{v} = \mathbf{Lv}. \tag{A.5}$$

A.2 Articulated Characters

We can derive the world-frame velocity of a material point in terms of the velocity state vector by time differentiation of (4.5):

$$\begin{aligned} \mathbf{x} &= \sum_{i=1}^{k} w_i(\mathbf{c}_i - \mathbf{R}_i\mathbf{u}_i \quad _i + \mathbf{R}_i\mathbf{R}_{o\,i}\mathbf{S}\mathbf{q}_s) \\ &= \sum_{i=1}^{k} w_i(\mathbf{c}_i + \mathbf{B}_i\theta_i + \mathbf{R}_i\mathbf{R}_{o\,i}\mathbf{S}\mathbf{q}_s) \end{aligned} \tag{A.6}$$

Each matrix $\mathbf{B}_i$ is the Jacobian of a vector $\mathbf{R}_i\mathbf{u}_i$ w.r.t. θ_i, with $\mathbf{u}_i$ a position in local bone space. The Jacobian can be proven to be equal to $-\mathbf{R}_i\mathbf{u}_i\mathbf{G}_i$ [Sha89], where $\mathbf{u}_i$ is the skew-symmetric cross-product matrix, and $\mathbf{G}$ relates local-frame angular velocities to time derivatives of quaternions through $\quad = \mathbf{G}\theta$. We can rewrite (A.6) in compact matrix form as a linear function of the velocity state vector $\mathbf{v}$ (after application of $\mathbf{q} = \mathbf{P}^{+}\mathbf{v}$ that encapsulates the adjoint relationship $\mathbf{G}$, see Appendix A.1 for details):

$$\mathbf{x} = \mathbf{L}_W\mathbf{v} = \begin{bmatrix} \mathbf{W} & \mathbf{B}_W & \mathbf{R}_W\mathbf{S} \end{bmatrix} \mathbf{v}, \tag{A.7}$$

$$\mathbf{B}_W = \begin{bmatrix} -w_1\mathbf{R}_1\mathbf{u}_1 & \ldots & -w_k\mathbf{R}_k\mathbf{u}_k \end{bmatrix}, \quad \mathbf{R}_W = \sum_{i=1}^{k} w_i\mathbf{R}_i\mathbf{R}_{o\,i},$$

and $\mathbf{W}$ is a diagonal weight matrix. $\mathbf{L}_W$ is position-dependent.

Appendix B

Lagrangian Motion Equations with Finite Element Method

B.1 Lagrangian Formulation

From Lagrangian continuum mechanics, the motion equations of a deformable body with generalized coordinate set $\mathbf{q}$ can be written in their general form as [GPS02]:

$$\frac{d}{dt}\left(\frac{\mathcal{T}}{\mathbf{q}}\right)^T - \left(\frac{\mathcal{T}}{\mathbf{q}}\right)^T + \left(\frac{\mathcal{F}}{\mathbf{q}}\right)^T + \ {}_{\mathbf{x}}\mathcal{E} = \bar{\mathbf{Q}}, \tag{B.1}$$

where $\mathcal{T}$ is the kinetic energy of the body, $\mathcal{F}$ is the work done by the body against dissipative forces, $\mathcal{E}(\mathbf{x})$ is the elastic energy of the body, and $\bar{\mathbf{Q}}$ is the vector of generalized external forces which includes gravity and contact forces.

B.2 Elastic Energy

The virtual work due to elastic forces can be written as

$$\delta W_e = - \int_V \sigma^T \delta\epsilon dV \tag{B.2}$$

where ϵ and σ are the stress and strain vectors. With our choice of linear strain model, the strain can be written in terms of the displacement field as $\epsilon = \mathbf{B}\mathbf{u}_e$, where $\mathbf{B}$ is a differential operator matrix. In terms of the generalized elastic coordinates of the body,

this becomes $\epsilon = \mathbf{BSq}_e$.

For a linear isotropic material, the constitutive relationship between stress and strain is $\sigma = \mathbf{E}\epsilon$, with $\mathbf{E}$ the symmetric matrix of elastic coefficients, defined by the two Lamé material constants λ and μ. This enables writing the stress vector in terms of the generalized elastic coordinates. Substituting ϵ and σ into (B.2) yields an expression for the virtual work due to elastic forces:

$$\delta W_e = -\mathbf{q}_e^T \left[\int_V (\mathbf{BS})^T \mathbf{EBS} dV \right] \delta \mathbf{q}_e = -\mathbf{q}_e^T \mathbf{K}_e \delta \mathbf{q}_e \tag{B.3}$$

Here, $\mathbf{K}_e$ is the symmetric positive definite stiffness matrix associated with the elastic coordinates of the body. The generalized stiffness matrix $\bar{\mathbf{K}}$ can be formed from $\mathbf{K}_e$ as $\begin{matrix} 0 & 0 \\ 0 & \mathbf{K}_e \end{matrix}$. Following equation (B.3), the virtual work due to elastic forces can be written as $\delta W_e = \bar{\mathbf{Q}}_e^T \delta \mathbf{q}_e$, where $\bar{\mathbf{Q}}_e = -\bar{\mathbf{K}}\mathbf{q}_e$ is regarded as a generalized force acting on the body, or equivalently, $-\ {}_{\mathbf{x}}\mathcal{E}$.

B.3 Motion Equations

The various terms of the Lagrangian equation (B.1) can be rewritten by integrating the kinetic energy $\mathcal{T}$, the work produced against dissipative forces $\mathcal{F}$, and the elastic energy $\mathcal{E}$ over the entire deformable body, exploiting the texture-based discretization of the deformable layer described in Section 3.3 in the paper:

$$\frac{d}{dt}\left(\frac{\mathcal{T}}{\mathbf{q}}\right)^T - \left(\frac{\mathcal{T}}{\mathbf{q}}\right)^T = \bar{\mathbf{M}}\mathbf{q} + \bar{\mathbf{M}}\mathbf{q} - \left[\frac{}{\mathbf{q}}\left(\frac{1}{2}\mathbf{q}^T\bar{\mathbf{M}}\mathbf{q}\right)\right]^T, \tag{B.4}$$

$$\left(\frac{\mathcal{F}}{\mathbf{q}}\right)^T = \bar{\mathbf{D}}\mathbf{q}, \tag{B.5}$$

$${}_{\mathbf{x}}\mathcal{E} = \bar{\mathbf{K}}\mathbf{q}, \tag{B.6}$$

where $\bar{\mathbf{M}}$, $\bar{\mathbf{D}}$, and $\bar{\mathbf{K}}$ are, respectively, the generalized mass, damping, and stiffness matrices of the deformable body. They are obtained by integration with linear elements and linear basis functions, and for their exact expressions we refer to [Sha89]. Note that, due to definition of the elastic energy based on the displacement field, the generalized elastic forces only depend on the elastic coordinates.

