

Interactive Character Animation Using Dynamic Elastic Simulation

Steve Capell

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Washington

2004

Program Authorized to Offer Degree: Department of Computer Science and Engineering

UMI Number: 3131129

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.



UMI Microform 3131129

Copyright 2004 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

University of Washington

Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

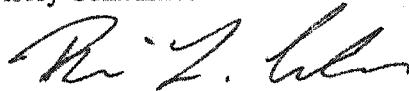
Steve Capell

and have found that it is complete and satisfactory in all respects,

and that any and all revisions required by the final

examining committee have been made.

Co-Chairs of Supervisory Committee:



Brian Curless



Zoran Popović

Reading Committee:



Brian Curless



Tom Ducharme



Zoran Popović

Date:



In presenting this dissertation in partial fulfillment of the requirements for the Doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to Bell and Howell Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature 

Date April 7, 2004

University of Washington

Abstract

Interactive Character Animation Using Dynamic Elastic Simulation

by Steve Capell

Co-Chairs of Supervisory Committee:

Professor Brian Curless
Department of Computer Science and Engineering

Professor Zoran Popović
Department of Computer Science and Engineering

This dissertation describes a framework for interactively animating characters such as humans and animals based on dynamic elastic simulation. Using dynamic simulation, the secondary motion of the character's soft tissue is computed automatically, and the shape of the character reflects environmental influences not anticipated by the animator. To endow an elastic body with animation-friendly control mechanisms we unify dynamic elastic simulation, skeleton-driven deformation, and shape interpolation. We model a character as an elastic body simulated using the finite element method. For computational efficiency, we embed the object in a coarse subdivision volume. Subdivision provides topological flexibility, smooth deformations, and hierarchical structure for adaptive simulation. Skeletal control is made efficient by aligning the subdivision control lattice with the skeleton. In order to make the computation of elastic dynamics efficient enough for interactive applications, we introduce a new method of linearizing the nonlinear equations of elasticity dependent on the pose of the character. Our framework also provides a mechanism to control the shape of the character via abstract parameters. Because the shape of a character is determined by physical dynamics, it cannot simply be dictated as in traditional computer animation. Instead, we introduce forces to control the shape. Force-based shape control *guides* the shape of the character but is combined with other forces acting on the system and integrated into the dynamics. The result is a system that produces interactive animations with automatic secondary flesh dynamics, and at the same time gives the animator a large degree of control over the pose and shape of the character.

TABLE OF CONTENTS

List of Figures	iv
Chapter 1: Introduction	1
1.1 Overview	3
1.2 Contributions	4
Chapter 2: Related Work	7
2.1 Free-Form Deformation	7
2.2 Skeleton-Driven Deformation and Character Animation	8
2.3 Dynamic Deformation	9
2.4 Anatomical Modeling	10
2.5 Multiresolution and Subdivision-Based Simulation	11
Chapter 3: Finite Element Elasticity	13
3.1 Lagrangian Formulation	13
3.2 Kinetic Energy	15
3.3 Elastic Potential Energy	16
3.4 Gravity	17
3.5 Constraints	17
3.6 System of Equations	17
Chapter 4: Interactive Simulation Framework	19
4.1 Embedding	19
4.2 The Hierarchical Basis	23
4.3 Quasi-linearization	25

4.4	Numerical Integration	27
4.5	Solving the ODEs	28
4.6	Runtime Details	29
4.7	Adapting the Basis	29
4.8	Linear Subspace Constraints	30
4.9	Realtime Simulation	31
4.10	Results	32
Chapter 5:	Skeleton-Driven Deformation	38
5.1	Skeletal Constraints	39
5.2	Pose-Dependent Linearization	42
5.3	Bone Displacement Energy	49
5.4	Instrumenting Characters	49
5.5	Results	50
Chapter 6:	Shape Control Using Forces	54
6.1	Overview	55
6.2	Simulating Rig Forces	56
6.3	The Rigging Process	57
6.4	Rigging with Force Field Templates	58
6.5	Deriving Rigs from Surface Deformations	63
6.6	Adaptive Rigging	68
6.7	Retargetting Surface Deformation Rigs	69
6.8	Results	72
Chapter 7:	Conclusions and Future Work	77
7.1	Improving the Physical Model	77
7.2	Geometric Mechanical Components	78
7.3	Learning and Interpolating Physical Properties	79

Bibliography	81
Appendix A: Derivatives of Elastic Potential	90
Appendix B: Review of Trilinear Functions	92
Appendix C: Estimating the Rotation in a Displacement Field	94

LIST OF FIGURES

1.1	Comparison between traditional and elastic character animation	2
3.1	Decomposition of the state of a body into rest and displacement	14
4.1	Comparison among grid types	20
4.2	Domains parameterized by a cell complex	22
4.3	The hierarchical basis	24
4.4	Deformation of a sharp subdivision object	32
4.5	Simulation with planar constraints	33
4.6	Simulation with spatially varying material properties	33
4.7	Comparison between deformations using regular and subdivision grids	35
4.8	Adaptive simulation of an embedded dragon	36
5.1	A skeletal constraint aligned with a control lattice	40
5.2	Comparison between pose-dependent linearization schemes	45
5.3	Instrumentation of a character	51
5.4	Skeleton-driven animation results	52
5.5	Comparison among linear, nonlinear, and blended linear animation	52
6.1	Rigging with force field templates	62
6.2	Pose optimization	65
6.3	A surface deformation rig for a bent arm.	66
6.4	Forces interpolated to new poses	67
6.5	Input surfaces for rig transfer	72
6.6	Comparison between transferred arm bulge and original deformation	73

6.7	Comparison between transferred chest flex and original deformation	74
6.8	Animations demonstrating a chest flex rig	75
6.9	Adaptive simulation of the chest flex rig	76

ACKNOWLEDGMENTS

The department of Computer Science and Engineering at the University of Washington has been a wonderful place to work. I want to thank the faculty and staff, as well as my fellow students and friends, who made it such a supportive and lively environment.

My advisors and collaborators Brian Curless, Zoran Popović, Tom Duchamp, and Seth Green have had a significant influence on this work, contributing many ideas and insights. This dissertation subsumes and extends the work that appeared in two papers that we coauthored [18, 19]. I'm especially grateful for their enthusiasm, warmth, and generosity, which made our collaborations great fun. I feel very fortunate to have them as friends.

Without the influence of my parents Fred and Sue Capell I would not have such a fulfilling life. They taught me to buck the system, and have given me their love and support unconditionally. They also provided me with a gaggle of siblings who count among my closest friends and have provided me with a lot of inspiration. My family has been a continuous source of joy and comfort in my life.

My lovely wife and constant companion Rimli Sengupta continues to make life colorful in too many ways to count.

DEDICATION

To Gramma Briggs, whose boundless enthusiasm and determination has been a great source of inspiration for me.

Chapter 1

INTRODUCTION

Computer animation enjoys one chief advantage over traditional hand-drawn animation. By providing a modeling layer between the animator and the output images, computer animation enables the animator to express the animation more succinctly, while ignoring unnecessary details. This leads to a considerable improvement in animation quality because the efforts of the animator are spent working at a higher level of abstraction than that of ink on paper. The modeling layer turns the high-level guidance of the animator into frames of animation.

In the traditional computer animation pipeline, one form of modeling is in the *rigging* of virtual characters such as animals and humans [2]. The rigging process is analogous to setting up a puppet to be controlled by strings. After having been rigged, a character's shape can be controlled via a set of abstract parameters with meaningful names, like *lift left eyebrow* or *bend right knee*. For each keyframe, instead of having to position each vertex of the surface mesh, the animator needs only to set the values of the control parameters. Once the character has been rigged, shape transformations such as a smile can be reused throughout the lifetime of a character. But the process of rigging is quite difficult using current techniques. Much of this difficulty is due to the inherent complexity of realistic shape deformations. Changes in the apparent shape of an actual animal are due to the motion of underlying tissues, as well as physical forces.

Another form of modeling that has seen some success in computer animation is the use of physical laws to simulate realistic motion (e.g., [10, 11, 27, 28, 42, 80]). Physical simulation allows the automatic synthesis of effects that are difficult to animate otherwise, such as ballistic motion, fluid flow, and the sagging and vibration of tissue caused by gravity and locomotion. Ideally, the animator would be freed 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.

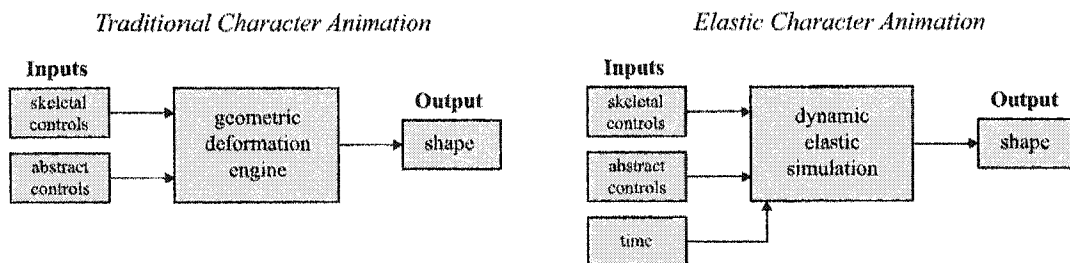


Figure 1.1 The diagram on the left represents a traditional character animation system. At the core is a geometric deformation engine which maps skeletal controls (such as joint angles) and abstract controls (such as *frown*) to the output shape. Animation is effected by varying the controls over time. The diagram on the right represents the system presented in this work. 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.

The modeling of motion is especially important for interactive animation systems such as video games. Since there is no animator available to respond to the user's input during an interactive session, it is only through automation that the computer can respond. And it is succinct models that make automation feasible. Physical simulation is a good example; when a virtual character with a fat belly is guided interactively through a virtual world, we would expect to see his belly bounce and sway in correlation with changes in momentum caused by the user's commands. An efficient way to achieve such motion would be to encode it in the abstraction of physical equations, from which it is conceptually straightforward for the computer to produce the appropriate motion.

In this work we bring together rigged character animation, physical simulation of flesh dynamics, and interactivity in a unified framework. In our framework, the motion of the character is dictated by both the physical model of the character and the rigging of the character. As noted by Zeltzer [90], the choice between direct control and simulation can be made at various levels of abstraction; one may choose to simulate atoms, neurons, muscles, etc. Our approach is to use physical simulation as it fits most naturally into the traditional animation pipeline. Rather than considering characters that are driven entirely by physical processes (and thus require elaborate control mechanisms), we consider characters that are controlled in roughly the same manner as traditional non-dynamic animated characters, but exhibit secondary dynamic behavior. As can be seen

in Figure 1.1, our system is based on elastic dynamics but supports the kinds of controls found in traditional character animation systems.

1.1 Overview

The characters in our system are defined as elastic solids. As such, their dynamic behavior is defined by physical equations that we simulate using the finite element method (FEM). We present the mathematical model and FEM approximation in Chapter 3.

The volumetric finite element mesh need only be specified coarsely, subject to the requirement that it encompass the geometric model on which simulation will be performed. This requirement is necessary in order to ensure complete integration over the interior of the object. In fact, as long as the interior of the object is well-defined, simulation of its elastic deformation is possible regardless of the surface representation or complexity.

The volumetric mesh that we use for simulation is not restricted to a regular grid; rather, it is comprised of elements such as tetrahedra and hexahedra, which may be smoothed using various subdivision schemes. This flexibility permits construction of meshes that conform better to the surface of the object, improving simulation quality. In addition, to support adaptive level of detail during simulation, we construct a hierarchical basis, which allows detail to be introduced or removed as needed. Our deformable body simulator is presented in Chapter 4.

At the center of traditional computer character animation is control via a skeleton in the form of a joint hierarchy. In our elastic simulation framework, a skeletal hierarchy drives the motion of line constraints representing bones. In order to incorporate these constraints efficiently, we align the edges of the volumetric mesh with the bones of the skeleton. To achieve interactive rates, we provide a novel method of approximating the nonlinear equations of elasticity with a pose-dependent linear system. Because one-dimensional line constraints do not fully capture the behavior of three-dimensional bones, we include a bone deformation energy in a neighborhood of the skeleton. Chapter 5 describes simulation using a skeletal constraint.

Beyond skeletal control, traditional computer animation provides more general mechanisms of shape control. Our approach is to use forces to effect the desired shape changes. Forces fit nicely into a physical simulation framework and have many other useful properties. They can be combined

easily, they need not be closely associated with a particular surface shape or representation, and they can be easily adapted to simulations at any level of resolution. Forces also present an abstraction of what really happens in animals, without requiring the animator to model every detail of animal anatomy—they allow the shape to change naturally since the forces and the inertia due to movement both affect the resulting shape in motion.

We provide tools for the user to design forces to effect the desired shape changes. Because editing the forces directly would be highly unintuitive, we provide a mechanism for the user to simply drag the surface with the mouse. The system automatically computes the underlying forces that deform the shape to meet the animators specification. Alternatively, forces can be derived from example surface deformations. Since the forces are not tightly coupled to the surface representation, they can be transferred to a completely different shape. Shape control using forces is described in Chapter 6.

1.2 Contributions

The main contribution of this work is a system for interactive character animation that is built on dynamic elastic simulation, yet provides the same kinds of control mechanisms as traditional character animation. Using this system we demonstrate interactive character animation that exhibits realistic secondary dynamics. In order to accomplish this goal, various subproblems were addressed, which we place into the three categories that appear below.

1.2.1 Elastic Deformation

Previous to this work, the embedding of objects had been used for the purpose of deformation under the name of free-form deformation (FFD) [31, 41, 50, 71, 73]. Our framework extends FFD to the domain of physically based dynamic deformations by combining embedding with FEM.¹ Taking advantage of the flexibility of subdivision solids as in the work of MacCracken et al. [50], our framework is easier than previous approaches to apply to complex characters, and produces more realistic deformations than methods based on regular grids of comparable complexity.

¹The work of Faloutsos et al. [31] involved dynamic but not physically based FFD.

As in previous work [27, 28, 33, 37], our simulator supports adaptive processing. Its novel feature is the use of precomputation to make the adaptation faster and simpler. Like Gortler and Cohen [33] and Grinspun et al. [37], we adapt by activating and deactivating basis functions. After precomputing the terms of the mass and stiffness matrices for a fixed number of levels of the hierarchy, adaptation simply requires rows and columns of the linear system to be added and removed.

Our elastic simulation framework has a few other novel features. We introduce a new method of linearizing the nonlinear equations of elasticity that is simpler than previous methods but provides similar results. We also extend to FEM the method of Baraff and Witkin [11] for applying position constraints directly within an iterative solver, avoiding the need to explicitly reparameterize the system or apply constraints using costly Lagrange multipliers.

1.2.2 Skeleton-Driven Deformation

Another area in which this work contributes is in the driving of an elastic simulation using a skeleton. We introduce a novel method of crafting the function space to make it computationally easier to enforce the skeletal constraint. A potential energy is introduced that causes these one-dimensional constraints to behave more like three-dimensional bones.

A key to interactivity is approximation of the full nonlinear equations of elasticity. A new method of linearizing the equations as a function of the pose of the character makes interactivity possible while maintaining realistic looking deformations that do not suffer from gross distortions. This pose-dependent linearity aids not only in dynamic simulation but also in computing static equilibrium states, and performing optimization.

1.2.3 Shape Control Using Forces

The idea of using forces as a form of rigging to effect shape change in dynamic elastic characters is another novel idea presented in this work. Other contributions include the efficient use of inverse-control for rigging and the determination of force rigs from examples. We introduce the transfer of physically based rigs from one model to another, drawing a correspondence between characters and transferring forces in a way that takes into consideration changes in scale.

More generally, this work provides a new abstract modeling layer for representing physically

based deformations. Rather than model internal anatomy directly, which seems like an unnecessary burden for computer animation, forces provide a mechanism to obtain realistic surface deformations without modeling internal structure.

Although this work focuses on interactive simulations, the ideas and techniques developed are also useful for traditional animations that are not interactive. Computation can be a bottleneck for any animation based on physical simulation, so the techniques used to improve efficiency are broadly applicable. More importantly, the idea of building character animation systems on a core of elastic simulation is a general one. This work is a step toward a future in which the benefits of physical simulation are available to animators, without sacrificing the control that is needed to achieve the desired results.

Chapter 2

RELATED WORK

Our work draws from and brings together a variety of related work in computer animation and other fields. This chapter provides a brief review of related previous work, and discusses how we build on previous work and in what ways our framework is different.

2.1 *Free-Form Deformation*

Early work on deformations focused on static techniques that enable the user to create individual deformed states of an object. These deformations could then be used as keyframes for animation. The introduction of free-form deformation (FFD) by Sederberg et al. made it possible to deform objects independent of their structure by embedding them in a regular grid [71]. Using FFD, a complex object can be deformed by positioning the control vertices of the coarse control grid. Hsu et al. provided a method of directly manipulating FFDs [41]. Instead of moving the control points the user could position any point on the object, making it easier to achieve the desired deformations. Other extensions to FFD include volume preservation [7, 39], and efforts to provide a more intuitive sculpting interface [26, 53]. MacCracken and Joy developed three-dimensional lattice subdivision (a.k.a. subdivision solids or subdivision volumes), an extension of Catmull-Clark subdivision surfaces, to ease the topological restrictions of FFD [50]. Their scheme allowed FFD using a non-regular control lattice. Another important extension to FFD was the introduction of dynamics by Faloutsos et al. [31]. Our framework builds on both of these extensions, using a flexible class of control lattices as in [50], and embedding objects in *dynamic* free-form lattices as in [31]. But unlike [31], where a diagonal stiffness matrix was used, we apply FEM to the theory of continuum elasticity to simulate the elastic dynamics of the embedded object.