We define the mass matrix $\mathbf{M} = (\mathbf{P}^+)^T\bar{\mathbf{M}}\mathbf{P}^+$, damping matrix $\mathbf{D} = (\mathbf{P}^+)^T\bar{\mathbf{D}}\mathbf{P}^+$, and stiffness matrix $\mathbf{K} = (\mathbf{P}^+)^T\bar{\mathbf{K}}\mathbf{P}^+$. We also define transformed external forces $\mathbf{Q} = (\mathbf{P}^+)^T\bar{\mathbf{Q}}$, and a *quadratic velocity vector* $\mathbf{Q}_v$ as

$$\mathbf{Q}_v = (\mathbf{P}^+)^T \quad -\bar{\mathbf{M}}\mathbf{q} + \left[\frac{}{\mathbf{q}}\left(\frac{1}{2}\mathbf{q}^T\bar{\mathbf{M}}\mathbf{q}\right)\right]^T \quad . \tag{B.7}$$

After some algebraic manipulation, and applying $\mathbf{v} = \mathbf{P}\mathbf{q}$ (See Appendix A.1 for the definition of $\mathbf{P}$ and $\mathbf{P}^+$), the system of motion equations can be reduced to its familiar form:

$$\begin{aligned} &\mathbf{M}\mathbf{v} = \mathbf{Q} + \mathbf{Q}_v - \mathbf{K}\mathbf{q} - \mathbf{D}\mathbf{v} = \mathbf{F}, \\ &\mathbf{q} = \mathbf{P}^+\mathbf{v}. \end{aligned} \tag{B.8}$$

B.4 External Forces

Generalized forces $\bar{\mathbf{Q}}$ can be computed from world-frame forces $\mathbf{f}$ applying the principle of virtual work. Using the kinematic relationship $\mathbf{x} = \mathbf{L}\mathbf{P}\mathbf{v}$ (See Appendix A in the paper), a world-frame force $\mathbf{f}_p$ applied at a point p on the deformable body induces a generalized force $\bar{\mathbf{Q}}_p = \mathbf{P}^T\mathbf{L}(p)^T\mathbf{f}_p$.

B.5 Mass Matrix

The mass matrix $\mathbf{M}$ has the following structure [Sha89]:

$$\mathbf{M} = \begin{matrix} m\mathbf{I}_3 & \mathbf{R}\mathbf{S}_t & \mathbf{R}\bar{\mathbf{S}} \\ -\mathbf{S}_t\mathbf{R}^T & \mathbf{I}_\theta & \mathbf{I}_{\theta e} \\ \bar{\mathbf{S}}^T\mathbf{R}^T & \mathbf{I}_{\theta e}^T & \mathbf{M}_e \end{matrix}, \tag{B.9}$$

with mass integral m, $\mathbf{I}_\theta$ the usual inertia tensor, time dependent inertia shape integrals

$$\mathbf{S}_t = \int \rho \mathbf{u} dV, \tag{B.10}$$

$$\bar{\mathbf{S}} = \int \rho \mathbf{S} dV, \tag{B.11}$$

and

$$\mathbf{I}_{\theta e} = \int \rho \mathbf{u}\mathbf{S} dV. \tag{B.12}$$

B.5.1 Quadratic Velocity Vector

From the mass matrix $\mathbf{M}$ and (B.7), the quadratic velocity vector reverts to [Sha89]:

$$\mathbf{Q}_v = \begin{matrix} \mathbf{Q}_{vc} \\ \mathbf{Q}_{ve} \end{matrix}, \quad \mathbf{Q}_{vc} = \begin{matrix} -\mathbf{R} & \times \left(& \times \mathbf{S}_t + 2\bar{\mathbf{S}}\mathbf{v}_e \right) \\ -2\mathbf{I}_\theta & - 2\mathbf{I}_{\theta e}\mathbf{v}_e - \mathbf{I}_\theta & \end{matrix}, \quad \mathbf{Q}_{ve} = \left(-\mathbf{M}_e[\ \ ^2]\underline{\mathbf{u}} - 2\mathbf{M}_e[\ \ ]\mathbf{v}_e \right), \tag{B.13}$$

where $[\mathbf{A}]$ denotes a block diagonal matrix with $\mathbf{A}$ replicated in every block, and $\underline{\mathbf{u}}$ is a column vector that packs the body-frame position $\mathbf{u}$ of all simulation nodes.

Appendix C

Joint Compliance for Hinge Joint

For a hinge joint aligned with axis of rotation $\mathbf{u}$, we model joint stiffness between bones i and j with an angular spring generating torques $\mathbf{T} = \pm k\theta\mathbf{u}$ proportional to the joint angle θ. This torque is (conceptually) be encoded in the system stiffness matrix $\mathbf{K}$, more specifically the block $\mathbf{K}_b$ associated with the character's bones (see Section 4.2.1).

For implicit integration in (4.15), we also need the Jacobians $\mathbf{J} = \frac{\mathbf{T}}{\mathbf{q}}$. We use quaternions $\mathbf{q} = (s, x, y, z) = (q_s, \mathbf{q_u})$ to represent orientations and quaternion matrices [Die06] to represent quaternion multiplication: $\mathbf{q}_i \otimes \mathbf{q}_j = \mathbf{Q}(\mathbf{q}_i)\mathbf{q}_j = \bar{\mathbf{Q}}(\mathbf{q}_j)\mathbf{q}_i$:

$$\mathbf{Q}(\mathbf{q}) = \begin{matrix} s & -x & -y & -z \\ x & s & -z & y \\ y & z & s & -x \\ z & -y & x & s \end{matrix} = \begin{matrix} \mathbf{Q}_s \\ \mathbf{Q_u} \end{matrix}$$

$$\bar{\mathbf{Q}}(\mathbf{q}) = \begin{matrix} s & -x & -y & -z \\ x & s & z & -y \\ y & -z & s & x \\ z & y & -x & s \end{matrix}$$

$$\text{with} \quad \mathbf{Q}_s \in \mathbb{R}^{1\times 4} \quad \text{and} \quad \mathbf{Q_u} \in \mathbb{R}^{3\times 4}.$$

The difference orientation $\mathbf{q} = (q_s, \mathbf{q_u})$ between two quaternions $\mathbf{q}_i$ and $\mathbf{q}_j$ can be extracted with these quaternion matrices:

$$\mathbf{Q}_i = \bar{\mathbf{Q}}(\bar{\mathbf{q}}_i)$$

$$\mathbf{Q}_j = \mathbf{Q}(\bar{\mathbf{q}}_j) \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{matrix}$$

as follows:

$$q_s = \mathbf{Q}_{j\,s}\mathbf{q}_i = \mathbf{Q}_{i\,s}\mathbf{q}_j$$
$$\mathbf{q_u} = \mathbf{Q}_{j\,\mathbf{u}}\mathbf{q}_i = \mathbf{Q}_{i\,\mathbf{u}}\mathbf{q}_j.$$

Here, the subscripts $\mathbf{Q}_{\,s}$ and $\mathbf{Q}_{\,\mathbf{u}}$ refer to the first row respectively the lower 3×4 submatrix of $\mathbf{Q}$. Then, the Jacobians can be computed as follows:

$$\mathbf{J}^i_j = \frac{\mathbf{T}_i}{\mathbf{q_j}} = -\mathbf{J}^j_j = kz\big(w\mathbf{u}\mathbf{Q}_{i\,s} + \theta\mathbf{Q}_{i\,\mathbf{u}}\big)\frac{\mathbf{G}^T_j}{4}$$
$$\mathbf{J}^i_i = \frac{\mathbf{T}_j}{\mathbf{q_i}} = -\mathbf{J}^j_i = kz\big(w\mathbf{u}\mathbf{Q}_{j\,s} + \theta\mathbf{Q}_{j\,\mathbf{u}}\big)\frac{\mathbf{G}^T_i}{4}$$
$$\text{with} \quad z = cosec(\frac{\theta}{2}) \quad w = \frac{\theta}{tan(\theta/2)} - 2$$

For very small difference angles, we compute $\lim_{\theta\to 0}\mathbf{J}$:

$$\mathbf{J}^i_j = -\mathbf{J}^j_j = k\big(2\mathbf{Q}_{i\,\mathbf{u}} - \frac{\theta}{2}\mathbf{u}^T\mathbf{Q}_{i\,s}\big)\frac{\mathbf{G}^T_j}{4}$$
$$\mathbf{J}^i_i = -\mathbf{J}^j_i = k\big(2\mathbf{Q}_{j\,\mathbf{u}} - \frac{\theta}{2}\mathbf{u}^T\mathbf{Q}_{j\,s}\big)\frac{\mathbf{G}^T_i}{4}$$

As defined in Appendix A.2, $\mathbf{G}$ relates local-frame angular velocities to time derivatives of quaternions through $\quad = \mathbf{G}\theta$.

Appendix D

Transformation of Multivariate Cubic Polynomials

In Section 5.1.2 it is explained that, in order to do force interpolation, I have opted for elastic models for which the forces can be expressed as polynomial functions in function of the degrees of freedom. In the case of the reduced StVK model (Section 5.1.4) I have shown that the elastic force vector $\mathbf{R}(\mathbf{q})$ is a multivariate cubic polynomial with vector coefficients $\mathbf{P}^i, \mathbf{Q}^{ij}, \mathbf{S}^{ijk} \in \mathbb{R}^r$. Note that, when using linear modal analysis to compute these coefficients for each of the dynamic morph targets $\mathbf{R}_i(\mathbf{q}_i)$ using Eqn. (5.9), the computed polynomial $\mathbf{R}_i(\mathbf{q}_i)$ is a function of the displacement $\mathbf{q}_i$ from rest pose p_i. Given these polynomial coefficients, one can compute a new set of coefficients for the cubic polynomial $\bar{\mathbf{R}}_i(\mathbf{q})$ that is in function of the degrees of freedom $\mathbf{q}$ of the system. By substituting $\mathbf{q}_i$ with $(\mathbf{q} + \Delta\mathbf{q})$ one can find:

$$\bar{\mathbf{R}}(\mathbf{q}) = \bar{\mathbf{T}} + \bar{\mathbf{P}}^i \Delta q_i + \bar{\mathbf{Q}}^{ij} \Delta q_i \Delta q_j + \bar{\mathbf{S}}^{ijk} \Delta q_i \Delta q_j \Delta q_k \tag{D.1}$$

with

$$\begin{aligned}
\bar{\mathbf{T}} &= \mathbf{P}^i \Delta q_i + \mathbf{Q}^{ij} \Delta q_i \Delta q_j + \mathbf{S}^{ijk} \Delta q_i \Delta q_j \Delta q_k \\
\bar{\mathbf{P}}^l &= \mathbf{P}^l + (\mathbf{Q}^{li} + \mathbf{Q}^{il}) \Delta q_i + (\mathbf{S}^{lij} + \mathbf{S}^{ilj} + \mathbf{S}^{ijl}) \Delta q_i \Delta q_j \\
\bar{\mathbf{Q}}^{kl} &= \mathbf{Q}^{kl} + (\mathbf{S}^{lik} + \mathbf{S}^{ilk} + \mathbf{S}^{ikl}) \Delta q_i \\
\bar{\mathbf{S}}^{klm} &= \mathbf{S}^{klm},
\end{aligned}$$

where $\bar{\mathbf{a}} = \begin{bmatrix} \bar{\mathbf{T}} & \bar{\mathbf{P}}^i & \bar{\mathbf{Q}}^{ij} & \bar{\mathbf{S}}^{ijk} \end{bmatrix}$ are the new vector coefficients that define $\bar{\mathbf{R}}_i(\mathbf{q})$ and $\Delta\mathbf{q}$ is the displacement between a reference rest pose and the rest pose p_i of a morph target. The new coefficients are then eventually used to solve (5.3) for the RBF weights.

Appendix E

Code Snippets

Code Snippet E.1 Routine to update two pixel buffers (PBO) from texture memory. The PBOs can then be interpreted as a vertex and normals buffer (VBO)

```
void HighResRenderMesh::updateVBOfromTextures(FramebufferObject* fb,
  const TextureRef& positionTexture, const TextureRef& normalTexture)
{
  // read the vertex data back from framebuffer-attached texture into the PBO
  if (!positionTexture.isNull())
  {
    fb->AttachTexture(GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D,
                      positionTexture->openGLID());
    glReadBuffer(GL_COLOR_ATTACHMENT0_EXT);
    fb->IsValid();
    debugAssertGLOk();
    glBindBuffer(GL_PIXEL_PACK_BUFFER_ARB, VBOs[POSITION]);
    debugAssertGLOk();
    glReadPixels(0, 0, pos_tex_height, pos_tex_width,
                 GL_RGBA /*BGRA*/, GL_FLOAT, 0);
    debugAssertGLOk();
  }

  // read the normal data back from framebuffer-attached texture into the PBO
  if (!normalTexture.isNull())
  {
    fb->AttachTexture(GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D,
                      normalTexture->openGLID());
    glReadBuffer(GL_COLOR_ATTACHMENT0_EXT);
    fb->IsValid();
    debugAssertGLOk();
    glBindBuffer(GL_PIXEL_PACK_BUFFER_ARB, VBOs[NORMAL]);
    debugAssertGLOk();
    glReadPixels(0, 0, pos_tex_height, pos_tex_width,
                 GL_RGBA /*BGRA*/, GL_FLOAT, 0);
    debugAssertGLOk();
  }

  // Unbind
  glBindBuffer(GL_PIXEL_PACK_BUFFER_ARB, 0);
  debugAssertGLOk();
}
```