2.2 *Skeleton-Driven Deformation and Character Animation*

One of the foundational techniques used in modern character animation is to drive the deformation of the surface via an underlying skeleton (a piecewise linear stick figure in the form of a joint hierarchy). This technique was introduced in the context of two-dimensional animation by Burtnyk and Wein [15]. In their framework the skeleton is surrounded by polygons. When the vertices of the stick figure are moved, the polygons deform; any image or curve represented in the coordinates of one of the polygons is thus deformed.

Skeleton-driven deformation was extended to three-dimensional animation by Komatsu [44] and Magnenat-Thalmann et al. [51]. In both cases the control vertices of the surface are procedurally positioned based on the joint angles of an underlying skeleton and a few user-defined parameters. Due to their procedural nature, both techniques are limited in the degree to which the animator can influence the deformation.

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 [60]. 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, 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.

Extended versions of the methods described above form the basis for modern character animation software such as Maya [2]. A skeleton, which warps the space around it, is used to coarsely deform the object. Rather than deforming the surface geometry directly, the skeleton typically affects the vertices of a variety of deformers (such as FFDs or those of [60]), which in turn affect the surface geometry. These deformers have additional parameters that can be directly set by the animator. During the rigging process, the deformers and their relationship to the skeleton and surface geometry are established. During animation, the shape is partially determined by the configuration of the skeleton but can be augmented using the deformation parameters.

Many improvements have been made to the techniques discussed above. Lewis et al. [46] and

Sloan et al. [74] introduced methods to perform shape interpolation for skeleton-driven characters. By factoring out the nonlinear warping due to skeletal joints, they are able to linearly interpolate character shapes associated with particular skeletal configurations. The surface of the character can be hand-edited in any pose of the skeleton, and the given surfaces are then interpolated to provide a surface for any other pose. This technique was extended by Kry et al. to use commodity graphics hardware [45]. They decompose deformations into basis functions associated with each joint. Scanned data has been used by Allen et al. as the source of shape deformations for character animation [3, 4].

2.3 *Dynamic Deformation*

While widely used, static deformation techniques have an important shortcoming. In order to model realistic dynamic phenomena such as the vibration of fatty tissue, the animator must configure the deformation for each keyframe. An alternative set of techniques is based on simulating the laws of physical dynamics over the body.

The use of physically based dynamic deformations for computer animation was pioneered by Terzopoulos et al. [80]. They applied the Lagrangian equations of motion using a finite difference scheme to simulate elastic objects with regular parameterizations. Their framework was extended by Terzopoulos and Fleischer to include plastic deformation and fracture [79]. Terzopoulos and Witkin later showed that the equations of motion can be linearized about a moving frame of reference, as long as the deformations are modest [82]. The linearized equations are more stable and can be solved much more quickly.

Physically based deformations have since been extended in many ways. In order for dynamic deformable models to be more controllable for animation, Platt and Barr introduced better constraint handling using Lagrange multipliers [63]. Pentland and Williams obtained realtime simulations by using only a few vibration modes [61]. Witkin and Welch introduced the use of low-order polynomial deformations to achieve fast deformations, and employed constraints to create composite objects and to force deformable objects to follow prescribed paths [89]. Baraff and Witkin introduced non-penetration constraints to prevent objects from intersecting [10]. Metaxas and Terzopoulos combined global deformations with local finite element deformations [55]. O'Brien and Hodgins

computed realistic fractures of viscoelastic bodies using a finite element framework in which objects are approximated by a fine tessellation of tetrahedra [58]. A similar framework was used by Picinbono et al. to achieve interactive simulations of a virtual liver [62].

The techniques described above produce realistic flexible objects, but are not immediately applicable to character animation. The main sources of difficulty are that such simulations are computationally expensive and difficult to control. The computational expense comes from the inherent nonlinearity of the equations of elasticity. Using constraints based on Lagrange multipliers has not been shown to be effective in allowing the level of control needed for character animation.

2.4 Anatomical Modeling

Dynamic deformations have been applied to character animation primarily within the context of anatomical modeling. Anatomical modeling refers to the modeling of the internal anatomical structure of a character, such as bones, muscles, and fat. It is motivated by the assertion that by modeling the internal anatomical structures, increased realism of surface deformations can be achieved. While not all anatomical models are dynamic, it is natural to consider anatomy when applying dynamics to animated characters. The anatomy is the source of the dynamical properties of humans and animals. Thus the works discussed in this section also represent much of the progress in the application of dynamic deformations to character animation.

In one of the earliest uses of anatomical structure for animation, Waters animated faces using two muscle models: linear (*pull*) and sphincter (*squeeze*) [85]. Chadwick et al. first applied a layered anatomical approach to character animation [22]. Their models are composed of three layers: skeleton, fat, and skin. Much like some of the methods described earlier, their deformations are driven by an underlying skeleton. The skeleton is connected directly to some of the vertices of a dynamic deformation lattice (the fat layer). The surface of the character (the skin) is embedded in the deformation lattice and deforms accordingly.

At around the same time, Gourret et al. applied the finite element method to the modeling of the human hand [35]. Anatomically correct bones were used to constrain the deformation of the flesh of the hand. Their simulation was not dynamic (they solved for static equilibrium states), and muscles were not explicitly modeled.

The first explicit modeling of muscle for skeletal character animation was by Chen and Zeltzer [23]. They modeled not only the geometry of bones and muscles, but the muscle action forces. The muscle forces, based on biomechanical models, are fed into a dynamic finite element simulation. Their framework was demonstrated on an individual muscle, but was not applied to a complex character.

Scheepers et al. [70] and Wilhelms and Van Gelder [87] applied the explicit modeling of anatomical bones and muscles to full-body character animation. In both cases, the muscle shapes are a function of the configuration of the skeleton. In reality (as in [23]) it is the muscles that control the bones, but for the purpose of animation it is convenient for the animator to position the skeleton directly. An important drawback of both of these works is that the muscles do not deform dynamically. However, in [87] a dynamic fatty layer is modeled to connect the statically deforming muscles to dynamic skin, similar to [22].

Anatomical modeling has been incorporated into the animation pipeline for feature films such as Jurassic Park III [83], in which dinosaurs were modeled using techniques similar to [87]. Rhythm and Hues Studios has also used anatomical modeling to obtain realistic animal animation, but without a physically simulated fat layer [68]. In their system, muscle shapes are controlled by the configuration of the skeleton, and skin floats above the muscles.

Although anatomical modeling has been used effectively in production, it does have some shortcomings. The methods described in [70] and [68] do not model dynamics, so any desired dynamic effects must be created by the animator. The methods described in [87] and [83] contain a dynamic fatty layer between the muscles and the skin, but none of these systems support the dynamic simulation of muscles. Another drawback of anatomical modeling is the inherent complexity of the anatomy. The difficulty for the user in creating internal geometric structure and for the computer in updating it as it deforms is prohibitive for many applications.

2.5 Multiresolution and Subdivision-Based Simulation

Hierarchical bases have been widely studied in the field of numerical analysis (see, e.g., [9]). In computer graphics, hierarchical methods have been used to solve many problems including rendering [34], geometric modeling [33], and deformable model simulations. Terzopoulos et al. employed

a multigrid solver on a regular grid to solve for dynamic deformations [79]. Debunne et al. created interactive simulations using an octree representation, adaptive in both space and time [27]. To animate a surface, the surface points are linked to the grid by a weighting scheme. Later, Debunne et al. developed an adaptive framework for deformable models employing an unstructured hierarchy of tetrahedral meshes [28]. At each level of the hierarchy the object is approximated by a tetrahedral mesh. More recently, Grinspun et al. have developed a general method for organizing hierarchical simulations involving refinable function bases [37].

Our framework is similar to [27] in that we embed objects in domains that are easier to parameterize than the object itself. But since we do not require that the domain be a parallelepiped, we can fit it more closely to the underlying object. In our framework, coarse simulations do not require that the object be coarsely approximated as in [28]. Our coarse simulations factor in the detailed shape and material properties of the entire object, as approximated at a fine level of detail during the preprocessing stage.

Subdivision schemes are closely related to hierarchical methods and have also been used for simulation. Weimer and Warren employed 3D subdivision to solve partial differential equations associated with fluid flow [86]. Cirak et al. used subdivision surfaces to solve thin shell finite element problems, exploiting the smoothness of subdivision basis functions to satisfy the integrability requirements of thin shell elements [24, 25]. McDonnell et al. simulated volumetric subdivision objects using a mass-spring model [53] and then applied the finite-element methodology to the problem [54]. Subdivision surfaces have also been shown by Green et al. to be effective in solving thin shell equations hierarchically using the multigrid method [36].

Chapter 3

FINITE ELEMENT ELASTICITY

This chapter describes the mathematical model for computing the dynamics of an elastic body, and its finite element discretization. We model the dynamics of the deformable body as a system of second-order ordinary differential equations obtained by applying the finite element method (FEM) [59, 64, 67] to the Lagrangian formulation of the equations of elasticity [32, 49, 72].

3.1 Lagrangian Formulation

Consider a body whose rest state is defined by a domain $\Omega \subset \mathbb{R}^3$. The trajectory of the body over time is represented by a function

$$\mathbf{p} : \Omega \times \mathbb{R} \rightarrow \mathbb{R}^3 : (\mathbf{x}, t) \mapsto \mathbf{p}(\mathbf{x}, t). \quad (3.1)$$

It is convenient to decompose $\mathbf{p}(\mathbf{x}, t)$ as the sum of the rest state and a displacement

$$\mathbf{p}(\mathbf{x}, t) = \mathbf{x} + \mathbf{d}(\mathbf{x}, t), \quad (3.2)$$

as shown in Figure 3.1. The displacement function \mathbf{d} is the solution of a system of second-order partial differential equations, which we want to approximate by a finite system of second-order ordinary differential equations. This is done by introducing a finite *basis*, which is a collection of *basis functions*¹

$$\mathcal{B} = \{\phi_i(\mathbf{x}) : i = 1, 2, \dots, N\}. \quad (3.3)$$

¹We will discuss the choice of a particular basis in the next chapter.

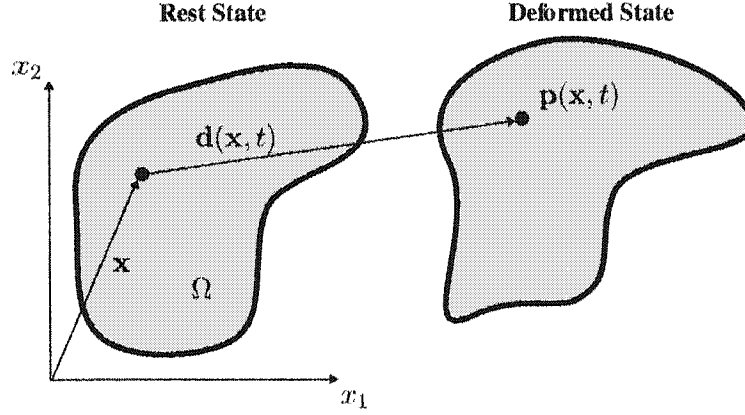


Figure 3.1 The position $\mathbf{p}(\mathbf{x}, t)$ of a body as it deforms over time can be decomposed into a rest state \mathbf{x} and displacement $\mathbf{d}(\mathbf{x}, t)$.

Expressing the displacement of the body in terms of \mathcal{B} yields the expansion

$$\mathbf{d}(\mathbf{x}, t) = \sum_{i=1}^N \mathbf{q}_i(t) \phi_i(\mathbf{x}), \quad (3.4)$$

where $\mathbf{q}_i(t) \in \mathbb{R}^3$.

We represent the state of the body at time t as a column vector of generalized coordinates $\mathbf{q} = \mathbf{q}(t)$ whose i -th component is $\mathbf{q}_i(t) \in \mathbb{R}^3$. Thus both the kinetic energy T and the potential energy U are functions of \mathbf{q} :

$$T = T(\dot{\mathbf{q}}) \text{ and } U = U(\mathbf{q}),$$

where $\dot{\mathbf{q}}$ denotes the time derivative of \mathbf{q} . The equations of motion are then

$$\frac{d}{dt} \left(\frac{\partial T}{\partial \dot{\mathbf{q}}} \right) + \frac{\partial U}{\partial \mathbf{q}} + \mu \dot{\mathbf{q}} = \mathbf{Q}, \quad (3.5)$$

where $\frac{\partial T}{\partial \dot{\mathbf{q}}}$ and $\frac{\partial U}{\partial \mathbf{q}}$ denote gradients with respect to $\dot{\mathbf{q}}$ and \mathbf{q} , respectively. The term $\mu \dot{\mathbf{q}}$ is a generalized dissipative force, added to simulate the effect of friction. A more realistic damping term could

easily be added (see, e.g., [58]). The term \mathbf{Q} is a *generalized force* corresponding to external forces such as gravity and those arising from constraints. Given an external *body force* $\mathbf{f}(\mathbf{x})$ representing force per unit volume, the generalized force is computed using the formula

$$\mathbf{Q}_i = \int_{\Omega} \mathbf{f}(\mathbf{x}) \phi_i(\mathbf{x}) dV. \quad (3.6)$$

Each component of \mathbf{Q} represents the force acting on one generalized coordinate.

The kinetic energy $T(\dot{\mathbf{q}})$ and potential energy $U(\mathbf{q})$ can be expressed as integrals over the domain Ω . To derive the equations of motion, we need to express T and U in terms of integrals involving the basis functions \mathcal{B} .

3.2 Kinetic Energy

The kinetic energy of a moving body is a generalization of the familiar $\frac{1}{2}mv^2$:

$$T = \frac{1}{2} \int_{\Omega} \rho(\mathbf{x}) \dot{\mathbf{p}} \cdot \dot{\mathbf{p}} dV.$$

Substituting Equations (3.2) and (3.4) into the above expression results in

$$T = \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N M_{ij} \dot{\mathbf{q}}_i \cdot \dot{\mathbf{q}}_j, \quad (3.7)$$

where ρ is the mass density of the body, and

$$M_{ij} = \int_{\Omega} \rho(\mathbf{x}) \phi_i(\mathbf{x}) \phi_j(\mathbf{x}) dV. \quad (3.8)$$

Equation (3.7) yields the formula

$$\frac{d}{dt} \left(\frac{\partial T}{\partial \dot{\mathbf{q}}} \right) = \mathbf{M} \ddot{\mathbf{q}}.$$

The matrix \mathbf{M} , composed of the 3×3 matrix elements $\mathbf{M}_{ij} = \mathbf{I} M_{ij}$, where \mathbf{I} is a 3×3 identity matrix, is commonly referred to as the *mass matrix*.

3.3 Elastic Potential Energy

The potential energy associated with deformation is the *elastic potential energy* U . It is based on measuring the *strain* or distortion present in the body. Green's strain tensor is a common measure of strain (see, e.g., [32]):

$$e_{ij} = \frac{\partial d_i}{\partial x_j} + \frac{\partial d_j}{\partial x_i} + \sum_{k=1}^3 \frac{\partial d_k}{\partial x_i} \frac{\partial d_k}{\partial x_j}, \quad i, j = 1, 2, 3. \quad (3.9)$$

A related concept is that of *stress* (also a tensor), which measures the forces present in a continuous body. For *linear elastic* (stress is proportional to strain) and isotropic bodies, stress has the following relation to strain:

$$\tau_{ij} = 2G \left(\frac{\nu}{1-2\nu} \text{tr}(e) \delta_{ij} + e_{ij} \right), \quad i, j = 1, 2, 3,$$

where $\text{tr}(e) = \sum_{i=1}^3 e_{ii}$, and

$$\delta_{ij} = \begin{cases} 0 & \text{for } i \neq j \\ 1 & \text{for } i = j \end{cases}$$

is the Kronecker delta function. The scalar G , called the *shear modulus*, determines how much the body resists deformation, and the scalar ν , called *Poisson's ratio*, determines the extent to which strains in perpendicular directions are related to each other.

The elastic potential energy U , analogous to computing *work* as force times distance, is computed by taking the component-wise product of the stress and strain tensors:

$$U = \int_{\Omega} G \left(\frac{\nu}{1-2\nu} \text{tr}^2(e) + \sum_{i=1}^3 \sum_{j=1}^3 e_{ij} e_{ij} \right) dV. \quad (3.10)$$

Note that the quantities G , ν and e are functions of \mathbf{x} . By combining Equations (3.4), (3.9), and (3.10) we can express the elastic potential U and its derivatives (with respect to \mathbf{q}) as polynomial functions of \mathbf{q} . The coefficients of these polynomials are integrals that can be precomputed. Their exact form can be found in Appendix A. The matrix $\mathbf{S} = \frac{\partial^2 U}{\partial \mathbf{q} \partial \mathbf{q}}$, which is needed during the simulation (Section 4.5), is commonly referred to as the *stiffness matrix*.

3.4 Gravity

Gravity is an example of a *body force* that affects all points inside the body. We treat gravity as a constant acceleration field specified by the vector \mathbf{g} . The gravitational potential energy is then the integral

$$U^g = \int_{\Omega} \rho(\mathbf{x}) \mathbf{g} \cdot \mathbf{p}(\mathbf{x}) dV = \int_{\Omega} \rho(\mathbf{x}) \mathbf{g} \cdot \left(\mathbf{x} + \sum_{i=1}^N \mathbf{q}_i \phi_i(\mathbf{x}) \right) dV. \quad (3.11)$$

Although gravity is conservative, as is evident from the existence of the above potential, for simplicity of exposition we treat it as an external force that is added into \mathbf{Q} in Equation (3.5). The *generalized gravitational force* is the gradient

$$\mathbf{Q}_i^g = -\frac{\partial U^g}{\partial \mathbf{q}_i} = -\left(\int_{\Omega} \rho(\mathbf{x}) \phi_i(\mathbf{x}) dV \right) \mathbf{g}. \quad (3.12)$$

The above force can be interpreted as the familiar $m\mathbf{g}$ except that the mass term represents all of the mass associated with a particular basis function.

3.5 Constraints

Using Lagrange multipliers, we support standard constraints that can be described by equations of the form $\mathbf{C} = \mathbf{0}$. For example, we can constrain a body point $\mathbf{p}(\mathbf{x}_0)$ to coincide with the arbitrary point \mathbf{p}_0 as follows:

$$\mathbf{0} = \mathbf{C}(\mathbf{q}) = \mathbf{p}(\mathbf{x}_0) - \mathbf{p}_0 = \mathbf{x} + \left(\sum_{i=1}^N \mathbf{q}_i \phi_i(\mathbf{x}_0) \right) - \mathbf{p}_0. \quad (3.13)$$

The force required to maintain the constraint is of the form $-\frac{\partial \mathbf{C}}{\partial \mathbf{q}}^T \lambda$, where the Lagrange multiplier λ is the magnitude of the force. Our total external force is then $\mathbf{Q} = \mathbf{Q}^g - \frac{\partial \mathbf{C}}{\partial \mathbf{q}}^T \lambda$. To maintain the constraint an additional condition must hold: $\ddot{\mathbf{C}} = \frac{\partial \mathbf{C}}{\partial \mathbf{q}} \ddot{\mathbf{q}} = \mathbf{0}$.

3.6 System of Equations

Collecting together the terms described above, substituting them into Equation (3.5), and applying Baumgarte stabilization (see [55]) to our constraints yields the system of ordinary differential

equations

$$\begin{bmatrix} \mathbf{M} & \frac{\partial \mathbf{C}^T}{\partial \mathbf{q}} \\ \frac{\partial \mathbf{C}}{\partial \mathbf{q}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{Q}^g - \mu \dot{\mathbf{q}} - \frac{\partial U}{\partial \mathbf{q}} \\ -\alpha \dot{\mathbf{C}} - \beta \ddot{\mathbf{C}} \end{bmatrix}. \quad (3.14)$$

The Baumgarte stabilization parameters α and β control a generalized, damped spring that acts to restore the constraints when they are not being met.

Summary

We have laid the groundwork for simulating the dynamic motion of elastic bodies. Computing the motion of an object corresponds to solving Equation (3.14) given initial values for \mathbf{q} and $\dot{\mathbf{q}}$, for which standard techniques are available. However, such solutions are computationally expensive (primarily due to the nonlinearity of $\frac{\partial U}{\partial \mathbf{q}}$) and thus impractical for interactive systems. In the next chapter we will present a practical system for solving an approximation of Equation (3.14) for interactive animation.

Chapter 4

INTERACTIVE SIMULATION FRAMEWORK

The formulation described in the previous chapter can be used to simulate elastic bodies, but is not practical for interactive systems on current hardware. There are essentially two sources of computational expense. One is the inherent nonlinearity of the system in Equation (3.14); the function $U(\mathbf{q})$ is a quartic polynomial. The other is that a large number of degrees of freedom may be required to model a complex shape. In this chapter we address these problems and present an interactive system for simulating elastic bodies. Our approach has three key components:

- *Embedding* - To reduce the number of variables being computed, we embed complex objects in simple domains (Section 4.1).
- *Adaptive simulation* - Rather than compute at a fixed resolution, we allow the simulation to adaptively add detail where it is needed (Sections 4.2 and 4.7).
- *Quasi-linearization* - We approximate $\frac{\partial U}{\partial \mathbf{q}}$ by linearizing it about a rotated rest state (Section 4.3).

This chapter addresses the above topics along with various technical details of our simulation system.

4.1 *Embedding*

The framework presented in the previous chapter left open the choice of a particular set of basis functions (Equation (3.3)). In the FEM setting, the basis is associated with a grid over the body. The grid provides the body with a regular parametric structure from which the basis functions can be automatically generated. Four common approaches to grid construction are shown in Figure 4.1. The advantages and disadvantages of each are outlined below.

- *Parameterization* – The interior of the object is exactly parameterized by a function on a

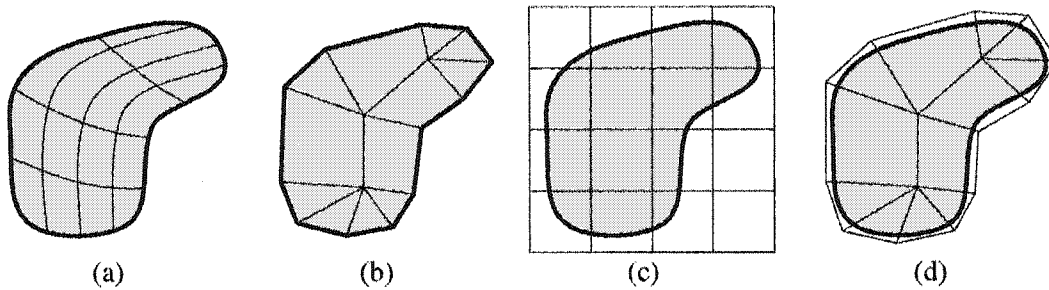


Figure 4.1 Comparison among grid types. (a) *Parameterization* – The body is shown in gray with a thick border. The lines on the body represent isoparametric curves of the parameterization. (b) *Approximation* – The body is approximated by a set of regions, each of which is parameterized. (c) *Regular embedding* – A regular grid, with isoparametric curves shown, embeds the body. (d) *Semi-regular embedding* – A set of regions, each parameterized, embeds the body.

regular domain. This approach is ideal from the point of view of representing the object and its physical properties exactly. But creating a parameterization is difficult for complex objects.

- *Approximation* – The object is approximated by a semi-regular domain such as a tetrahedral mesh. This approach is common for engineering applications but is deficient for computer graphics because the appearance of the object is crucial.
- *Regular embedding* – The object is embedded in a regular region such as a parallelepiped. Unlike approximation, the parametric domain is a superset of the domain of the body, so every point on the original surface is parameterized by the grid. This allows the original surface to be factored into the simulation and displayed. The disadvantage is that a fine grid is necessary for a good fit to the object, without which significant fidelity may be lost.
- *Semi-regular embedding* – The object is embedded in a semi-regular region such as a tetrahedral mesh. Although more difficult to construct than a regular embedding, a semi-regular embedding allows the grid to more closely conform to the object, improving the fidelity of the results.

Semi-regular embedding was selected as the best gridding scheme for the current application. It is much easier to construct a semi-regular embedding than to exactly parameterize the body. Embedding allows a coarse grid to be used in conjunction with the original surface. Embedding is also

agnostic to the underlying surface representation, decoupling the simulation from the modeling of the surface. And a semi-regular grid produces much better results than a regular grid of comparable resolution.

Our approach is a generalization to subdivision functions of the embedding methods used in the finite element community under the headings *domain imbedding* or *fictitious domain* methods [13, 52]. Rather than attempting to construct a basis of functions on the region Ω , one instead views Ω as embedded in a semi-regular region $\tilde{\Omega}$ (for instance, a collection of cubes in \mathbb{R}^3) for which a basis is known. Restriction to Ω then induces a collection of functions on the original domain. Figures 4.7(a) and 4.7(d) illustrate the embedding approach applied to a dragon-shaped region in three dimensions.

More generally, consider a piecewise smooth homeomorphism

$$\mathbf{r} : K \rightarrow \tilde{\Omega} \subset \mathbb{R}^3 : u \mapsto \mathbf{r}(u) \quad (4.1)$$

from a three-dimensional hexahedral complex K with canonical coordinates u onto a superset $\tilde{\Omega} \supset \Omega$. Roughly speaking, a hexahedral complex is a collection of abstract cubes which may share faces, edges, and vertices. The homeomorphism \mathbf{r} , which parameterizes $\tilde{\Omega}$, can then be used to transfer a basis defined on K to one on $\tilde{\Omega}$.

We use subdivision rules on K to define both the parameterization \mathbf{r} and a hierarchical basis \mathcal{B} of functions on Ω that will be described in the next section. A subdivision framework was chosen for the following reasons:

1. They allow a large class of semi-regular grids so the object can be embedded snugly with fewer elements.
2. They are intrinsically hierarchical, so a hierarchical basis is easily constructed (Section 4.2).
3. They provide a choice between basis functions that are smooth almost everywhere, and trilinear basis functions with smaller support for faster computation.
4. They provide a smoothing filter.

Our implementation is based on hexahedral subdivision and supports trilinear subdivision (see Appendix B) and the subdivision schemes of MacCracken and Joy [50] and Bajaj et al. [8] (multi-linear

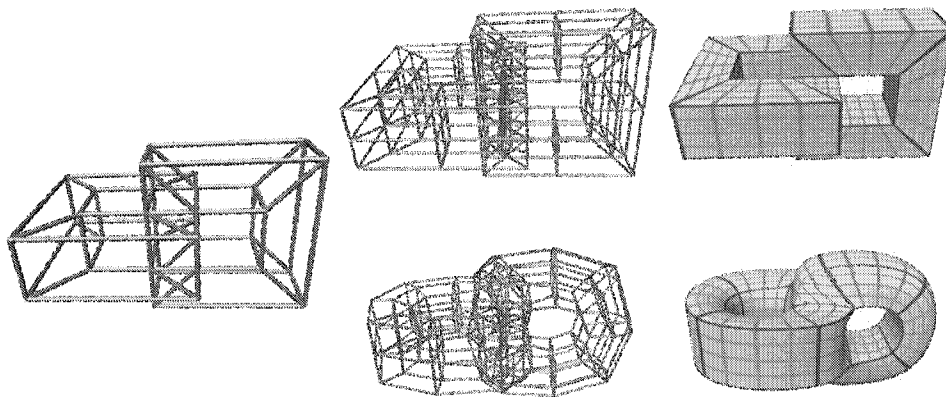


Figure 4.2 On the left is a cell complex whose vertices have been positioned in \mathbb{R}^3 . We refer to a cell complex together with a mapping of its vertices into \mathbb{R}^3 as a *control lattice*. We visualize a control lattice using linear interpolation to define edges and faces in \mathbb{R}^3 . The center column of images show the result of subdividing the control lattice on the left once using trilinear subdivision (top) and the MacCracken-Joy scheme augmented with the sharp surface rules of [29] (bottom). The rightmost images represent the limit volumes produced by infinite subdivision. They are computed by subdividing a few times and then applying limit masks to the vertices. In the rightmost images the object has been colored to indicate the surface of the volumetric parameterization induced by subdivision. The dark green lines correspond to boundaries between the cells of K . The light green lines represent isoparametric curves within a cell.

cell averaging, or MLCA), which are generalizations of Catmull-Clark subdivision surfaces [20]. In all three cases, we support only those complexes that result in hexahedral complexes after one subdivision step. This restriction is necessary in order to ensure that all objects are parameterized by K in a straightforward manner. It still allows a variety of polyhedral cells, including hexahedra, tetrahedra and triangular prisms. While less has been proven about the smoothness of the MacCracken-Joy scheme than MLCA, we have found that the former produces surfaces that are more visually pleasing. Therefore we used the trilinear and MacCracken-Joy schemes for the examples shown in this chapter. In Chapter 5 we use the MLCA scheme because it supports the internal sharp edges required to meet skeletal constraints.

When one of the subdivision schemes mentioned above is applied to a cell complex K it gives rise to a function space spanned by a basis $\mathcal{B}_0 = \{\phi_i^0(u) : i = 1, 2, \dots, N_0\}$, where N_0 is the number of vertices of K . The basis function $\phi_i^0(u)$ is computed by assigning a 1 to the i -th vertex of K

and a 0 to all other vertices and then applying subdivision. Thus, there is a correspondence between the vertices of K and the basis functions. For more details on the relationship between subdivision schemes and basis functions see [48]. The parameterization \mathbf{r} is formed by assigning positions to the vertices of K , which corresponds to taking a linear combination of the basis functions:

$$\mathbf{r}(u) = \sum_{i=1}^{N_0} \mathbf{r}_i \phi_i^0(u), \quad (4.2)$$

where $\mathbf{r}_i \in \mathbb{R}^3$. We refer to a cell complex K together with a mapping of its vertices into \mathbb{R}^3 as a *control lattice*.

Not every set of coefficients $\{\mathbf{r}_i\}$ defines a homeomorphism when assigned to the vertices of K . Unfortunately, we do not know of an easy way to detect overlap in \mathbf{r} , so we resort to numerical approximation to verify that \mathbf{r} is indeed a homeomorphism. This is done by approximating K as a collection of tetrahedra and \mathbf{r} as a piecewise linear function, and then checking that no two tetrahedra overlap when mapped by \mathbf{r} to $\tilde{\Omega}$. If tetrahedra do overlap, the vertices must be repositioned.

Figure 4.2 shows examples of a trilinear and a MacCracken-Joy subdivision solid that are parameterized by a cell complex. Figure 4.8(b) shows another example of a region parameterized using trilinear subdivision.

4.2 The Hierarchical Basis

By composing the basis functions in \mathcal{B}_0 with \mathbf{r} and then restricting them to Ω they can be treated as basis functions on Ω . We augment this basis to form the *hierarchical basis* [9] that we use for adaptive simulation. Roughly speaking, a collection $\mathcal{B} = \{\phi_i(u) : i = 1, 2, \dots\}$ is called a hierarchical basis if the diameter of the support of ϕ_i decreases as i increases.¹

The aforementioned subdivision schemes give rise to infinite sequences of nested function spaces [48] spanned by the elements of a hierarchical basis \mathcal{B} . Analogous to the the basis \mathcal{B}_0 described above on K , each finite-dimensional space is spanned by a set of functions, one corresponding to each vertex of the *subdivided* cell complexes. Using a standard construction (see, e.g., [76]), we form a hierarchical basis by selecting a linearly independent subset of these basis functions.

¹It is also desirable that \mathcal{B} be well-behaved with respect to the L^2 -inner product on $L^2(\Omega)$.

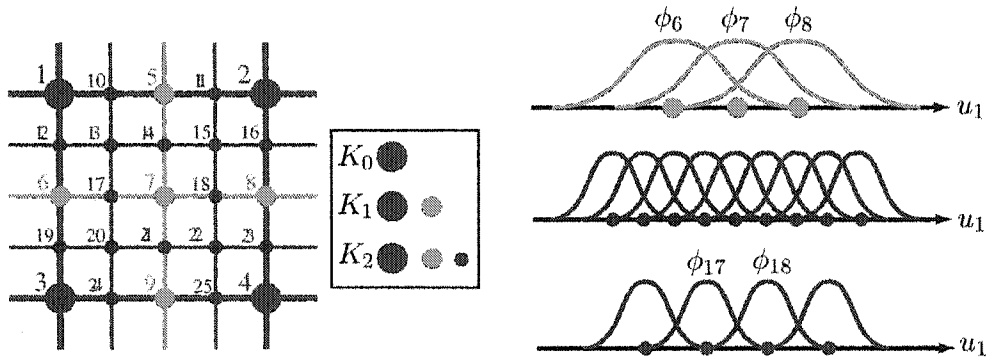


Figure 4.3 On the left is a visualization of one cell in a complex $K = K_0$. The original cell is shown in red; green and blue edges indicate two finer subdivisions. In our framework, the complex is actually three dimensional, but we show the two dimensional case for convenience. Corresponding to each vertex of K_0 is a basis function. After subdividing K_0 once, we introduce new vertices that belong to K_1 (in addition to the vertices of K_0 , as shown in the key next to the grid). We could place basis functions (half as big in each dimension with respect to the coarser level) at all vertices of K_1 . Instead, we trim this overrepresentation by only placing the new functions at the newly created vertices (shown in green). The process continues recursively (e.g., new basis functions at the blue vertices). Note that, since only one basis function is centered at each vertex, the index of that vertex uniquely identifies the basis function. On the right we see a one dimensional visualization of a slice through the basis functions along the line from vertex 6 to vertex 8 as the parameter u_1 varies. On the top are the subdivision basis functions at level 0. In the middle are the level 1 subdivision basis functions, which form an overcomplete basis when combined with the level 0 basis functions. On the bottom are the level 1 basis functions that remain after discarding every other basis function; they are included in the hierarchical basis.

Figure 4.3 demonstrates the hierarchical basis construction. At the coarsest level are the basis functions $\mathcal{B}_0 = \{\phi_i^0 : i = 1, 2, \dots, N_0\}$ that correspond to the original vertices of K . Repeated subdivision of K introduces an increasingly fine sequence of complexes K_k (see Figure 4.2). Applying subdivision to the complex K_k gives rise to basis functions at level k . The basis \mathcal{B}_k is inductively formed from \mathcal{B}_{k-1} by appending the basis functions ϕ_i^k on K_k where a ranges over the set of *odd vertices* of K_k (vertices introduced when we subdivide K_{k-1}). The hierarchical basis \mathcal{B} is the union of the nested sequence

$$\mathcal{B}_0 \subset \mathcal{B}_1 \subset \mathcal{B}_2 \subset \dots \quad (4.3)$$

For practical computation, we truncate the basis after N basis functions by including only the first N_{levels} levels of the hierarchy. Typically, we let $N_{levels} = 3$. Notice that the index k is redundant because each vertex i appears at a unique subdivision level k . We, therefore, drop the index k , writing ϕ_i instead of ϕ_i^k .

As mentioned previously, the basis functions can be treated as functions on Ω . This is convenient because the equations of elasticity are easiest to describe in Euclidean coordinates, as in Chapter 3. Considering the basis functions as functions on Ω , the parameterization \mathbf{r} provides an expansion of the identity function:

$$\mathbf{x} = \mathbf{r}(\mathbf{x}) = \sum_{i=1}^{N_0} \mathbf{r}_i \phi_i^0(\mathbf{x}) = \sum_{i=1}^N \mathbf{r}_i \phi_i(\mathbf{x}).$$

Note that for convenience we represent \mathbf{r} as a sum over the entire hierarchical basis although only the coarse basis functions are used (i.e., $\mathbf{r}_i = \mathbf{0}$ for $i > N_0$).