Code Snippet E.2 CPU driver code for sparse matrix multiply. Two passes on the GPU are invoked with the `Compute()` call.

```
/**
* Compute sparse Kx product. Does *not* write alpha.
*
* @param x             x, an RGB(A) texture
* @param y             y, result buffer, an RGB(A) buffer
* @param tempbuffer    z, tempbuffer, should match the format of y (default RGBA)
*/
template <typename model_type>
  void compute_sparse_product(model_type& model,
  texture_pointer* A, texture_pointer x, texture_pointer y, texture_pointer tempbuffer)
{
  shared_ptr<GPUOps> gpu = model.m_gpu;
  shared_ptr<FramebufferObject> fbo = gpu->get_fbo();

  // Update the boundary information
  model.m_boundaryops->update_boundaries(model, x);
  DebugTexture(fbo, x);

  if (!tempbuffer)
    tempbuffer = gpu->m_tempbuffer2;

  // First pass, 3 neighbors and self
  tempbuffer->Attach(get_pointer(fbo), GL_COLOR_ATTACHMENT0_EXT);

  gpu->Ax1->SetTextureParameter("A00", A[0]->Texture());
  gpu->Ax1->SetTextureParameter("A01", A[1]->Texture());
  gpu->Ax1->SetTextureParameter("A02", A[2]->Texture());
  gpu->Ax1->SetTextureParameter("A20", A[3]->Texture());
  gpu->Ax1->SetTextureParameter("A21", A[4]->Texture());
  gpu->Ax1->SetTextureParameter("A22", A[5]->Texture());
  gpu->Ax1->SetTextureParameter("A30", A[6]->Texture());
  gpu->Ax1->SetTextureParameter("A31", A[7]->Texture());
  gpu->Ax1->SetTextureParameter("A32", A[8]->Texture());
  gpu->Ax1->SetTextureParameter("A80", A[18]->Texture());
  gpu->Ax1->SetTextureParameter("A81", A[19]->Texture());
  gpu->Ax1->SetTextureParameter("A82", A[20]->Texture());

  gpu->Ax1->SetTextureParameter("x", x->Texture());
  gpu->Ax1->SetMesh(model.GetParameterizedMesh());
  gpu->Ax1->Compute();

  tempbuffer->FastUnAttach();

  // Second pass, 3 neighbors and tempself
  y->Attach(get_pointer(fbo), GL_COLOR_ATTACHMENT0_EXT);

  gpu->Ax2->SetTextureParameter("A40", A[9]->Texture());
  gpu->Ax2->SetTextureParameter("A41", A[10]->Texture());
  gpu->Ax2->SetTextureParameter("A42", A[11]->Texture());
  gpu->Ax2->SetTextureParameter("A60", A[12]->Texture());
  gpu->Ax2->SetTextureParameter("A61", A[13]->Texture());
  gpu->Ax2->SetTextureParameter("A62", A[14]->Texture());
  gpu->Ax2->SetTextureParameter("A70", A[15]->Texture());
  gpu->Ax2->SetTextureParameter("A71", A[16]->Texture());
  gpu->Ax2->SetTextureParameter("A72", A[17]->Texture());

  gpu->Ax2->SetTextureParameter("x", x->Texture());
  gpu->Ax2->SetTextureParameter("tempy", tempbuffer->Texture());
  gpu->Ax2->SetMesh(model.GetParameterizedMesh());
  gpu->Ax2->Compute();
}
```

Code Snippet E.3 Set up texture matrix for projection of contact domain to D2T atlas and render into D2T atlas.

```
void ContactComputePolicy::ComputePolicy(const Matrix4 & contactCamMVP)
{
  // load contact camera matrices
  glMatrixMode(GL_TEXTURE);

  static Matrix4 bias(
      0.5f, 0.0f, 0.0f, 0.5f,
      0.0f, 0.5f, 0.0f, 0.5f,
      0.0f, 0.0f, 0.5f, 0.5f - .000001f,
      0.0f, 0.0f, 0.0f, 1.0f);

  glLoadMatrix(m_bias);
  glMultMatrix(contactCamMVP);

  CheckErrorsGL("Loaded contact camera matrices");

  // Render into D2T atlas
  m_mesh->RenderNearContactToAtlas(contact->Point(m_numobj), m_normal);
}
```

Fragment Program E.1 Compute $\mathbf{Ax} = \mathbf{A}_l\mathbf{x} + \mathbf{A}_r\mathbf{x}$ with D2T mapped sparse matrix in two passes. The intermediary result from `Ax1()` is passed on to `Ax2()` as input in the second pass.

```
#define SAMPLER samplerRECT

float3 value3(SAMPLER sampler, float2 offset)
{ return texRECT(sampler, offset).xyz; }

float3 Ax(SAMPLER A0, SAMPLER A1, SAMPLER A2, float3 x, float2 coord)
{
    float3 y;
    y = mul( float3x3(
                value3(A0, coord),
                value3(A1, coord),
                value3(A2, coord)),
             x);
    return y;
}


void Ax1(
in float2 coord : WPOS,
uniform SAMPLER x,
uniform SAMPLER A00, uniform SAMPLER A01, uniform SAMPLER A02,
uniform SAMPLER A20, uniform SAMPLER A21, uniform SAMPLER A22,
uniform SAMPLER A30, uniform SAMPLER A31, uniform SAMPLER A32,
uniform SAMPLER A80, uniform SAMPLER A81, uniform SAMPLER A82,
    out float3 result         : COLOR0)
{
    float3 x0  = value3(x, coord + float2(0.0, 1.0));
    float3 x2  = value3(x, coord + float2(1.0, 0.0));
    float3 x3  = value3(x, coord + float2(1.0, -1.0));
    float3 x8  = value3(x, coord);

    result  = Ax(A00, A01, A02, x0, coord);
    result += Ax(A20, A21, A22, x2, coord);
    result += Ax(A30, A31, A32, x3, coord);
    result += Ax(A80, A81, A82, x8, coord);
}
```

Fragment Program E.2 Compute $\mathbf{Ax} = y + \mathbf{A}_r\mathbf{x}$ with D2T mapped. The intermediary result from `Ax1()` is passed as input to `Ax2()`.

```
void Ax2(
in float2 coord : WPOS,
uniform SAMPLER x, uniform SAMPLER tempy,
uniform SAMPLER A40, uniform SAMPLER A41, uniform SAMPLER A42,
uniform SAMPLER A60, uniform SAMPLER A61, uniform SAMPLER A62,
uniform SAMPLER A70, uniform SAMPLER A71, uniform SAMPLER A72,
    out float3 result         : COLOR0)
{
    float3 x4  = value3(x, coord + float2(0.0, -1.0));
    float3 x6  = value3(x, coord + float2(-1.0, 0.0));
    float3 x7  = value3(x, coord + float2(-1.0, 1.0));

    result  = value3(tempy, coord);
    result += Ax(A40, A41, A42, x4, coord);
    result += Ax(A60, A61, A62, x6, coord);
    result += Ax(A70, A71, A72, x7, coord);
}
```