In Figure 4.8(c) the vertices of a subdivided control lattice have been colored to indicate the structure of the hierarchical basis.

4.3 Quasi-linearization

For complex models in which there are many basis functions, the full nonlinear equations of elasticity are too expensive to solve interactively because even evaluating the stiffness matrix once per simulation step is costly, and the equations of motion must be linearized. We support two methods of linearization, in addition to the nonlinear solver.

If the deformations are small, the nonlinear terms of strain (Equation (3.9)) can be dropped,

resulting in the traditional quadratic (instead of quartic) elastic potential, and thus a constant stiffness matrix. If the deformations are large but differ only slightly from a rigid motion then the strain can be linearized about a floating frame of reference that roughly tracks the orientation of the object (see [82]). If large deformations are required and significant error is unacceptable, then the full non-linear formulation is necessary.

We provide a novel approach to the large-rotation small-deformation scenario. Terzopoulos et al. [82] (and similar formulations in the engineering literature, e.g., [72]) integrate a moving frame of reference into the dynamic equations, adding greatly to the complexity of the exposition and implementation. The frame of reference attempts to track the configuration of the object as if it were a rigid body. Besides the added complexity, another problem is that over time, due to numerical error, the frame of reference will drift out of alignment with the deforming body.

Our approach is to choose a frame of reference for each timestep to reduce the amount of rotation present in the displacement function. We rewrite the state during timestep k as

$$\mathbf{p}(\mathbf{x}, t) = \mathbf{R}^k(\mathbf{x} + \mathbf{d}^k(\mathbf{x}, t)), \quad (4.4)$$

where \mathbf{R}^k is a rotation matrix that transforms points from the local reference frame into the world coordinate system, and \mathbf{d}^k represents the displacement of the body in the local reference frame. We proceed to solve for \mathbf{d}^k in the local coordinate frame. Using nonlinear elasticity, due to rotational invariance, this reparameterization does not effect the solution. In the linear case, choosing \mathbf{R}^k so that \mathbf{d}^k is small and contains little rotation greatly reduces the error caused by linearization.

After timestep k , the displacement \mathbf{d}^k may contain a rotational component. We extract this component in order to compute \mathbf{R}^{k+1} , which determines the local reference frame that will be used during timestep $k + 1$. As shown in Appendix C, the rotation present in \mathbf{d}^k can be estimated as $\frac{1}{2}\|\nabla \times \mathbf{d}^k\|$ radians about the $\nabla \times \mathbf{d}^k$ axis. Under the assumption that the body is mildly deformed and thus the rotation is nearly uniform throughout the body, we evaluate the rotation at a single point \mathbf{x}_0 . Letting $\Delta\mathbf{R}^k$ be the corresponding rotation matrix, we have $\mathbf{R}^{k+1} = \mathbf{R}^k\Delta\mathbf{R}^k$. Applying Equation (4.4) for timesteps k and $k + 1$, the displacement in the new coordinate frame is

$$\mathbf{d}^{k+1} = (\Delta\mathbf{R}^k)^T(\mathbf{x} + \mathbf{d}^k) - \mathbf{x},$$

and its velocity is

$$\dot{\mathbf{d}}^{k+1} = (\Delta \mathbf{R}^k)^T (\dot{\mathbf{d}}^k).$$

Applying finite element expansion, the update at the end of timestep k becomes

$$\begin{aligned} \mathbf{q}_i^{k+1} &= (\Delta \mathbf{R}^k)^T (\mathbf{r}_i + \mathbf{q}_i^k) - \mathbf{r}_i, \text{ and} \\ \dot{\mathbf{q}}_i^{k+1} &= (\Delta \mathbf{R}^k)^T (\dot{\mathbf{q}}_i^k). \end{aligned}$$

4.4 Numerical Integration

To speed up the computation, we precompute the mass, gravity, and elasticity integrals (Equations (3.8), (3.12), and (A.2)) using numerical quadrature. We first subdivide to the level desired for quadrature (at least once) and compute the values of all of the basis functions at each of the vertices. After subdividing, the domain is composed of hexahedral cells, which we proceed to split into tetrahedra. We then compute the integrals over each domain tetrahedron using piecewise linear approximations to the basis functions. Since at every vertex the Euclidean coordinates are known, we can compute the spatial derivatives of the basis functions directly without using knowledge about the parameterization of the object by the complex K .

As described in Section 4.2, we represent functions on Ω by restricting functions on K to Ω . The restriction is approximated during numerical quadrature at the level of the tetrahedra mentioned in the previous paragraph. When computing integrals over Ω , we do not integrate over tetrahedra that are deemed to be outside the object. We accomplish this by regularly sampling each tetrahedron and testing whether the sampled point is outside the surface. If all of the sampled points are outside the surface then the tetrahedron is discarded. In our current implementation, tetrahedra that straddle the surface contribute to the computed integrals (an improvement would be to weight the integral over a tetrahedron according to the fraction of sample points inside the object). Figure 4.8(d) shows the tetrahedra that were used to approximate the interior of the dragon model.

The computed integrals involve products of as many as four basis functions (see Appendix A), so it is important to know which basis functions are nonzero over a given tetrahedron; otherwise, all 4-tuples would be integrated. We accomplish this by storing the basis heights as sparse vectors at each vertex. Our subdivision scheme operates directly on the sparse vectors to compute the

basis heights. By examining the sparse vectors of basis heights at each vertex of a tetrahedron, the integrator knows which basis functions are nonzero.

4.5 Solving the ODEs

Once we have precomputed the mass and stiffness integrals, we are prepared to solve the system in Equation (3.14) together with initial values for \mathbf{p} and $\dot{\mathbf{p}}$ (and thus \mathbf{q} and $\dot{\mathbf{q}}$). Solution techniques typically start with known values for \mathbf{q} and $\dot{\mathbf{q}}$ and proceed to compute the values of these variables at a sequence of subsequent points in time.

There are two common classes for solving such systems of differential equations. Explicit techniques compute the future state of the system using information about the state of the system at the current and previous timesteps. Forward Euler and Runge-Kutta are examples of such explicit methods. Implicit techniques express the future state in terms of quantities evaluated at the end of the timestep, in addition to previously known quantities. Implicit methods are much more stable for large timesteps than explicit methods because rather than jumping blindly forward, the conditions at the future state are taken into consideration. For discussions of implicit methods see [11, 38].

We desire a fast, stable solution, so we use an implicit method to solve our system of equations. Applying the method of [11], adapted to our constrained system, results in the following nonlinear system of equations:

$$\begin{bmatrix} \mathbf{M} & \frac{\partial \mathbf{C}^T}{\partial \mathbf{q}} \\ \frac{\partial \mathbf{C}}{\partial \mathbf{q}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \Delta \mathbf{v} \\ \lambda \end{bmatrix} = \begin{bmatrix} h\mathbf{F}(\mathbf{q}_0 + h(\mathbf{v}_0 + \Delta \mathbf{v}), \mathbf{v}_0 + \Delta \mathbf{v}) \\ -\alpha \dot{\mathbf{C}} - \beta \ddot{\mathbf{C}} \end{bmatrix}, \quad (4.5)$$

where h is the timestep length, $\Delta \mathbf{v}$ is the change in the velocity $\dot{\mathbf{q}}$ over the timestep, \mathbf{q}_0 is the value of \mathbf{q} at the beginning of the timestep, \mathbf{v}_0 is the value of $\dot{\mathbf{q}}$ at the beginning of the timestep, and $\mathbf{F} = \mu \dot{\mathbf{q}} - \mathbf{Q}^g - \frac{\partial U}{\partial \mathbf{q}}$ is the sum of all forces acting on the system (excluding the constraint force). We solve the above system of equations for $\Delta \mathbf{v}$ and λ using the Newton-Raphson root-finding method. The cost of solving the equations comes primarily from computing the stiffness matrix, which is required to compute the gradient of Equation (4.5). For our simulations we have found it acceptable to perform only one iteration of Newton-Raphson. This corresponds to linearization of Equation (4.5) at each timestep (as in [11]), but should not be confused with the commonly used

linearization of strain, which is only accurate for small deformations (we discuss such linearization in Section 4.3). The linear systems that need to be solved when performing Newton-Raphson on Equation (4.5) are symmetric but indefinite, so we solve them using the iterative *minres* algorithm using sparse matrices [65]. After computing $\Delta \mathbf{v}$, the state of the system is updated as follows:

$$\begin{aligned}\dot{\mathbf{q}} &\leftarrow \dot{\mathbf{q}} + \Delta \mathbf{v} \\ \mathbf{q} &\leftarrow \mathbf{q} + h\dot{\mathbf{q}}.\end{aligned}$$

4.6 Runtime Details

The surface of the object is described as a triangle mesh. At each vertex of the surface mesh, we store a sparse vector of basis function values. The value of a function (such as \mathbf{d}) at a vertex of the surface mesh can be reconstructed from the basis values stored at the vertex. To evaluate a function within a triangle of the surface mesh, we interpolate between the values at the three vertices. The basis function information at the vertices facilitates the setting up of a constraint when the user clicks and drags the surface. Using the basis information at the vertices we can quickly select the relevant subset of basis functions, and set up a constraint at that point as in Section 3.5.

4.7 Adapting the Basis

Because the basis \mathcal{B} is hierarchical, it is possible to adaptively choose an *active* basis $\mathcal{B}_A \subset \mathcal{B}$ to be used in the place of \mathcal{B} so that detail is added where needed. There are two pertinent questions regarding adaptation: “*how to adapt?*” and “*when and where to adapt?*”

The first question is easily answered in our framework. We choose *a priori* a maximum allowable subdivision level k and precompute the mass and stiffness terms needed to form the system in Equation (4.5) when $\mathcal{B}_A = \mathcal{B}_k$. These terms are stored in sparse data structures². When a decision is made to add or remove a basis function from the active basis \mathcal{B}_A , it is only necessary to add or remove precomputed terms from the current mass and stiffness matrices. We note that the idea of adapting the basis is not new (see, e.g., [33]), and has recently been generalized by Grinspun et

²In the linearized case, the terms of the mass and stiffness matrices are stored in sparse matrices. In the nonlinear case, we store the integrals from Equation (A.2) in hash tables.

al. [37].

We address the second question by a heuristic similar to the one used in [28]: areas of higher deformation require more detail. The elements of \mathcal{B}_h are organized into a tree, with a parent-child relationship between basis functions at adjacent subdivision levels and having intersecting support. Each level of the hierarchy has two separate thresholds for determining when to refine or coarsen. If the deformation is above the *activation threshold* in the region of support of a basis function ϕ_i then the children of ϕ_i are activated. A basis function is deactivated if the deformation is below the *deactivation threshold* in its region of support. As noted in [28], a lower deactivation threshold discourages the system from immediately removing newly introduced basis functions.

4.8 Linear Subspace Constraints

Because we would like our objects to interact with other objects, position constraints are important, but the use of Lagrange multipliers as in Equation (3.14) slows down the simulation by increasing its complexity. The framework of Baraff and Witkin [11] provides an elegant alternative for particle systems. During each internal step of a conjugate gradients (CG) solver they project out certain components of $\Delta \mathbf{v}$ corresponding to constrained particles. Here we show that this technique can be extended to include position constraints at any point in a continuous body.

If we remove the constraints from Equation (4.5), and build into the formulation a single iteration of Newton-Raphson, we arrive at the system presented in [11]:

$$\Delta \mathbf{q} = h(\mathbf{v}_0 + \Delta \mathbf{v}) \quad (4.6)$$

$$\left(\mathbf{M} - h \frac{\partial \mathbf{F}}{\partial \dot{\mathbf{q}}} - h^2 \frac{\partial \mathbf{F}}{\partial \mathbf{q}} \right) \Delta \mathbf{v} = h \left(\mathbf{F}_0 + h \frac{\partial \mathbf{F}}{\partial \mathbf{q}} \mathbf{v}_0 \right), \quad (4.7)$$

where \mathbf{F}_0 is the value of \mathbf{F} at the beginning of the timestep, and all derivatives are evaluated at the beginning of the timestep. Moving position constraints in our framework are of the form:

$$\mathbf{d}_c(t) = \sum_{i=1}^N \mathbf{q}_i \phi_i(\mathbf{x}_c), \quad (4.8)$$

which simply says that the displacement at \mathbf{x}_c conforms to some known function \mathbf{d}_c . Evaluating

Equation (4.6) at \mathbf{x}_c results in:

$$\sum_{i=1}^N \Delta \mathbf{v}_i \phi_i(\mathbf{x}_c) = \frac{\mathbf{d}_c(t+h) - \mathbf{d}_c(t)}{h} - \sum_{i=1}^N \mathbf{v}_{0,i} \phi_i(\mathbf{x}_c). \quad (4.9)$$

The right hand side of the above equation is simply a constant \mathbf{a}_c that can be computed at the beginning of each timestep. If we accumulate the x , y , and z components of the 3-vectors \mathbf{a}_c into the N -vectors \mathbf{a}^α , where $\alpha \in \{x, y, z\}$, define the matrix \mathbf{C} with elements $C_{ic} = \phi_i(\mathbf{x}_c)$, and separate $\Delta \mathbf{v}$ into its x , y , and z components $\Delta \mathbf{v}^\alpha$, Equation (4.9) becomes:

$$\mathbf{C}^T \Delta \mathbf{v}^\alpha = \mathbf{a}^\alpha, \alpha \in \{x, y, z\}. \quad (4.10)$$

So each constraint requires that $\Delta \mathbf{v}$ be constant along three particular directions. Maintaining the constraints involves the following steps:

1. At the beginning of each timestep, $\Delta \mathbf{v}$ is initialized so that Equation (4.10) holds. This is accomplished by computing the QR-decomposition $\mathbf{C} = \hat{\mathbf{Q}}\hat{\mathbf{R}}$ and transforming Equation (4.10) into $\hat{\mathbf{R}}^T \mathbf{b}^\alpha = \mathbf{a}^\alpha$, $\Delta \mathbf{v}^\alpha = \hat{\mathbf{Q}} \mathbf{b}^\alpha$, from which $\Delta \mathbf{v}$ can be easily computed. Although QR-decomposition of an $n \times m$ matrix requires $O(nm^2)$ time, in our case the number of constraints m is typically small, so the computational cost is low.
2. Each column \mathbf{c} of \mathbf{C} has an associated *projection matrix* $\mathbf{P} = \mathbf{I} - \mathbf{c}\mathbf{c}^T / \mathbf{c}^T \mathbf{c}$, which, when applied to a vector, eliminates the component in the direction of \mathbf{c} . These projectors are applied during CG such that incremental updates to $\Delta \mathbf{v}$ are orthogonal to the vectors \mathbf{c} , ensuring that Equation (4.10) remains true (for details see [11]).

In our current framework, conflicting constraints can be detected during QR-decomposition and removed. In the future we hope to augment this method to solve over-constrained systems more elegantly, as was done for FFD by Hsu et al. [41].

4.9 Realtime Simulation

In order for an interactive simulation to have the appearance of realism, it is important for the simulation proceed at a consistent pace. Adaptively changing the basis introduces variation in the

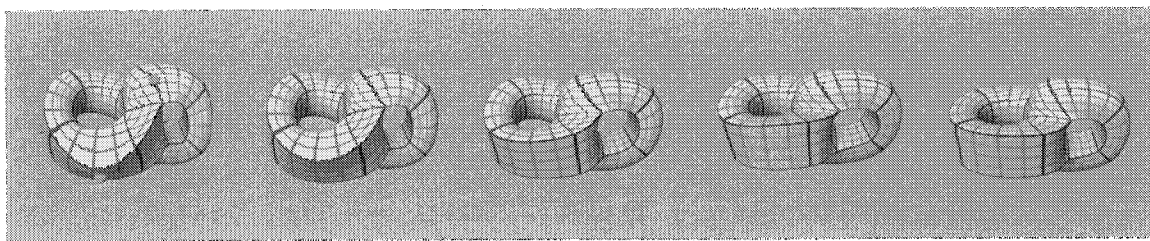


Figure 4.4 The deformation of an object modeled using MacCracken-Joy subdivision with sharp surface rules. Upon releasing the constraints (represented as cyan spheres in the first frame), the object dynamically vibrates and eventually returns to its rest shape.

amount of time required to compute a single timestep. Since our simulator can take large timesteps we can remedy this problem by adjusting the timestep to stay in sync with actual time. For example, when basis functions are added each step of the simulation will take longer due to the increased number of degrees of freedom in the system. We compensate by integrating over a longer period of virtual time during each timestep.

So why not take arbitrarily large timesteps? First, interactive applications demand high frame rates. It is best to display a new state of the system at each video refresh cycle. Second, implicit integration exacts payment for its improved stability. Large timesteps result in unrealistic damping (see, e.g., [30]). For these reasons we typically set the timestep to the amount of physical time that lapsed during the previous iteration of simulation and display.

4.10 Results

We now describe the results of implementing our framework and running a variety of simulations. All of the simulations were performed interactively on a standard desktop PC (Athlon 1.4 GHz, 256 MB ram).

Sharp Features. DeRose and Kass [29] added rules for sharp features to the Catmull-Clark subdivision framework. Since the boundaries of MacCracken-Joy solids are Catmull-Clark surfaces, we can easily include sharp features in our framework.³ Figure 4.4 shows a simulation involving an

³We have not studied the limit behavior of the subdivision scheme that results from adding sharp features to the surfaces of MacCracken-Joy volumes, but it seems to work well in practice.