Fragment Program E.3 Rasterize distance to eye.

```
void main(
    float4 pos          : WPOS,
    float4 eyepos       : TEXCOORD0,
    float4 tidpos       : TEXCOORD1,
    uniform samplerRECT triangleidmap : TEX0,
    out float3 result   : COLOR0
)
{
    // Copy the triangle ID to green
    result.g = value(triangleidmap, tidpos.xy);

    // transfer depth (with and without perspective divide)
    // z component of eye space position is distance to the eye
    result.rb = eyepos.zw;
}
```

Fragment Program E.4 Compute per-texel depth differences.

```
void main(
    in float2 coord   : WPOS,
    uniform samplerRECT texture1,
    uniform samplerRECT texture2)
{
   // Subtract texture values and copy to red
   float2 val1 = f2texRECT(texture1, coord.xy);
   float2 val2 = f2texRECT(texture2, coord.xy);
   result.r = val1.r - val2.r;

   //Copy triangle ID to green and blue
   result.g = val1.g;
   result.b = val2.g;
}
```

Fragment Program E.5 Tag colliding texels in the D2T by transferring the collision data from D with the appropriate mapping and with triangle checking.

```
void tagcontactobj1(  // code for object 1
    in float2 coord                         : WPOS, in float2 texcoord               : TEXCOORD0,
    uniform float3   lowresnormal,
    uniform samplerRECT pdtexture, uniform samplerRECT trianglemap,
    out TYPE result                    : COLOR0)
{
    float3 pd = value3(pdtexture, texcoord);
    float triangle_id = value(trianglemap, coord);
    result = 0.0;
    // compare triangle ID and penetration depth.
    // Note: the triangle ID for object 1 is stored
    //       in the green component of pd
    if ((pd.r > 0.0) && (abs(pd.g - triangle_id) < 0.00001))
    {
        //store inwards lowres normal
        result.xyz = lowresnormal;
        //store penetration depth
        result.a = pd.x;
    }
    else { discard; }
}

void tagcontactobj2(  // code for object 2
    in float2 coord                         : WPOS, in float2 texcoord               : TEXCOORD0,
    uniform float3   lowresnormal,
    uniform samplerRECT pdtexture, uniform samplerRECT trianglemap,
    out TYPE result                    : COLOR0)
{
    float3 pd = value3(pdtexture, texcoord);
    float triangle_id = value(trianglemap, coord);
    result = 0.0;
    // compare triangle ID and penetration depth.
    // Note: the triangle ID for object 2 is stored
    //       in the blue component of pd
    if ((pd.r > 0.0) && (abs(pd.b - triangle_id) < 0.00001))
    {
        //store inwards lowres normal
        result.xyz = lowresnormal;
        //store penetration depth
        result.a = pd.x;
    }
    else { discard; }
}
```

Fragment Program E.6 Generate normal map by sampling of each D2T texel neighborhood.

```
void generate_normals(
    in float2           coord : WPOS,
    uniform samplerRECT bodypos,
    out float3 normal : COLOR0
    )
{
// fetch body position from position texture
float3 pos = value3(bodypos, coord);

float3 up = value3(bodypos, coord + float2(0,1)) - pos;
float3 down = value3(bodypos, coord + float2(0,-1)) - pos;
float3 left = value3(bodypos, coord + float2(-1,0)) - pos;
float3 right = value3(bodypos, coord + float2(1,0)) - pos;

float3 upright = value3(bodypos, coord + float2(1,1)) - pos;
float3 downright = value3(bodypos, coord + float2(1,-1)) - pos;
float3 upleft = value3(bodypos, coord + float2(-1,1)) - pos;
float3 downleft = value3(bodypos, coord + float2(-1,-1)) - pos;

float3 norm = (float3)0;
norm += normalize(cross(up, left));
norm += normalize(cross(left, down));
norm += normalize(cross(down, right));
norm += normalize(cross(right, up));

norm += normalize(cross(upright, upleft));
norm += normalize(cross(upleft, downleft));
norm += normalize(cross(downleft, downright));
norm += normalize(cross(downright, upright));

    normalize(norm);

    normal = norm;
}
```

Vertex Program E.1 Transform position to eye space.

```
void main(
    float4 pos          : POSITION,
    in  float4 tin      : TEXCOORD0,
    out float4 eyepos   : TEXCOORD0,
    out float4 tidpos   : TEXCOORD1,
    out float4 clippos  : POSITION
)
{
    eyepos = mul(glstate.matrix.modelview[0], pos);
    tidpos = tin;
    clippos = mul(glstate.matrix.mvp, pos);
}
```

Bibliography

[AOW+08] Bart Adams, Maks Ovsjanikov, Michael Wand, Hans-Peter Seidel, and Leonidas J Guibas. Meshless modeling of deformable shapes and their motion. *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2008.

[Bar84] A Barr. Global and local deformations of solid primitives. *Proc. of ACM SIGGRAPH*, 1984.

[Bar96] D Baraff. Linear-time dynamics using lagrange multipliers. *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 137 146, 1996.

[BBK05] M Botsch, D Bommes, and L Kobbelt. Efficient linear system solvers for mesh processing. *IMA Mathematics of Surfaces XI, Lecture Notes in Computer Science*, 3604:62 83, 2005.

[BFA02] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. 2002.

[BJ05] J Barbic and Doug L James. Real-time subspace integration of st. venant-kirchhoff deformable models. 2005.

[BKSS90] N Beckmann, H Kriegel, R Schneider, and B Seeger. The r*-tree: an efficient and robust access method for points and rectangles. *Proceedings of ACM SIGMOD*, 1990.

[BMWG07] M Bergou, S Mathur, M Wardetzky, and E Grinspun. Tracks: toward directable thin shells. *ACM Transactions on Graphics (TOG)*, 2007.

[BNC96] Morten Bro-Nielsen and Stephane Cotin. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. *Computer Graphics Forum*, 15(3), 1996.

[Bul08] Bullet physics library. 2008.

[BW92] David Baraff and Andrew Witkin. Dynamic simulation of non-penetrating flexible bodies. 1992.

[BW98] David Baraff and Andrew Witkin. Large steps in cloth simulation. *Proc. of ACM SIGGRAPH*, 1998.

[Cap04] S Capell. *Interactive Character Animation Using Dynamic Elastic Simulation*. Phd thesis, 2004.