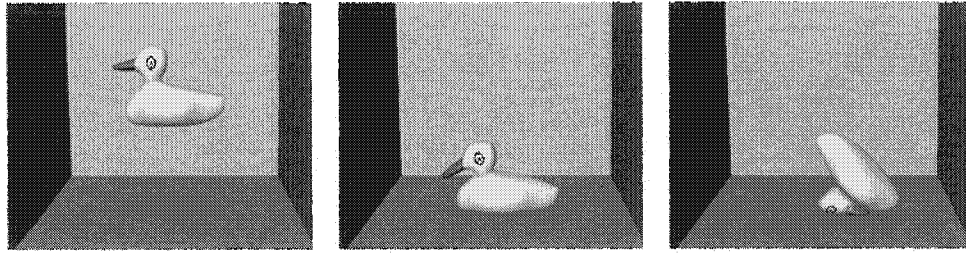


Figure 4.5 A collection of frames from an interaction with a duck model. The duck is modeled directly as a MacCracken-Joy subdivision solid. The motion of the duck is limited by the floor and wall constraints.

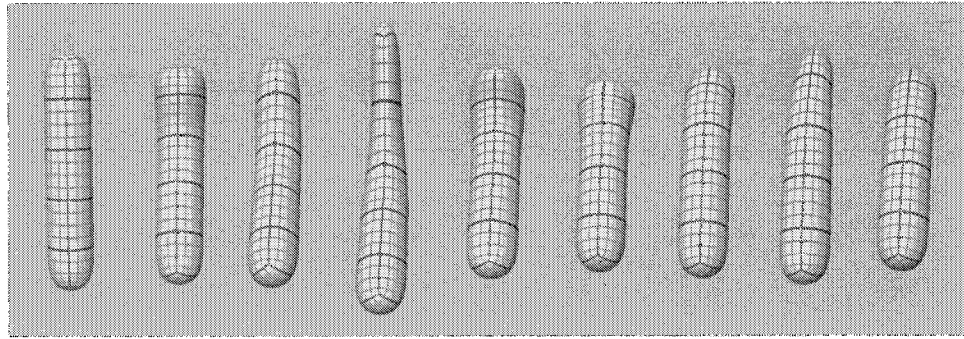


Figure 4.6 A cucumber-like object (modeled as a MacCracken-Joy subdivision solid), with a longitudinally varying shear modulus G . The object is being shaken by the bottom using a position constraint. The bottom is firm and maintains its shape, while the top deforms drastically.

object with sharp surface features.

Virtual Environments. We have implemented a rudimentary collision detection scheme to demonstrate the feasibility of placing our objects in a virtual environment. We use surface constraints to stop vertices on the model from passing through walls in the environment. In Figure 4.5 a duck is being tossed about in a box interactively. Our quasi-linearization scheme performed as expected; as long as deformations are modest the results appear realistic.

Varying Material Properties. Our system supports material properties that vary both spatially and temporally. Material properties are incorporated during the computation of the stiffness and mass matrices. During the quadrature phase, the values for ν , G , and ρ need not be constant. In ad-

dition, because our basis is hierarchical, material properties are smoothly factored into the the mass and stiffness matrices at all levels. For a particular generalized coordinate, the material properties at all points in the support of its associated basis function are factored into the computation of the mass and stiffness matrices. Consequently, the material properties of the object are correctly modeled, even when only a subset of the basis is used in the simulation. For instance, the shear modulus G of the object in Figure 4.6 varies along its length. When it is shaken, one end wobbles like soft rubber while the other remains almost rigid. An easy way to represent varying material properties over the body is to use the coarse subdivision basis. Then we need only specify control values at the coarse level vertices. The subdivision rules generate values throughout the object.

We also allow the shear modulus G to be scaled globally at runtime. This does not require re-computation because it uniformly scales the entire stiffness matrix. Scaling G globally makes the object seem more or less firm.

Comparison with Embedding in a Regular Grid. Figure 4.7 shows a comparison between our method and one in which the object is embedded in a regular grid. In the simulation, position constraints were used to stretch the dragon (by pulling on its front and back) and open its mouth. Despite having more degrees of freedom, and thus requiring more computation time, the simulation based on a regular grid is less convincing. Rather than having the mouth open and the body uncoil, as we would expect, the mouth and body seem to stretch uniformly. This effect is caused by basis functions whose support spans the empty regions adjacent to distinct parts of the dragon, thus correlating the motion of parts that would not naturally move together. In contrast, our method produces a more natural deformation because the grid can be made to fit the object much more closely.

Adaptation. Figure 4.8 illustrates our adaptive simulation algorithm applied to the dragon model. The coarsest level basis \mathcal{B}_0 has 88 basis functions, while the finest level basis \mathcal{B}_2 has 2245 elements. Using the quasi-linear solver, the simulation took about 0.02 seconds per frame when only the coarse basis is active. At the level of adaptation shown, each frame required about 0.1 seconds.

Summary

The collection of techniques presented in this chapter enable high-quality deformation of complex objects at interactive rates. Embedding allows complex objects to be simulated coarsely, while the

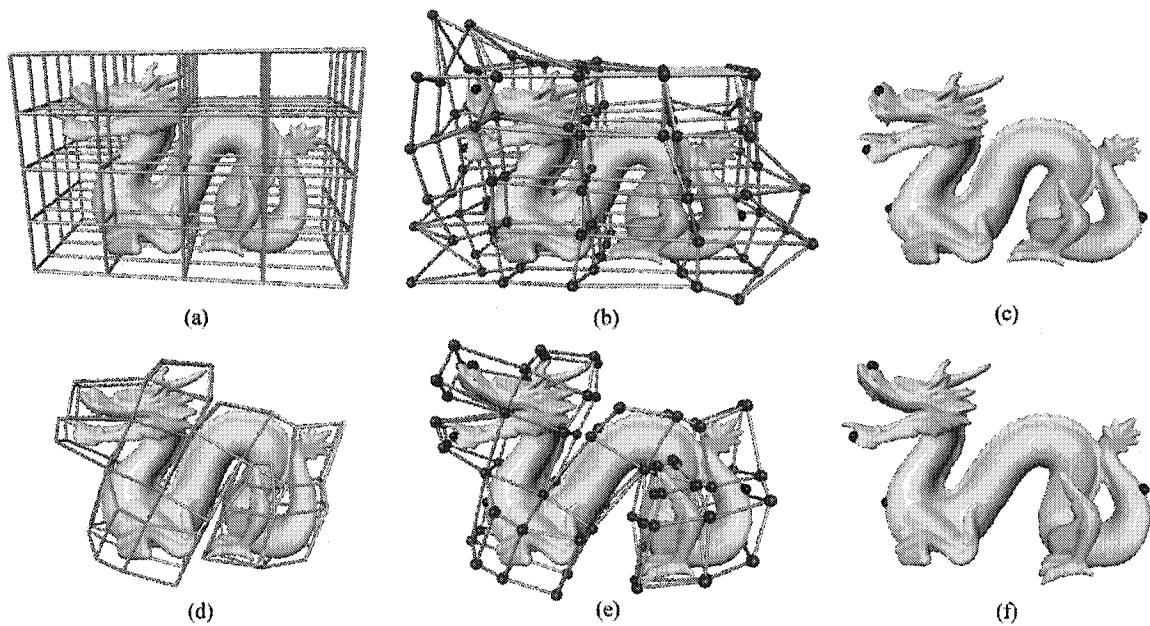


Figure 4.7 A comparison of simulations using a regular grid (top row) and a non-regular grid (bottom row), both using trilinear basis functions. The left images show the dragon in its rest state surrounded by (a) a regular grid (375 degrees of freedom) and (d) a non-regular grid (264 degrees of freedom). The center images show a simulation in which position constraints have been used to stretch the dragon longitudinally and open its mouth. Black spheres represent position constraints and red spheres represent the degrees of freedom of the system. In both cases, trilinear basis functions were used. The rightmost images show the deformed state of the dragon.

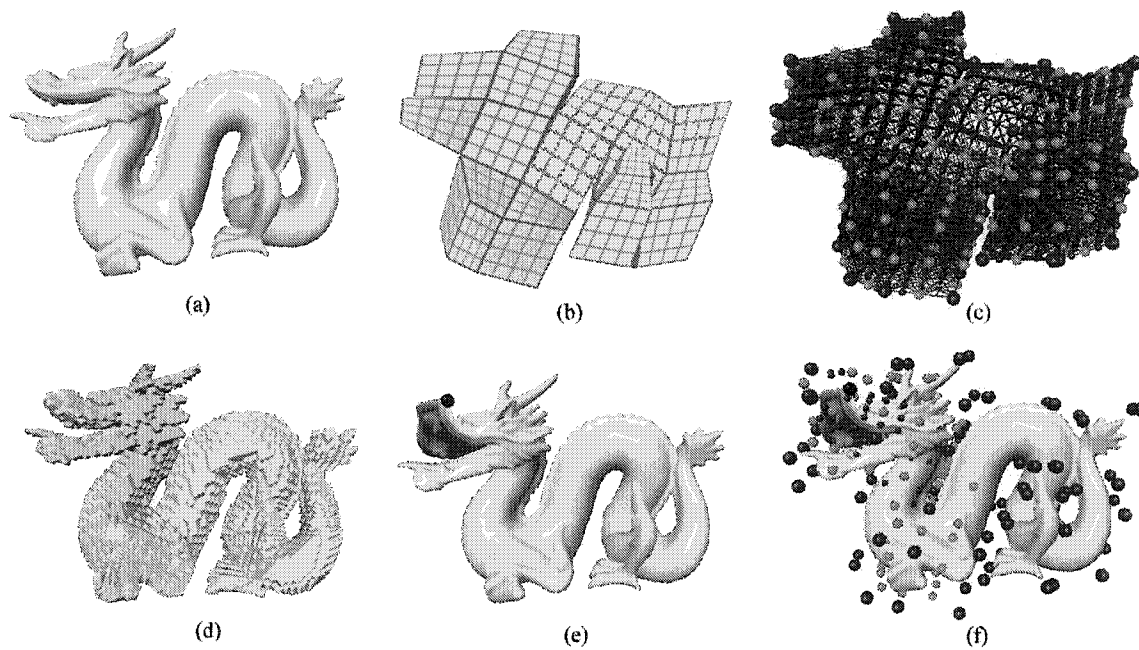


Figure 4.8 (a) The dragon in its rest state. (b) A trilinear subdivision volume that embeds the dragon. The surface has been textured to indicate how the volume is parameterized. (c) The hierarchical structure of the subdivision volume. Red spheres correspond to level 0 basis functions, green to level 1, and blue to level 2. (d) The tetrahedral approximation of the dragon that is used to compute integrals over its interior (i.e., for numerical quadrature). (e) The dragon being deformed by a position constraint pulling on the upper lip. (f) The basis functions introduced by the adaptive solver.

use of subdivision solids provides the flexibility needed to fit the coarsely parameterized domain to the object. By linearizing strain about a rotated rest state we achieve interactive rates while allowing the object to rotate freely. Adaptive simulation using a hierarchical basis allows detail to be added to the simulation where and when it is needed.

However, the system presented in this chapter has two significant deficiencies in the context of character animation. Our linearization scheme assumes that the state of the object is only a modest deformation from a global rotation of the rest state. This assumption is invalid for characters such as humans. For example, when an arm bends, the upper arm and lower arm are only modestly deformed, but they are oriented differently. The other problem is one of control. Our only control mechanism is to constrain the motion of a small number of points on the surface of the body, which is not a very effective way to dictate useful configurations. In the next chapter we address these issues.

Chapter 5

SKELETON-DRIVEN DEFORMATION

While the framework presented thus far produces interactive animation of elastic objects, it is hard to imagine using it for character animation because there are no mechanisms for an animator to control the shape of the object except by using general constraints. In this chapter we augment the framework by providing skeletal control of the simulated objects.

Controlling characters by posing an underlying skeleton is at the core of modern animation practice [2]. Typically, at each keyframe the character is posed by setting the joint angles of the skeleton, thereby positioning and orienting each bone. The shape of the character is determined primarily by a *skinning* procedure that deforms the surface of the character according to the pose of the skeleton. The final surface results from then applying additional deformations to the skinned surface.

Our goal is to provide an analogous skeletal control mechanism in our simulation framework. We would like the shape of the character to follow the motion of a skeleton that is dictated by the animator, and at the same time exhibit realistic elastic dynamics. To accomplish this we address three issues, each of which is critical in order to produce realistic results within the computational limits imposed by interactivity:

- *Skeletal constraints* – We introduce an efficient mechanism of constraining the motion of the elastic body to the prescribed motion of a piecewise linear skeleton (Section 5.1).
- *Pose-dependent linearization* – We linearize the equations of elasticity using rotations predicted from the configuration of the skeleton (Section 5.2).
- *Bone deformation energy* – To reproduce the effect of volumetric bones, we introduce a potential energy that penalizes deformation in the neighborhood of the skeleton (Section 5.3).

Following the three sections dedicated to the above topics, we discuss the instrumentation of models in Section 5.4, and results in Section 5.5.

5.1 Skeletal Constraints

Since our goal is to provide the animator with traditional skeletal control, we augment our framework with a traditional animation skeleton in the form of a transformation hierarchy. We can view the skeleton as a union of bones

$$S = \bigcup_{b=1}^{N_B} B_b,$$

where $B_b \subset \Omega$ is a line segment. The skeleton is posed by setting a vector of *pose parameters* $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_{N_\alpha}) \in \mathbb{R}^{N_\alpha}$, which may be Euler angles, quaternions, etc. The pose parameters induce an affine mapping associated with each bone

$$\mathbf{T}_b(\alpha) : \mathbb{R}^3 \rightarrow \mathbb{R}^3 : \mathbf{x} \mapsto \mathbf{R}_b \mathbf{x} + \mathbf{c}_b(\mathbf{x}), \quad (5.1)$$

where \mathbf{R}_b is a rotation matrix and $\mathbf{c}_b(\mathbf{x}) = \mathbf{c}_b$ is a translation (a constant displacement field). It is important to express translational displacement fields such as \mathbf{c}_b as a function of \mathbf{x} because we will need to expand them against the basis \mathcal{B} as follows:

$$\mathbf{c}_b(\mathbf{x}) = \sum_{i=1}^N \hat{\mathbf{c}}_{b,i} \phi_i(\mathbf{x}). \quad (5.2)$$

Since the coarse basis functions form a partition of unity,

$$\hat{\mathbf{c}}_{b,i} = \begin{cases} \mathbf{c}_b & \text{for } i \leq N_0 \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

An animation of S involves varying the pose parameters over time: $\alpha = \alpha(t)$. This induces a time dependent function

$$\mathbf{p}_S : S \times \mathbb{R} \rightarrow \mathbb{R}^3$$

that positions the points on the skeleton over time. Rigidity of the bones implies that \mathbf{p}_S is an isometry on each edge of S . In particular, \mathbf{p}_S is a piecewise linear function on S . In order for the motion of the character to conform to the motion of the skeleton, \mathbf{p}_S must be the restriction of \mathbf{p} to S (recall Equation (3.1)). The function $\mathbf{p}(\mathbf{x}, t)$ is then the solution of a system of partial differential

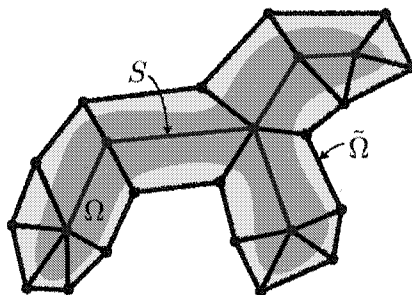


Figure 5.1 An object $\Omega \subset \mathbb{R}^3$ instrumented with a skeleton S that is aligned with the control lattice that determines the domain $\tilde{\Omega} \supset \Omega$.

equations, subject to the constraint $\mathbf{p}(\mathbf{x}, t) = \mathbf{p}_S(\mathbf{x}, t)$ for $\mathbf{x} \in S$.

One possible implementation of the skeletal constraint would be to use Lagrange multipliers as discussed in Chapter 3, but it is a poor solution for a variety of reasons. First, the chosen basis is unlikely to include functions that allow the constraint to be met (the basis would have to allow creases along arbitrary line segments, and derivative discontinuities where bones meet). Second, a constraint function in the form of Equation (3.13) is complicated for the skeleton because a continuum of points is involved. Third, since many of the elements of \mathbf{q} affect \mathbf{p}_S , the system in Equation (3.14) becomes much larger than without the constraint.

Our approach is to align the skeleton with the control lattice as shown in Figure 5.1. Consider some useful properties of the trilinear basis. Because it is an interpolating basis, $\mathbf{d}(\mathbf{x}_i, t) = \mathbf{q}_i(t)$ for $1 \leq i \leq N$. Additionally, functions composed from the trilinear basis are linear along the edges of the control lattice. Together these conditions imply that we can constrain the body to the skeleton by simply constraining the coefficients $\mathbf{q}_i(t)$, for $\mathbf{x}_i \in S$, to the motion of the skeleton dictated by $\boldsymbol{\alpha}(t)$. The remaining coefficients are then determined by solving the elastodynamic equations.

The MLCA subdivision scheme has a similar property: if an edge and its adjoining vertices are tagged as *sharp*, a rigid transformation of the endpoints of the edge always corresponds to an isometry on the edge. So when using the MLCA scheme we can also constrain the skeleton simply by constraining a subset of the coefficients. Our scheme for constraining the motion of the body to the skeleton takes advantage of the ability of the trilinear and MLCA bases to meet linear constraints along the edges of the control lattice. Thus, for the remainder of this work we will no longer consider

the MacCracken-Joy scheme.