[CBC+01] JC Carr, RK Beatson, JB Cherrie, TJ Mitchell, WR Fright, BC McCallum, and TR Evans. Reconstruction and representation of 3d objects with radial basis functions. 2001.

[CBC+05] S Capell, M Burkhart, B Curless, T Duchamp, and Z Popovic. Physically based rigging for deformable characters. *Proc. of Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, 2005.

[CGC+02a] S Capell, S Green, B Curless, T Duchamp, and Z Popovic. Interactive skeleton-driven dynamic deformations. *Proc. of ACM SIGGRAPH*, 2002.

[CGC+02b] S Capell, S Green, B Curless, T Duchamp, and Z Popovic. A multiresolution framework for dynamic deformations. *Proc. of ACM SIGGRAPH Symposium on Computer Animation*, 2002.

[CHP89] John E Chadwick, David R Haumann, and Richard E Parent. Layered construction for deformable animated characters. 1989.

[CK05] M Choi and H Ko. Modal warping: real-time simulation of large rotational deformation and manipulation. *IEEE Transactions on Visualization and Computer Graphics*, 2005.

[COM] Co-me: Computer aided and image guided medical interventions. `http://co-me.ch/`.

[COM98] J Cohen, M Olano, and D Manocha. Appearance-preserving simplification. 1998.

[Coq90] S Coquillart. Extended free-form deformation: a sculpturing tool for 3d geometric modeling. *Proceedings of ACM SIGGRAPH*, 1990.

[CP03] M Cline and D Pai. Post-stabilization for rigid body simulation with contact and constraints. pages 3744 3751, 2003.

[CUD07] Cuda programming guide. 2007.

[CW05] Fehmi Cirak and Matthew West. Decomposition contact response (dcr) for explicit finite element dynamics. *International Journal for Numerical Methods in Engineering*, 64(8), 2005.

[CZ92] D Chen and D Zeltzer. Pump it up: computer animation of a biomechanically based model of muscle using the finite element method. *ACM SIGGRAPH Computer Graphics*, 1992.

[DAK04] Christian Duriez, Claude Andriot, and Abderrahmane Kheddar. Signorini's contact model for deformable objects in haptic simulations. *Proc. of IEEE/RSJ IROS*, 2004.

[DCKY02] F Dong, G J Clapworthy, M A Krokos, and J Yao. An anatomy-based approach to human muscle modeling and deformation. *IEEE Trans. on Visualization and Computer Graphics*, 8(2), 2002.

[DDCB01] G Debunne, M Desbrun, M P Cani, and A H Barr. Dynamic real-time deformations using space and time adaptive sampling. *Proc. of ACM SIGGRAPH*, 2001.

[DGLN04] Timothy A. Davis, John R. Gilbert, Stefan I. Larimore, and Esmond G. Ng. A column approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, 30(3):353 376, 2004.

[Die06] James Diebel. Representing attitude: Euler angles, quaternions, and rotation vectors. Technical report, 2006.

[EDS05] Kenny Erleben, Henrik Dohlmann, and Jon Sporring. The adaptive thin shell tetrahedral mesh. *Journal of WSCG*, pages 17 24, 2005.

[EL00] Stephen A Ehmann and Ming C Lin. Accelerated proximity queries between convex polyhedra by multi-level voronoi marching. 2000.

[EL01] Stephen A Ehmann and Ming C Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. 2001.

[Erl04] Kenny Erleben. *Stable, Robust and Versatile Multibody Dynamics Animation*. Phd thesis, 2004.

[Fea87] R Featherstone. *Robot Dynamics Algorithms*. Book, 1987.

[Fel00] C A Felippa. A systematic approach to the element-independent corotational dynamics of finite elements. Technical report, 2000.

[FP03] Anthony C. Fang and Nancy S. Pollard. Efficient synthesis of physically valid human motion. *ACM Trans. Graph.*, 22(3):417 426, 2003.

[Gas98] Marie-Paule Gascuel. Layered deformable models with implicit surfaces. 1998.

[GGK06] A Gress, M Guthe, and R Klein. Gpu-based collision detection for deformable parameterized surfaces. *Proc. of Eurographics, Computer Graphics Forum*, 25(3), 2006.

[GHZ99] L Guibas, D Hsu, and L Zhang. H-walk: hierarchical distance computation for moving convex bodies. *Proceedings of the fifteenth annual symposium on Computational Geometry*, 1999.

[GJK88] E Gilbert, D Johnson, and S Keerthi. A fast procedure for computing the distance between complex objectsin three-dimensional space. *Robotics and Automation*, 1988.

[GKS02] E Grinspun, P Krysl, and P Schröder. Charms: A simple framework for adaptive simulation. *Proc. of ACM SIGGRAPH*, 2002.

[GL96] Gene H Golub and Charles F Van Loan. *Matrix Computations*. Book, 1996.

[GLM96] S Gottschalk, M Lin, and D Manocha. Obbtree: a hierarchical structure for rapid interference detection. *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996.

[GM97] S F Gibson and B V Mirtich. A survey of deformable modeling in computer graphics. Technical report, 1997.

[Got00] S Gottschalk. Collision queries using oriented bounding boxes. 2000.

[GPS02] Herbert Goldstein, Charles Poole, and John Safko. *Classical Mechanics, 3rd Ed.* Book, 2002.

[GRLM03] N Govindaraju, S Redon, M Lin, and D Manocha. Cullide: interactive collision detection between complex models in large environments using graphics hardware. *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware*, 2003.

[GRPS07] T Goktekin, J Reisch, D Peachey, and A Shah. An effects recipe for rolling a dough, cracking an egg and pouring a sauce. *International Conference on Computer Graphics and Interactive Techniques (ACM SIGGRAPH)*, 2007.

[GTO+07] N Galoppo, S Tekin, M Otaduy, M Gross, and M Lin. Interactive haptic rendering of high-resolution deformable objects. *Lecture Notes in Computer Science*, 2007.

[GTT89] Jean-Paul Gourret, Nadia Magnenat Thalmann, and Daniel Thalmann. Simulation of object and human skin deformations in a grasping task. 1989.

[GW05] Z Guo and K C Wong. Skinning with deformable chunks. *Proc. of Eurographics, Computer Graphics Forum*, 24(3):373 382, 2005.

[Har05] Mark Harris. Mapping computational concepts to gpus. chapter 31. 2005.

[Hav08] Havok. Havok physics engine. 2008.

[HES03] M Hauth, O Etzmuss, and W Strasser. Analysis of numerical methods for the simulation of deformable models. *The Visual Computer*, 2003.

[HFS03] Gentaro Hirota, Susan Fisher, and Andrei State. An improved finite element contact model for anatomical simulations. *The Visual Computer*, 19(5), 2003.

[HGB06] J O Hallquist, GL Gourdreau, and D J Benson. Tetgen. a quality tetrahedral mesh generator and three-dimensional delaunay triangulator., 2006.