To simplify notation we introduce the functions *skel* and *free* to select the indices of the constrained and free control vertices, respectively:

$$\begin{aligned} \text{skel}(s) &= i, \text{ where } \mathbf{x}_i \text{ is the } s\text{-th control vertex s.t. } \mathbf{x}_i \in S \\ \text{free}(f) &= i, \text{ where } \mathbf{x}_i \text{ is the } f\text{-th control vertex s.t. } \mathbf{x}_i \in K \setminus S. \end{aligned}$$

The constrained vertices then form the set $\{\mathbf{x}_{\text{skel}(s)} : s = 1, 2, \dots, N_S\}$ and the free vertices form the set $\{\mathbf{x}_{\text{free}(f)} : f = 1, 2, \dots, N_F\}$. We partition the vector \mathbf{q} into the vectors \mathbf{q}_S and \mathbf{q}_F , where

$$\begin{aligned} (\mathbf{q}_S)_s &= \mathbf{q}_{\text{skel}(s)}, \text{ for } 1 \leq s \leq N_S, \text{ and} \\ (\mathbf{q}_F)_f &= \mathbf{q}_{\text{free}(f)}, \text{ for } 1 \leq f \leq N_F. \end{aligned}$$

The vector \mathbf{q}_S contains the variables constrained to the motion of the skeleton and the vector \mathbf{q}_F contains the free variables. As we have already observed, $\mathbf{q}_S(t)$ is determined by $\alpha(t)$. The vector $\mathbf{q}_F(t)$ is determined by solving the Euler-Lagrange equation:

$$\frac{d}{dt} \left(\frac{\partial T}{\partial \dot{\mathbf{q}}_F} \right) + \frac{\partial U}{\partial \mathbf{q}_F} + \mu \dot{\mathbf{q}}_F = \mathbf{Q}_F, \quad (5.3)$$

where

$$(\mathbf{Q}_F)_f = \mathbf{Q}_{\text{free}(f)}. \quad (5.4)$$

We partition the mass matrix into \mathbf{M}_{SS} , \mathbf{M}_{SF} , \mathbf{M}_{FS} , and \mathbf{M}_{FF} as follows:

$$\begin{aligned} (\mathbf{M}_{SS})_{st} &= \mathbf{M}_{\text{skel}(s), \text{skel}(t)}, \\ (\mathbf{M}_{SF})_{sg} &= \mathbf{M}_{\text{skel}(s), \text{free}(g)}, \\ (\mathbf{M}_{FS})_{ft} &= \mathbf{M}_{\text{free}(f), \text{skel}(t)}, \text{ and} \\ (\mathbf{M}_{FF})_{fg} &= \mathbf{M}_{\text{free}(f), \text{free}(g)}, \end{aligned} \quad (5.5)$$

where \mathbf{M}_{SS} is of dimension $N_S \times N_S$, \mathbf{M}_{SF} is $N_S \times N_F$, \mathbf{M}_{FS} is $N_F \times N_S$, and \mathbf{M}_{FF} is $N_F \times N_F$,

each containing 3×3 matrices as elements. It follows that

$$\frac{d}{dt} \left(\frac{\partial T}{\partial \dot{\mathbf{q}}_F} \right) = \mathbf{M}_{FS} \ddot{\mathbf{q}}_S + \mathbf{M}_{FF} \ddot{\mathbf{q}}_F,$$

so Equation (5.3) becomes the system of ordinary differential equations

$$\mathbf{M}_{FF} \ddot{\mathbf{q}}_F = \mathbf{Q}_F - \mu \dot{\mathbf{q}}_F - \frac{\partial U}{\partial \mathbf{q}_F} - \mathbf{M}_{FS} \ddot{\mathbf{q}}_S. \quad (5.6)$$

We can now solve Equation (5.6) using the implicit method described in Section 4.5. By aligning the skeleton with the control lattice, we have reduced the system to size N_F and eliminated the need for a complicated Lagrangian constraint. However, Equation (5.6) is nonlinear because U is a fourth degree polynomial in \mathbf{q} (and thus \mathbf{q}_F). As in Chapter 4, the system must be simplified in order to simulate at interactive speeds using more than a few control vertices. In the next section we will discuss a novel linearization procedure that is appropriate for skeleton-driven character animation.

5.2 Pose-Dependent Linearization

The major bottleneck in solving the nonlinear system in Equation (5.6) is the computation of the stiffness matrix at each timestep. A well-known simplification is to linearize the strain tensor by dropping the last term in Equation (3.9), which results in a quadratic elastic potential and thus a constant stiffness matrix (which is composed of the first three addends in Equation (A.3)). The resulting system of equations is

$$\mathbf{M}_{FF} \ddot{\mathbf{q}}_F = \mathbf{Q}_F - \mu \dot{\mathbf{q}}_F - \mathbf{M}_{FS} \ddot{\mathbf{q}}_S - \mathbf{S}_{FF} \mathbf{q}_F - \mathbf{S}_{FS} \mathbf{q}_S,$$

where the stiffness matrix \mathbf{S} , evaluated at $\mathbf{q} = 0$, has been partitioned as in Equation (5.5). As compared to other simplifications, such as using a mass-spring-based elastic potential, linearization of strain has the advantage that it is a very good approximation of nonlinear elasticity, but only when the deformation is small; for large deformations, severe distortions occur. The method introduced in Section 4.3 allows the object to undergo a large rotation, but does not ease the restriction that the deformation be modest. For skeletally-controlled characters, this restriction is inappropriate because

a single rotation is not a good approximation of the state; large deformations occur at the joints.

In this section we present two new methods of linearizing the equations of motion for skeletally controlled characters. Both procedures are based on the observation that the soft tissues of vertebrates do not typically undergo large deformations *relative to nearby bones*. Based on this assertion, we use the configuration of the skeleton to predict the shape of the deformed object.

5.2.1 Blended Local Linearization

Our first approach is to divide the object into regions, each of which can be simulated using the linear strain model. Each region is associated with a bone of the skeleton. The idea is to use the transformation of the bone to predict the rotation of the region, and linearize about the predicted rotated state similar to the scheme presented in Section 4.3.

The user divides the object into regions by assigning weights to the control vertices, forming a partition of unity over the object. A piece of the object can belong to a single region or can be divided fractionally among several regions. We encode the region weights in the function $W(b, i)$, which returns the weight of vertex i in region b . The lower right image in Figure 5.3 shows a partitioned object, colored according to the region assignments. Our current system requires that the user manually set the weights for each control vertex¹.

From the region assignments we form a cell complex K_b corresponding to region b . The cell complex K_b contains the cells of K whose vertices all have positive weights for region b :

$$K_b = \{C \in K : \forall i \in v(C), W(b, i) > 0\}, \quad (5.7)$$

where $v(C)$ is the set of indices of vertices of cell C . Each region has an associated function space

$$\mathcal{B}_b = \{\phi_i|_{K_b} : \phi_i \in \mathcal{B}, \phi_i|_{K_b} \neq 0\}, \quad (5.8)$$

where $\phi_i|_{K_b}$ denotes the restriction of ϕ_i to K_b .

Because each basis function is associated with a unique control vertex, there is an injective map

¹A more intuitive painting interface would be a better way to assign region weights. It would also be helpful to automate the task of region assignment (the work of Li et al. [47] may be adaptable to our problem domain).

from \mathcal{B}_b to \mathcal{B} . In order to perform local computation within each region, we renumber the vertices within each region to have indices in the range $1, \dots, N_b$, where N_b is the number of vertices in K_b . Let the function $i(b, \bar{i})$ map the \bar{i} -th vertex of K^b to the corresponding vertex of K .

The pseudocode for taking a single simulation step is:

Algorithm 5.1: Simulation Step for Blended Local Linearization

```

1  foreach region  $b$  do
2      for  $\bar{i} \leftarrow 1$  to  $N_b$  do
             $\mathbf{q}_{\bar{i}}^b \leftarrow \mathbf{R}_b^T \left( \mathbf{r}_{i(b, \bar{i})} + \mathbf{q}_{i(b, \bar{i})} - \hat{\mathbf{c}}_{b, i(b, \bar{i})} \right) - \mathbf{r}_{i(b, \bar{i})}$ 
             $\dot{\mathbf{q}}_{\bar{i}}^b \leftarrow \mathbf{R}_b^T \dot{\mathbf{q}}_{i(b, \bar{i})}$ 
        end
3      Solve for  $\Delta \mathbf{v}^b$  using Equation (4.5) and inputs  $\mathbf{q}^b$  and  $\dot{\mathbf{q}}^b$ 
       $\Delta \mathbf{v} \leftarrow \mathbf{0}$ 
4      for  $\bar{i} \leftarrow 1$  to  $N_b$  do
             $\Delta \mathbf{v}_{i(b, \bar{i})} \leftarrow \Delta \mathbf{v}_{i(b, \bar{i})} + \mathbf{R}_b W(b, i(b, \bar{i})) \Delta \mathbf{v}_{\bar{i}}^b$ 
        end
      end
5   $\dot{\mathbf{q}} \leftarrow \dot{\mathbf{q}} + \Delta \mathbf{v}$ 
6   $\mathbf{q} \leftarrow \mathbf{q} + h \dot{\mathbf{q}}$ 

```

The main loop that begins at line 1 computes a $\Delta \mathbf{v}^b$ for each region and merges them into a global $\Delta \mathbf{v}$. First, at line 2, \mathbf{q} and $\dot{\mathbf{q}}$ are converted to the local coordinate frame of the region, and placed into the vectors \mathbf{q}^b and $\dot{\mathbf{q}}^b$, which are indexed using the local indexing scheme for the region. Note that \mathbf{R}_b is the rotation of the bone from Equation (5.1), and $\hat{\mathbf{c}}_{b, i}$ are the coefficients of the bone translation from Equation (5.2). Line 3 then solves the dynamic equations from Equation (4.5), ignorant of the fact that the computation is in a local frame. The result is a $\Delta \mathbf{v}^b$ for the region, in its local frame of reference. The loop at line 4 merges $\Delta \mathbf{v}^b$ into the global solution $\Delta \mathbf{v}$ by transforming the coordinate frame, applying the region weights, and mapping regional indices to global indices. The state of the global system is updated in lines 5 and 6.

The blended local linearization scheme produces much better results than pure linearization. A character can be posed in a variety of configurations without gross distortions occurring. However,

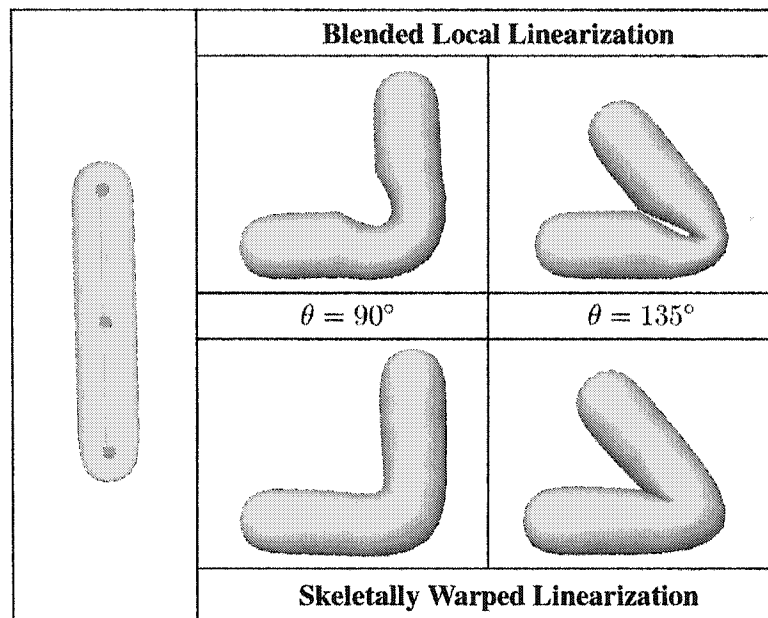


Figure 5.2 Comparison between *blended local linearization* and *skeletally warped linearization*. On the left is a simple object at rest, instrumented with a skeleton having a single joint in the middle. For each of the schemes, the joint has been posed at 90° and 135° . On the top row are the results using blended local linearization. Notice how the region near the bend unrealistically contracts, similar to traditional skinning methods used in animation. On the bottom row are the results using skeletally warped linearization, which look much more plausible.

the method fails to produce pleasing results in the areas where regions overlap, as can be seen in Figure 5.2. In such regions, the resulting static equilibrium states resemble the results of traditional skinning; near joints, the body becomes unrealistically narrow. The procedure introduced next eliminates this problem and offers additional benefits.

5.2.2 *Skeletally Warped Linearization*

We will now introduce a method that builds on the work of Müller et al. [57], in which $\frac{\partial U}{\partial \mathbf{q}}$ is linearized independently in the neighborhood of each vertex. In their scheme, an estimate of rotation is computed from the displacement field at each vertex. The rotation is used to linearize the neighborhood of the vertex. Their technique produces pleasing deformations without the gross distortions that accompany traditional linearization, but we have found it to suffer from some instability due to its inherent nonlinearity (the linearization is a function of the current configuration of the body).

Our method borrows from Müller et al. [57] the idea of performing the linearization independently at each vertex, but differs in how the linearization is performed. Our method is fast, stable, and free of gross distortions. A major advantage of our technique is that it results in a pose-dependent linear system: the system is linear for a fixed configuration of the skeleton. Pose-dependent linearity makes it computationally inexpensive to perform dynamic simulation or compute static equilibrium solutions given external forces (we will use such solutions later in this work to perform interactive optimization). In contrast, solving for static equilibrium using the method of Müller et al. is a nonlinear problem. A second advantage is that since we predict both the rotation and translation of the body (Müller et al. only estimate the rotation), our potential energy need not be translation invariant (in Section 5.3 we add an important translation-dependent potential).

Before discussing our method in detail, it is worthwhile considering the special case where at time t the pose parameters $\alpha(t)$ induce a rigid motion of the skeleton S , say

$$\mathbf{T}(\mathbf{x}) = \mathbf{R}\mathbf{x} + \mathbf{c}(\mathbf{x}),$$

where \mathbf{R} is a rotation matrix and $\mathbf{c}(\mathbf{x}) = \mathbf{c}$ is a translational displacement field. We may then write

$$\mathbf{x} + \mathbf{d}(\mathbf{x}, t) = \mathbf{R}(\mathbf{x} + \bar{\mathbf{d}}(\mathbf{x}, t)) + \mathbf{c}(\mathbf{x}),$$

where $\bar{\mathbf{d}}$ is the displacement of the object relative to the transformation of the skeleton. The mapping $\mathbf{p} : \Omega \rightarrow \mathbb{R}^3$ will be an approximately rigid motion (and therefore have small elastic energy) if and only if $\bar{\mathbf{d}}$ is small. Writing

$$\bar{\mathbf{d}}(\mathbf{x}, t) = \sum_{i=1}^N \bar{\mathbf{q}}_i(t) \phi_i(\mathbf{x}),$$

expressing U as a function of $\bar{\mathbf{q}}$, linearizing U about $\bar{\mathbf{q}} = 0$, and reverting back to the original generalized coordinates \mathbf{q} , gives the following approximation for $\frac{\partial U}{\partial \mathbf{q}}$, which we will refer to in the future as \mathbf{Q}^e :

$$\mathbf{Q}_i^e = \sum_{j=1}^N \mathbf{R} \mathbf{S}_{ij} (\mathbf{R}^T (\mathbf{x}_j + \mathbf{q}_j - \hat{\mathbf{c}}_j) - \mathbf{x}_j), \quad (5.9)$$

where

$$\mathbf{c}(\mathbf{x}) = \sum_{j=1}^N \hat{\mathbf{c}}_j \phi_j(\mathbf{x})$$

is the expansion of the translation $\mathbf{c}(\mathbf{x})$ against the basis \mathcal{B} . Substituting Equation (5.9) (via Equation (5.4)) into Equation (5.6) gives a linearization which can be used to compute \mathbf{q} at time $t + \Delta t$.

In general, however, the pose parameters α will deform the skeleton, and the transformation $\mathbf{T}(\mathbf{x})$ will, therefore, depend on α . In this case, our approach is to use a *skinning* methodology to predict $\mathbf{T}(\mathbf{x})$.

Recall from Equation (5.1) that $\alpha(t)$ determines a set of mappings that position and orient each bone B_b in \mathbb{R}^3 . We associate each vertex \mathbf{x}_i of K with two bones of S , denoted B_{i1} and B_{i2} and we let

$$\mathbf{T}_i(\mathbf{x}) = \mathbf{R}_i \mathbf{x} + \mathbf{c}_i,$$

where $\mathbf{R}_i = \omega_{i1} \mathbf{R}_{i1} + \omega_{i2} \mathbf{R}_{i2}$ and $\mathbf{c}_i = \omega_{i1} \mathbf{c}_{i1} + \omega_{i2} \mathbf{c}_{i2}$. The user-defined weights ω_{ik} dictate how much each bone affects each point. The sum in the formula for \mathbf{R}_i is shorthand for interpolation of rotations. In our current implementation we allow only two nonzero weights so that spherical linear interpolation can be used. More general methods are available for blending more than two rotations, at higher computational cost [1, 16]. We note that the idea of using skinning weights to blend rotations comes from [3].

We use the mapping \mathbf{T}_i to transform each vertex in a neighborhood of \mathbf{x}_i before computing the elastic force:

$$\mathbf{Q}_i^e = \sum_{j=1}^N \mathbf{R}_i \mathbf{S}_{ij} \left(\mathbf{R}_i^T (\mathbf{x}_j + \mathbf{q}_j - \hat{\mathbf{c}}_{i,j}) - \mathbf{x}_j \right) .$$

The generalized elastic force for a free variable is then

$$\begin{aligned} (\mathbf{Q}_F^e)_f &= \sum_{s=1}^{N_S} \mathbf{R}_{free(f)} (\mathbf{S}_{FS})_{fs} \left(\mathbf{R}_{free(f)}^T (\mathbf{x}_{skel(s)} + (\mathbf{q}_S)_s - \hat{\mathbf{c}}_{free(f),skel(s)}) - \mathbf{x}_{skel(s)} \right) \\ &+ \sum_{g=1}^{N_F} \mathbf{R}_{free(f)} (\mathbf{S}_{FF})_{fg} \left(\mathbf{R}_{free(f)}^T (\mathbf{x}_{free(g)} + (\mathbf{q}_F)_g - \hat{\mathbf{c}}_{free(f),free(g)}) - \mathbf{x}_{free(g)} \right) . \end{aligned} \quad (5.10)$$

Noting that Equation (5.10) is linear in \mathbf{q}_F , and that $\mathbf{R}_i = \mathbf{R}_i(\alpha(t))$ and $\mathbf{c}_i = \mathbf{c}_i(\alpha(t))$, one can see that substituting Equation (5.10) into Equation (5.6) results in a pose-dependent, and thus time-dependent, linear system in \mathbf{q}_F . The resulting system is positive definite but no longer symmetric, so we solve it using the bi-conjugate gradients algorithm [65].

The results of using skeletally warped linearization are shown in Figure 5.2. The resulting deformations near joints are clearly more plausible than those produced by the blended local linearization scheme.

5.2.3 Smoothing Trilinear Deformations

It was mentioned in Section 5.1 that either the trilinear basis or the MLCA subdivision basis can be used in the presence of skeletal constraints. The trilinear basis functions have the advantage that they have smaller support than those of the MLCA scheme, so the system is sparser, leading to faster computation. The MLCA basis has the advantage that the deformations are mostly smooth², whereas trilinear deformations have unappealing creases.

Our linearization scheme assumes that within the support of a basis function the body has undergone a constant rigid body motion. We found that this caused excessive distortion when smooth basis functions were used on a coarse basis, but we wanted a coarse basis to maintain interactivity. We therefore chose to use MLCA basis functions when reconstructing the state of the object for display, but trilinear basis functions for computing the dynamics. We factor in the smooth recon-

²The MLCA scheme is provably smooth everywhere except at extraordinary vertices, where it is not known if the scheme is smooth, but there is evidence that it is [8].

struction when computing the effects of constraints on the surface so that constraints are met exactly in the presence of smoothing.

5.3 Bone Displacement Energy

In natural creatures with three-dimensional bones, the flesh cannot twist (i.e., rotate) around the axis of the bone without causing the flesh to deform. Such deformations are resisted by emergent elastic forces, so the twisting is limited. But flesh can rotate about a line constraint without deforming. To avoid such unnaturally free movement, we introduce a potential energy to penalize all displacement (not just deformation) within a fixed radius of the bones. We denote this region $\Omega_\beta \subset \Omega$. The following potential describes the constraint:

$$U^\beta = \frac{1}{2} \int_{\Omega_\beta} \bar{\mathbf{d}}(\mathbf{x}) \cdot \bar{\mathbf{d}}(\mathbf{x}) dV. \quad (5.11)$$

The above potential is quadratic, so its Hessian is simply a constant matrix:

$$\frac{\partial^2 U^\beta}{\partial \bar{\mathbf{q}}_i \partial \bar{\mathbf{q}}_j} = \mathbf{I} \int_{\Omega_\beta} \phi_i(\mathbf{x}) \phi_j(\mathbf{x}) dV, \quad (5.12)$$

where \mathbf{I} is a 3×3 identity matrix. The above constraint must be computed in the frame of reference of the rigidly transformed bone, which fits well into our bone-relative computation framework.

5.4 Instrumenting Characters

Prior to simulation, a model must be instrumented with a skeleton and control lattice. Although automated skeleton construction has been addressed by Teichmann and Teller [77], we let the animator specify the skeleton in order to achieve the desired level of control. We have implemented a simple system that allows a skeleton to be constructed manually in just a few minutes. The user creates a joint by clicking on the object with the mouse. If the ray through the mouse point (from the camera projection center) intersects the object at least twice, a joint is placed midway between the first two intersections. This positioning scheme produces joints that are centrally located inside the object. The joints can then be dragged and dropped for final positioning. Two joints can be selected to define a bone, and with the selection of a root joint, a transformation hierarchy can be created

automatically.

We currently use a constructive procedure that allows the user to build the control lattice interactively by adding cells incrementally and repositioning the control vertices as needed. Several hours are required for an experienced user to create a moderately complex control lattice. The abundance of volumetric meshing schemes suggests that automatic creation of the control lattice is possible, and we hope to address this problem in the future. Figure 5.3 shows the skeleton and control mesh for a kangaroo model.

5.5 Results

We applied skeleton-driven deformation to two meshes: a cow and a kangaroo. Both were freely available on the Internet. The control lattice for the kangaroo model has 448 cells and 177 vertices; the cow control lattice has 572 cells and 214 vertices. On a 1 Ghz PC, both the cow and kangaroo animated at about 100 Hz using only the coarse basis functions, which is clearly within range for interactive applications (with adaptation, simulation time varies depending on the degree of adaptation required). Figure 5.4 shows frames of an animation of the cow model (using the coarse basis), which demonstrates the ability of our system to handle variable material properties; the ears flop around realistically while the horns stay rigid. Our system is able to produce plausible interactive simulations for bodies with varying material properties because the control lattice can be carefully crafted to respect material boundaries, and because our integration over the body takes variable material properties into account.

For our datasets, the blended local linear and global linear solutions required about the same amount of computation time. Yet the blended local linear solution produced much more pleasing results, as demonstrated in Figure 5.5. The blended local linear solution looks similar to the fully nonlinear solution, while the global linear solution is badly warped.

The results presented here were produced using blended local linearization because skeletally warped linearization was developed later. We presented both schemes in this chapter for clarity. The improvements offered by the skeletally warped linearization scheme were needed by the work presented in the following chapter, which uses that scheme exclusively.

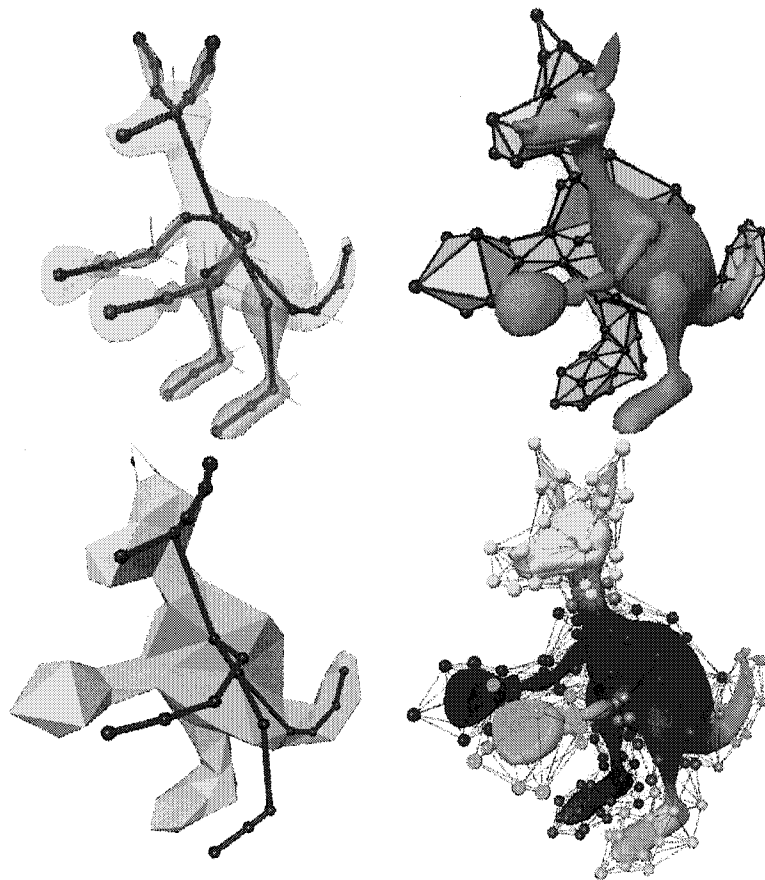


Figure 5.3 The upper left image shows an input model that has been instrumented with a skeleton. A coordinate frame is visible at the base of each bone for which a region is defined. The upper right image shows the model embedded in a control lattice (only half of the control lattice is drawn). The lower left image shows how the skeleton coincides with edges and vertices of the control lattice. The lower right image shows the entire control lattice in wireframe, as well as the division of the object into regions for local linearization. Each region is associated with one of the local coordinate systems in the upper left image. Note the color blending where regions overlap.

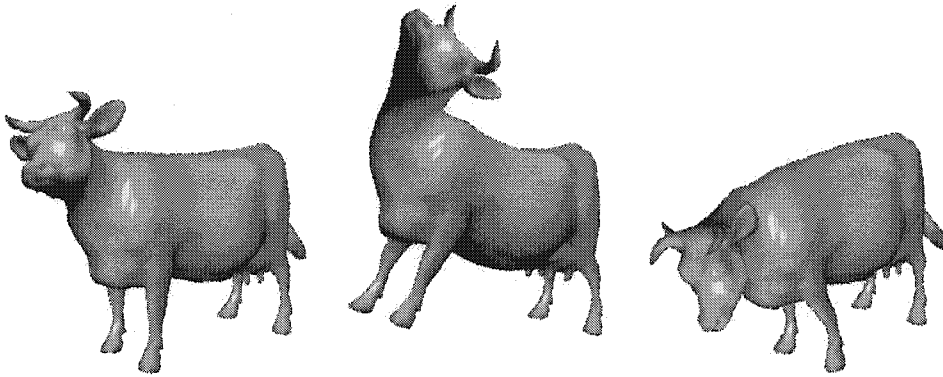


Figure 5.4 Frames from an interactive animation. There is no noticeable warping due to strain linearization, and the different materials (e.g., ears, horns) behave distinctly.

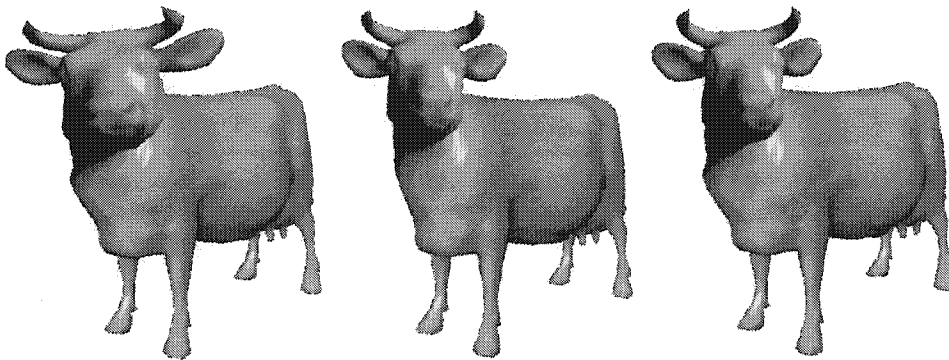


Figure 5.5 On the left is the global linear solution, which shows significant warping when the cow turns its head to one side. In the center is the fully nonlinear solution. On the right is the blended local linear solution, which shows no noticeable warping of the head. A slight protrusion can be seen in the neck of the right image due to region blending.

Summary

The techniques presented in this chapter demonstrate that interactive character animation can be achieved in an elastic deformation setting. Skeletal constraints provide the kind of skeletal control that animators already use heavily. Bone displacement energy makes the one-dimensional bone constraints produce an effect more like three-dimensional bones. By linearizing the strain in a bone-relative way we achieve interactive simulation rates with high quality deformations. In the next chapter we provide additional control mechanisms that further mirror the control paradigms in current use in animation.

Chapter 6

SHAPE CONTROL USING FORCES

The system presented in the previous chapter produces skeleton-driven character animation at interactive speeds. But it lacks the power of traditional animation systems to allow detailed shape control, because control is limited to the configuration of the underlying skeleton. Shape control is important because it enables the animator to create more realistic deformations and express the emotion and intent of a character. For example, skeletal controls will not effectively help the animator to make a character appear to breathe, produce a muscle bulge, or smile. To address this shortcoming, we introduce an additional control mechanism.

Since our framework is based on physical simulation, two natural mechanisms exist for influencing the shape: hard constraints and forces. Forces (or force fields) are better suited to our framework for a variety of reasons. First, hard constraints are rigid by definition, which is not a realistic model for any animal tissues other than bones (which we already model using hard constraints). Forces on the other hand can influence the shape without destroying its dynamic nature. Another reason that forces are more appealing is that they can be easily included in our simulation framework (via Equation (3.6)). Additionally, forces can coexist without troublesome compatibility issues; if two force fields overlap, their effects are simply summed. Finally, force fields need not line up exactly with the character's surface geometry. Our framework already handles the embedding of the character; if the force field extends beyond the interior of the character it can simply be restricted to the interior.

In choosing to use forces to effect deformations, we take advantage of a subtlety of constrained linearized elasticity. For a fixed skeletal pose, deformations and generalized forces are in correspondence due to pose-dependent linearization. Consider the body in static equilibrium. If we substitute Equation (5.10) into Equation (5.6) and apply the conditions of static equilibrium, $\dot{\mathbf{q}} = \mathbf{0}$ and $\ddot{\mathbf{q}} = \mathbf{0}$,

the resulting system is of the form

$$\mathbf{A}(\boldsymbol{\alpha})\mathbf{q}_F = \mathbf{b}(\boldsymbol{\alpha}) + \mathbf{Q}_F, \quad (6.1)$$

where $\mathbf{A}(\boldsymbol{\alpha})$ is a $N_F \times N_F$ invertible matrix containing 3×3 blocks, and $\mathbf{b}(\boldsymbol{\alpha})$ is an N_F -vector with elements that are 3-vectors. Since $\mathbf{A}(\boldsymbol{\alpha})$ is invertible¹, it creates a bijective linear map: every generalized force \mathbf{Q}_F has a unique corresponding displacement \mathbf{q}_F , and vice versa. This relationship is key to our approach to rigging; it allows us to think about shape in terms of forces. While our simulations are dynamic, the rigging is configured while running a static equilibrium solver (i.e., solving Equation (6.1)), where forces and displacements are in correspondence.

6.1 Overview

Recall from Chapter 5 that we control the animation of a character using a vector of pose parameters $\boldsymbol{\alpha}(t)$. We introduce an additional vector of *abstract parameters* $\boldsymbol{\beta}(t) = (\beta_1, \beta_2, \dots, \beta_{N_\beta}) \in \mathbb{R}^{N_\beta}$. The pose parameters and abstract parameters can be combined into a single vector of *control parameters*

$$\Theta = (\alpha_1, \alpha_2, \dots, \alpha_{N_\alpha}, \beta_1, \beta_2, \dots, \beta_{N_\beta}) = (\theta_1, \theta_2, \dots, \theta_{N_\theta}) \in \mathbb{R}^{N_\theta},$$

where $N_\theta = N_\alpha + N_\beta$. The components of $\boldsymbol{\beta}$ serve as general-purpose shape controllers. For example, one such parameter value could be named *raised left eyebrow* and be expected to change the shape of the character appropriately. The animator may then implement the raising of the left eyebrow via a curve $\boldsymbol{\beta}(t)$ in parameter space. An animation of the entire character is created by varying the control parameters $\Theta(t)$ over time.

The parameters Θ affect the shape of the character through objects that we call *rigs* (a variation on standard animation terminology). Each rig maps the control parameters to a force field acting on the body. The collection of all rigs on a character is thus modeled by a continuous family $\mathbf{f}^\Theta(\mathbf{x}, \Theta)$

¹The skeleton must provide a sufficient number of independent constraints in order to ensure that $\mathbf{A}(\boldsymbol{\alpha})$ is invertible. For example, in the unconstrained case, displacements that produce infinitesimal rigid motions all correspond to zero force. The skeleton must then constrain at least the six rigid degrees of freedom of the body. Any skeleton having at least two bones not on the same line will satisfy this condition. We can also check the determinant of $\mathbf{A}(\boldsymbol{\alpha})$ in the rest configuration of the skeleton ($\boldsymbol{\alpha} = \boldsymbol{\alpha}_0$) to ensure that it is invertible. If $\mathbf{A}(\boldsymbol{\alpha}_0)$ is invertible then $\mathbf{A}(\boldsymbol{\alpha})$ is invertible for all $\boldsymbol{\alpha}$ because the rotations introduced in Equation (5.10) do not change the rank of \mathbf{A} .

of force fields on Ω . Notice that we allow all of the elements of Θ to affect the shape, not just the elements of β . This allows the pose to affect the deformation, which is useful paradigm in animation. In particular, anatomical features of the shape such as muscle bulges are often pose dependent.

The remainder of this chapter discusses the rigging process, by which the mapping \mathbf{f}^Θ is constructed. We present two independent paradigms, based on two kinds of components: *force field templates* and *surface deformation rigs*.

Force field templates, which we will also refer to as *force templates* or simply *templates*, are reusable components representing parameterized force fields in a canonical coordinate frame. Attachment of a force template to the character is facilitated by a *template guide*, which simplifies the positioning and orienting of the template. Examples of force templates are presented in Section 6.4.1, and template guides are discussed in Section 6.4.2.

Surface deformation rigs (hereafter referred to as *surface rigs*) are rigs constructed from surface deformations. This gives the user more control by leveraging existing technologies such as surface scanners and geometric modeling software. Surface deformations are presented in Section 6.5.

The next two sections discuss how rig forces are simulated (Section 6.2), and the process of rigging a character (Section 6.3).