[HS04] M Hauth and W Strasser. Corotational simulation of deformable solids. *Proc. of WSCG*, 2004.

[HSO03] Kris K Hauser, Chen Shen, and James F O'Brien. Interactive deformation using modal analysis with constraints. *Proc. of Graphics Interface*, 2003.

[Hub95] P Hubbard. Collision detection for interactive graphics applications. *Visualization and Computer Graphics*, 1995.

[IC87] P Isaacs and M Cohen. Controlling dynamic simulation with kinematic constraints. *Proceedings of the 14th annual conference on Computer Graphics and Interactive Techniques*, 1987.

[ISF07] G Irving, C Schroeder, and R Fedkiw. Volume conserving finite element simulations of deformable models. *International Conference on Computer Graphics and Interactive Techniques (ACM SIGGRAPH)*, 2007.

[ITF04] Geoffrey Irving, Joseph Teran, and Ron Fedkiw. Invertible finite elements for robust simulation of large deformation. 2004.

[IZLM01] K Hoff III, A Zaferakis, M Lin, and D Manocha. Fast and simple 2d geometric proximity queries using graphics hardware. *Proceedings of the 2001 symposium on Interactive 3D graphics*, 2001.

[JF03] Doug L James and Kayvon Fatahalian. Precomputing interactive dynamic deformable scenes. 2003.

[JP99] Doug L James and Dinesh K Pai. Artdefo: accurate real time deformable objects. 1999.

[JP02] Doug L James and Dinesh K Pai. Dyrt: Dynamic response textures for real-time deformation simulation with graphics hardware. 2002.

[JP04] D James and D Pai. Bd-tree: output-sensitive collision detection for reduced deformable models. *ACM Transactions on Graphics (TOG)*, 2004.

[JT05] Doug L James and Christopher D Twigg. Skinning mesh animations. 2005.

[Kar03] George Karypis. Multi-constraint mesh partitioning for contact/impact computations. page 56, 2003.

[KCŽC07] L Kavan, S Collins, J Žára, and CO'Sullivan. Skinning with dual quaternions. *Proc. of the 2007 Symposium on Interactive 3D Graphics*, 2007.

[KEP05] Danny M Kaufman, Timothy Edmunds, and Dinesh K Pai. Fast frictional dynamics for rigid bodies. *Proc. of ACM SIGGRAPH*, 2005.

[KGCvB96] R Koch, M Gross, F Carls, and D von Büren. Simulating facial surgery using finite element models. *Proceedings of the 23rd annual conference on Computer Graphics and Interactive Techniques (ACM SIGGRAPH)*, 1996.

[KHMS98] J Klosowski, M Held, J Mitchell, and H Sowizral. Efficient collision detection using bounding volume hierarchies of k-dops. *Visualization and Computer Graphics*, 1998.

[KJP02] P Kry, D L James, and D K Pai. Eigenskin: Real time large deformation character skinning in hardware. 2002.

[KKA05] R Kondo, T Kanai, and K Anjyo. Directable animation of elastic objects. *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005.

[KM04] T Kurihara and N Miyata. Modeling deformable human hands from medical images. *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2004.

[KO88] N Kikuchi and J T Oden. *Contact Problems in Elasticity: A Study of Variational Inequalities and Finite Element Methods*. Book, 1988.

[Kom88] K Komatsu. Human skin model capable of natural shape variation. *The Visual Computer*, 1988.

[KZ05] L Kavan and J Zara. Spherical blend skinning: A real-time deformation of articulated models. 2005.

[Las87] J Lasseter. Principles of traditional animation applied to 3d computer animation. *ACM SIGGRAPH Computer Graphics*, 1987.

[LC91] M Lin and J Canny. A fast algorithm for incremental distance calculation. *Robotics and Automation*, 1991.

[LCF00] J P Lewis, Matt Cordner, and Nickson Fong. Pose space deformations: A unified approach to shape interpolation and skeleton-driven deformation. 2000.

[LCN99] J C Lombardo, M-P Cani, and F Neyret. Real-time collision detection for virtual surgery. *Proc. of Computer Animation*, 1999.

[LGLM00] E Larsen, S Gottschalk, M Lin, and D Manocha. Fast distance queries with rectangular swept sphere volumes. *Robotics and Automation*, 2000.

[LHP05] C Liu, A Hertzmann, and Z Popović. Learning physics-based motion style with nonlinear inverse optimization. *Proceedings of ACM SIGGRAPH 2005*, 2005.

[Lin93] M Lin. Efficient collision detection for animation and robotics. *cs.unc.edu*, 1993.

[LM04] Ming C Lin and D Manocha. Collision and proximity queries. *Handbook of Discrete and Computational Geometry*, 2004.

[LO08] Ming C Lin and Miguel A Otaduy. Haptic rendering: Foundations, algorithms, and applications. page 623, 2008.

[LT06] S Lee and D Terzopoulos. Heads up!: biomechanical modeling and neuromuscular control of the neck. *Proceedings of ACM SIGGRAPH 2006*, 2006.

[Mae06] G Maestri. Digital character animation 3. page 309, 2006.

[MDM+02] Matthias Müller, J Dorsey, L McMillan, R Jagnow, and B Cutler. Stable real-time deformations. *Proc. of ACM SIGGRAPH Symposium on Computer Animation*, 2002.

[MG03] A Mohr and M Gleicher. Building efficient, accurate character skins from examples. 2003.

[MG04] Matthias Müller and M Gross. Interactive virtual materials. *Proc. of Graphics Interface*, 2004.

[MHTG05] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. *Proc. of ACM SIGGRAPH*, 2005.

[Mir96] Brian V Mirtich. *Impulse-Based Dynamic Simulation of Rigid Body Systems*. Phd thesis, 1996.

[MT92] Dimitri Metaxas and Demetri Terzopoulos. Dynamic deformation of solid primitives with constraints. *Proc. of ACM SIGGRAPH*, 1992.

[MTLT88] N Magnenat-Thalmann, R Laperrière, and D Thalmann. Joint-dependent local deformations for hand animation and object grasping. *Proceedings on Graphics Interface*, pages 26 33, 1988.

[NMK+05] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxermann, and Mark Carlson. Physically based deformable models in computer graphics (state of the art report). *Eurographics STAR*, 2005.

[OBH02] J F O'Brien, A W Bargteil, and J K Hodgins. Graphical modeling and animation of ductile fracture. *Proc. of ACM SIGGRAPH*, pages 291 294, 2002.

[OGRG07] M Otaduy, D Germann, S Redon, and M Gross. Adaptive deformations with fast tight bounds. *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer Animation*, 2007.

[OH99] J O'Brien and J Hodgins. Graphical modeling and animation of brittle fracture. *Proceedings of the 26th annual conference on Computer Graphics and Interactive Techniques (ACM SIGGRAPH)]*, 1999.

[OJSL04] Miguel A Otaduy, Nitin Jain, Avneesh Sud, and Ming C Lin. Haptic display of interaction between textured models. 2004.

[Par82] F Parke. Parameterized models for facial animation. *Computer Graphics and Applications*, 1982.

[PDA01] Guillaume Picinbono, Hervé Delingette, and Nicholas Ayache. Non-linear and anisotropic elastic soft tissue models for medical simulation. *IEEE ICRA*, 2001.

[PH06] S I Park and J K Hodgins. Capturing and animating skin deformation in human motion. 2006.

[Phy08] Nvidia physx sdk. 2008.

[Pow87] MJD Powell. Radial basis functions for multivariate interpolation: a review. *Algorithms for Approximation*, 1987.

[PPG04] Mark Pauly, Dinesh K Pai, and Leonidas J Guibas. Quasi-rigid objects in contact. 2004.

[PW89] Alex Pentland and John Williams. Good vibrations: Modal dynamics for graphics and animation. volume 23, pages 215 222, 1989.

[PW99] Z Popović and A Witkin. Physically based motion transformation. *Proceedings of the 26th annual conference on Computer Graphics and Interactive Techniques (ACM SIGGRAPH)]*, 1999.

[Qui94] S Quinlan. Efficient distance computation between non-convex objects. *Robotics and Automation*, 1994.

[RLN06] T Rhee, J Lewis, and U Neumann. Real-time weighted pose-space deformation on the gpu. *Computer Graphics Forum*, 2006.

[SCS+08] L Seiler, D Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jermey Sugerman, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, and Pat Hanrahan. Larrabee: A many-core x86 architecture for visual computing. *Proc. of ACM SIGGRAPH*, 2008.

[SG06] Olaf Schenk and Klaus Gärtner. On fast factorization pivoting methods for symmetric indefinite systems. *Elec. Trans. Numer. Anal.*, 23:158 179, 2006.

[Sha89] Ahmed A Shabana. *Dynamics of Multibody Systems*. Book, 1989.

[Sha07] A Shah. Course 6: Anyone can cook: inside ratatouille's kitchen. *International Conference on Computer Graphics and Interactive Techniques (ACM SIGGRAPH)*, 2007.

[She94] Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, 1994.

[SIC01] P Sloan, C Rose III, and M Cohen. Shape by example. *Proceedings of the 2001 symposium on Interactive 3D graphics*, 2001.

[SK03] Peng Song and Vijay Kumar. Distributed compliant model for efficient dynamic simulation of systems with frictional contacts. 2003.

[SKP08] Shinjiro Sueda, Andrew Kaufman, and Dinesh K Pai. Musculotendon simulation for hand animation. *Proc. of ACM SIGGRAPH*, 2008.

[SNF05] E Sifakis, I Neverov, and R Fedkiw. Automatic determination of facial muscle activations from sparse motion capture marker data. *Proceedings of ACM SIGGRAPH 2005*, 2005.

[SOH99] Robert W Sumner, James F O'Brien, and Jessica K Hodgins. Animating sand, mud, and snow. *Computer Graphics Forum*, 18(1), 1999.

[SP86] T Sederberg and S Parry. Free-form deformation of solid geometric models. *ACM SIGGRAPH Computer Graphics*, 1986.

[SPCM97] F Scheepers, R Parent, W Carlson, and S May. Anatomy-based modeling of the human musculature. *Proceedings of the 24th annual conference on Computer Graphics and Interactive Techniques (ACM SIGGRAPH)*, 1997.

[SSGH01] P V Sander, J Snyder, S J Gortler, and H Hoppe. Texture mapping progressive meshes. *Proc. of ACM SIGGRAPH*, 2001.

[Sta03] Jos Stam. Flow on surfaces of arbitrary topology. 2003.

[TKH+05] Matthias Teschner, S Kimmerle, B Heidelberger, G Zachmann, L Raghupathi, A Furhmann, M-P Cani, F Faure, N Magnenat-Thalmann, W Strasser, and P Volino. Collision detection for deformable objects. *Computer Graphics Forum*, 24(1), 2005.

[TL94] G Turk and M Levoy. Zippered polygon meshes from range images. *Proceedings of the 21st annual conference on Computer Graphics and Interactive Techniques*, 1994.

[TPBF87] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. *Proc. of ACM SIGGRAPH*, 1987.

[TSIF05] Joseph Teran, Eftychios Sifakis, Geoffrey Irving, and Ron Fedkiw. Robust quasistatic finite elements and flesh simulation. 2005.

[TT93] Russell Turner and Daniel Thalmann. The elastic surface layer model for animate character construction. 1993.

[TW88] Demetri Terzopoulos and Andrew Witkin. Physically based models with rigid and deformable components. *IEEE Computer Graphics and Applications*, 8(6), 1988.

[WG97] J Wilhelms and A Van Gelder. Anatomically based modeling. *Proceedings of the 24th annual conference on Computer Graphics and Interactive Techniques (ACM SIGGRAPH)*, 1997.

[WPP07] R Wang, K Pulli, and J Popović. Real-time enveloping with rotational regression. *International Conference on Computer Graphics and Interactive Techniques (ACM SIGGRAPH)*, 2007.

[WRM05] Pawel Wrotek, Alexander Rice, and Morgan McGuire. Real-time collision deformations using graphics hardware. *Journal of Graphics Tools*, 10(5), 2005.

[WSLG07] O Weber, O Sorkine, Y Lipman, and C Gotsman. Context-aware skeletal shape deformation. *Computer Graphics Forum*, 2007.

[WTF06] R Weinstein, J Teran, and R Fedkiw. Dynamic simulation of articulated rigid bodies with contact and collision. *IEEE Trans. on Visualization and Computer Graphics*, 12(3), 2006.

[WVS90] P Wriggers, T Vu Van, and E Stein. Finite element formulation of large deformation impact-contact problems with friction. *Computers & Structures*, 37(3), 1990.

[WW90] Andrew Witkin and William Welch. Fast animation and control of nonrigid structures. *Computer Graphics (SIGGRAPH 90 Proceedings)*, 24(4):243 252, 1990.

[ZC99] Yan Zhuang and John Canny. Real-time simulation of physically realistic global deformation. *Proc. of IEEE Visualization*, 1999.

[ZCCD04] VB Zordan, B Celly, B Chiu, and PC DiLorenzo. Breathe easy: model and control of simulated respiration for animation. *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 29 37, 2004.

[ZMCF05] Victor B Zordan, Anna Majkowska, Bill Chiu, and Matthew Fast. Dynamics response for motion capture animation. 2005.

[ZSS97] D Zorin, P Schröder, and W Sweldens. Interactive multiresolution mesh editing. 1997.