6.2 Simulating Rig Forces

The displacement of the character must be taken into consideration when applying the rigging force. For example, if the force is simply applied in the global frame of reference, a force effecting an outward surface bump would cause an inward dimple to form when the character is rotated by 180° . The force can be evaluated relative to the current state of the object by multiplying by the Jacobian of the current position. The formula for computing the generalized rig force is then

$$\mathbf{Q}_{F,f}(\Theta) = \int_{\Omega} \frac{\partial \mathbf{p}}{\partial \mathbf{x}} \mathbf{f}^\Theta(\mathbf{x}, \Theta) \phi_{free(f)}(\mathbf{x}) dV .$$

To reduce computation, we make the simplifying assumption that in the neighborhood of vertex i , $\frac{\partial \mathbf{p}}{\partial \mathbf{x}} \approx \mathbf{R}_i(\boldsymbol{\alpha})$ (recall Equation (5.1)). The resulting approximation of the generalized rig force is

$$\mathbf{Q}_{F,f}(\Theta) \approx \mathbf{R}_{free(f)}(\boldsymbol{\alpha}) \int_{\Omega} \mathbf{f}^{\Theta}(\mathbf{x}, \Theta) \phi_{free(f)}(\mathbf{x}) dV.$$

Using the above formula, we integrate the rig forces in the rest state of the character and then use the rotations $\mathbf{R}_i(\boldsymbol{\alpha})$ to apply the generalized force to the object in its current state.

6.3 The Rigging Process

In order to rig a character, individual rigs must be configured and combined to form the rigging force $\mathbf{f}^{\Theta}(\mathbf{x}, \Theta)$. Each rig is described as a force $\mathbf{f}_i^{\Theta}(\mathbf{x}, \boldsymbol{\eta}_i)$, where $\boldsymbol{\eta}_i$ is a vector of *rig parameters* that determine the force produced by the i -th rig. The rigging process consists of deciding on a set of rigs and creating a mapping from Θ to $\boldsymbol{\eta}_i$ for each rig. For simplicity, we impose the restriction that the i -th rig depends on a single control parameter $\gamma_i \in \Theta$. The rig force is then formed by combining the force contributions from the rigs:

$$\mathbf{f}^{\Theta}(\mathbf{x}, \Theta) = \sum_{i=1}^{N_{rigs}} \mathbf{f}_i^{\Theta}(\mathbf{x}, \boldsymbol{\tau}_i(\gamma_i)), \quad (6.2)$$

where the function $\boldsymbol{\tau}_i : \gamma_i \mapsto \boldsymbol{\eta}_i$ maps the control parameter associated with rig i to its rig parameters. It is also convenient to express the rig force purely in terms of generalized forces:

$$\mathbf{Q}_F(\Theta) = \mathbf{R}_F(\boldsymbol{\alpha}) \sum_{i=1}^{N_{rigs}} \mathbf{Q}^i(\gamma_i), \quad (6.3)$$

where \mathbf{Q}^i is the generalized force (on the free variables) contributed by the i -th rig, and $\mathbf{R}_F(\boldsymbol{\alpha})$ is a diagonal matrix comprised of 3×3 blocks, with $\mathbf{R}_{F,ff}(\boldsymbol{\alpha}) = \mathbf{R}_{free(f)}(\boldsymbol{\alpha})$.

We represent the function $\boldsymbol{\tau}_i$ as the linear interpolation of a set of sample points $\{(\gamma_{i,k}, \boldsymbol{\eta}_{i,k}) :$

$k = 1, 2, \dots, N_i^T, \gamma_{i,k} < \gamma_{i,k+1}$ that are created by the user during an interactive session:

$$\tau_i(\gamma_i) = \begin{cases} \eta_{i,0} & \text{for } \gamma_i \leq \gamma_{i,0} \\ \eta_{i,k} + (\eta_{i,k+1} - \eta_{i,k}) \left(\frac{\gamma_i - \gamma_{i,k}}{\gamma_{i,k+1} - \gamma_{i,k}} \right) & \text{for } \gamma_{i,k} < \gamma_i \leq \gamma_{i,k+1}, 1 \leq k < N_i^T \\ \eta_{i,N_i^T} & \text{for } \gamma_{i,N_i^T} < \gamma_i. \end{cases}$$

The process by which the user interactively rigs a character can be described algorithmically as follows:

Algorithm 6.1: The Rigging Process

```

i ← 1
while the character is not fully rigged do
    instantiate rig i
    choose a control parameter  $\gamma_i \in \Theta$  for the rig
    k ← 1
    while the user desires more samples do
1      configure the character by setting  $\Theta$ 
2      set the control parameter  $\gamma_i \leftarrow \gamma_{i,k}$ 
3      configure the rig by setting  $\eta_i \leftarrow \eta_{i,k}$ 
        record the sample  $(\gamma_{i,k}, \eta_{i,k})$ 
        k ← k + 1
    end
    i ← i + 1
end

```

6.4 Rigging with Force Field Templates

Our first approach to rigging is to rely on a library of predefined force field templates. Each template defines a parameterized force field in a canonical coordinate frame.

6.4.1 Examples of Force Field Templates

Before describing specific force templates that we have implemented, we will enumerate the design goals that guided our selection:

- A force template should produce an effect that is easy for the user to understand.
- It should have few control parameters so that the user and optimizer (Section 6.4.3) can easily configure it. The parameters must be intuitively meaningful to the user.
- It should be continuous and have continuous first derivatives so that changes in the parameters produce smooth changes in the forces.
- It should be spatially localized and have a simple closed form so that it can be evaluated and integrated efficiently to compute generalized forces.

We provide two examples of force field templates that were designed with the above goals in mind. The first example is the *bump* template, parallel to the y -axis and supported on a ball centered at the origin:

$$\mathbf{f}(\mathbf{x}, A, R) = AB(\|\mathbf{x}\|, R)\hat{\mathbf{y}},$$

where

$$B(r, R) = \begin{cases} \frac{1}{2} + \frac{1}{2} \cos\left(\frac{\pi r}{R}\right) & \text{for } |r| < R \\ 0 & \text{otherwise,} \end{cases}$$

and $\hat{\mathbf{y}}$ is a unit vector in the direction of the y -axis. The parameter A controls the magnitude of the force, and R controls the radius of the support of the force field.

The second example is the *bulge* template, which is inspired by the behavior of a muscle. Roughly speaking, when a muscle contracts in one direction it expands in the other two directions (see, e.g., [70]). The bulge behaves similarly. Contraction along the x -axis is coupled with expansion in the yz -plane. The bulge is defined by the formula

$$\mathbf{f}(\mathbf{x}, A, R) = A \cos(2\theta) B\left(\|\mathbf{x}\| - \frac{R}{2}, \frac{R}{2}\right) \frac{\mathbf{x}}{\|\mathbf{x}\|}, \quad (6.4)$$

where θ is the angle between \mathbf{x} and the positive x -axis. The magnitude of the bulge is again controlled by A and its support is the ball of radius R . The factor $\cos(2\theta)$ creates a smooth transition

between expansion and contraction as the point \mathbf{x} varies from the x -axis to the yz -plane. The function B is scaled and shifted so that the force vanishes at the origin.

In addition to the two aforementioned templates, we have implemented two others whose details are omitted for brevity. The *radial* template is supported on a ball. Its forces point away from the origin, modulated by B . The *torus* template is supported on the interior of a torus. Each circular cross section of the torus contains forces radiating outward from the center of the cross section.

6.4.2 Template Guides

When a template is applied to a character, it must be positioned and oriented. To simplify this task we introduce template guides. Each template guide supports a unique user interface which allows the user to easily position and orient the template. The six degrees of freedom of position and orientation are reparameterized and divided into three categories by the template guide. Some of them are configured automatically as the user drag-and-drops the template. Some are eliminated. The remaining DOFs can be configured using sliders if desired. By reducing the degrees of freedom of position and orientation, we also ease the workload on the optimizer during optimization-driven rigging (Section 6.4.3).

Bone guides position and orient the template relative to a bone of the character. When the user drags a bone guided template with the mouse, the template is positioned at the closest point on the skeleton to the mouse ray, and oriented so that its coordinate system aligns with that of the bone (with the x -axis along the bone). Once positioned on a bone, the template has two positional DOFs: *slide* – the location along the bone, and *orbit* – translation along the local y -axis away from the bone. It has one rotational DOF: *spin* – orientation around the bone. These three DOFs are available to the optimizer (Section 6.4.3).

Skin guides position and orient the template with respect to the skin of the character. When the user drags a skin guided template with the mouse, the template is positioned where the mouse ray intersects the surface, and oriented so that its local y -axis is in the direction of the surface normal (the other axes are set arbitrarily but consistently). Once the template has been placed on the surface of the character, it has one remaining positional DOF: *depth* – translation along the surface normal, and one rotational DOF: *spin* – rotation around the surface normal.

Free guides can be used when the other two template guides are too restrictive. They allow complete positional and rotational freedom. When the user drags a free guided template with the mouse, the template moves perpendicular to the mouse ray. Its orientation can be set arbitrarily using sliders.

When a force field template is combined with a template guide, together they form a rig, producing a force of the form given in Equation (6.2). The rig parameters η are a concatenation of the template parameters (such as A and R) and the template guide parameters (such as *slide* and *spin*). The user is required to configure the rig at line 3 of Algorithm 6.3. This is done by manually setting the template parameters, drag-and-dropping the template according to the chosen template guide, and manually adjusting the remaining position and orientation DOFs if desired.

6.4.3 Optimization-Based Template Configuration

Although force field templates are designed to be simple and intuitive, and we simplify their placement using template guides, it is not always easy for the the user to understand which parameters to change to achieve the desired effect. For this reason we also support an interface wherein the user directly manipulates the surface of the character with the mouse and the system automatically optimizes the parameters of the rig so that the forces cause the surface of the character to conform to the users input.

Consider the static equilibrium equation where the pose is fixed and a single rig is involved:

$$\mathbf{A}\mathbf{q}_F = \mathbf{b} + \mathbf{R}_F\mathbf{Q}_F(\eta), \quad (6.5)$$

where η is the vector of rig parameters (the index i has been omitted since we are assuming the presence of only one rig). Using a drag-and-drop interface, the user indicates target positions $\bar{\mathbf{p}}_1, \bar{\mathbf{p}}_2, \dots, \bar{\mathbf{p}}_{N_P}$ for the points $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_{N_P}$ on the input surface Γ . Our goal is to choose η so that these constraints are met. This is accomplished by minimizing the cost function

$$C(\eta, \mathbf{q}_F) = C_{points}(\mathbf{q}_F) + C_{params}(\eta),$$

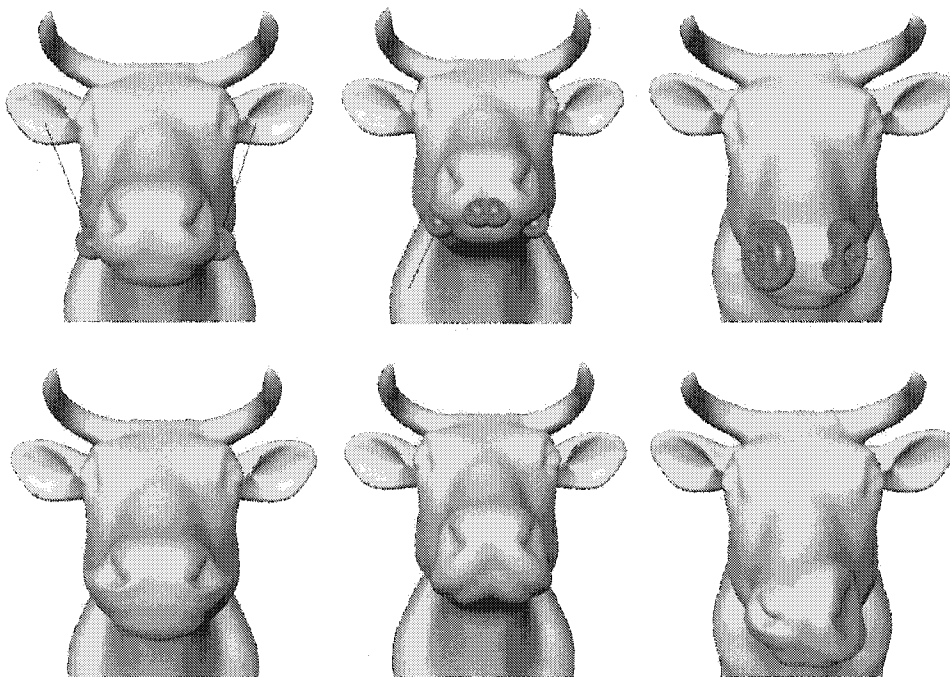


Figure 6.1 This cow head was interactively rigged using force field templates to create simple facial expressions. The templates are visualized as green objects in the top row of figures. The figures in the bottom row show the resulting deformations. On the left, two bump templates were used to create a smile. The red lines indicate the direction of the bump force. In the center, four bump templates effect a frown. On the right, two torus templates allow the dilation and contraction of the nostrils.

where

$$C_{points}(\mathbf{q}_F) = \sum_{j=1}^{N^p} |\mathbf{p}_j - \bar{\mathbf{p}}_j|^2$$

enforces the constraints, and

$$C_{params}(\boldsymbol{\eta}) = \sum_{j=1}^{N^\eta} \xi_j (\eta_{0,j} - \eta_j)^2$$

penalizes deviation of η from its preferred value η_0 . The weight ξ_j determines the magnitude of the penalty associated with the j -th parameter.

We optimize the cost C using the L-BFGS-B algorithm [17], a quasi-Newtonian solver with limited memory usage. Standard optimization methods for smooth cost functions, like L-BFGS-B,

require knowledge of the gradient $\frac{dC}{d\eta}$. Computing it directly using finite differences requires C to be evaluated once for each element of η and leads to a prohibitively slow optimization. Instead, we expand the gradient using the chain rule and regroup as follows:

$$\begin{aligned}\frac{dC}{d\eta} &= \frac{\partial C}{\partial \eta} + \frac{\partial C}{\partial \mathbf{q}_F} \frac{d\mathbf{q}_F}{d\eta} \\ &= \frac{\partial C}{\partial \eta} + \frac{\partial C}{\partial \mathbf{q}_F} \left(\mathbf{A}^{-1} \mathbf{R}_F \frac{d\mathbf{Q}_F}{d\eta} \right) \\ &= \frac{\partial C}{\partial \eta} + \left(\frac{\partial C}{\partial \mathbf{q}_F} \mathbf{A}^{-1} \right) \mathbf{R}_F \frac{d\mathbf{Q}_F}{d\eta}.\end{aligned}$$

Both $\frac{\partial C}{\partial \eta}$ and $\frac{\partial C}{\partial \mathbf{q}_F}$ have simple analytic forms, but $\frac{d\mathbf{Q}_F}{d\eta}$ requires integration over the body (Equation (3.6)) and is difficult to compute analytically. We therefore use finite differences to compute $\frac{d\mathbf{Q}_F}{d\eta}$, which requires one integration over the body per element of η . The important gain in efficiency comes from the regrouping involving \mathbf{A}^{-1} . After regrouping, rather than solving a linear system for each element of η , we need only solve one. This results in much faster optimization. Using this method of computing the gradient we were able to achieve interactive optimizations in the case where a single rig is being configured. Figure 6.1, shows the result of interactively rigging the head of a cow with force field templates.

6.5 Deriving Rigs from Surface Deformations

In addition to the predefined force field templates introduced in the previous section, our framework supports surface deformation rigs, which are constructed directly from surface deformations. This gives the user more control by leveraging existing technologies such as surface scanners and geometric modeling software.

A surface deformation rig uses an arbitrary generalized force to effect the desired shape change. We express the force of a surface deformation rig directly in the generalized form of Equation (6.3), dropping the index i since we will only be discussing a single rig:

$$\mathbf{Q}(\gamma) = \gamma \boldsymbol{\eta}. \quad (6.6)$$

For surface deformation rigs, the rig parameters $\boldsymbol{\eta}$ determine the direction of the generalized force

and the rig control γ determines its magnitude. When using a surface deformation rig, line 3 of Algorithm 6.3 is performed by computing the optimal parameters η so that the target surface is matched.

Given a target surface $\bar{\Gamma}$ represented as a triangle mesh, we want to compute a generalized force such that $\bar{\Gamma} \approx \Gamma$, where Γ is the surface of the simulated body at static equilibrium. Because Γ also depends on the skeletal configuration parameters α , which are unknown for the target surface, we must also compute an optimal skeletal pose. This computation replaces the user interaction at line 1 of Algorithm 6.3. Our approach to optimizing the skeletal pose is based on the work of Allen et al. [3].

6.5.1 Pose Optimization

Our procedure for pose optimization is based on matching a set of user-selected feature points $\{\mathbf{p}_i\}$ on Γ and $\{\bar{\mathbf{p}}_i\}$ on $\bar{\Gamma}$ using the following cost function:

$$C_{features}(\alpha) = \frac{1}{N_{features}} \sum_{k=1}^{N_{features}} |\mathbf{p}_k - \bar{\mathbf{p}}_k|^2,$$

where $N_{features}$ is the number of feature points. Although it requires user interaction, the selection of feature points by the user has the advantage that points can be chosen that are representative of the configuration of the underlying skeleton, such as places where the bone is close to the surface. As before, the L-BFGS-B algorithm is used to solve for the optimal pose parameters α . Figure 6.2 shows the results of optimizing the pose to fit a target surface.

6.5.2 Force Optimization

After finding the optimal skeletal pose associated with $\bar{\Gamma}$, and setting the value of γ^2 (line 2 of Algorithm 6.3), we compute the optimal generalized force (via η) that aligns the deformed surface Γ with the target surface $\bar{\Gamma}$. Our goal is then to find a vector η that minimizes the following cost function:

$$C(\eta) = C_{surf}(\eta) + \omega C_{smooth}(\eta),$$

²Recall that $\gamma \in \Theta = \alpha \cup \beta$. If γ is chosen to be a pose parameter (i.e., $\gamma \in \alpha$), it has already been set during pose optimization. If it is an abstract parameter (i.e., $\gamma \in \beta$), then it is typically set to 1.

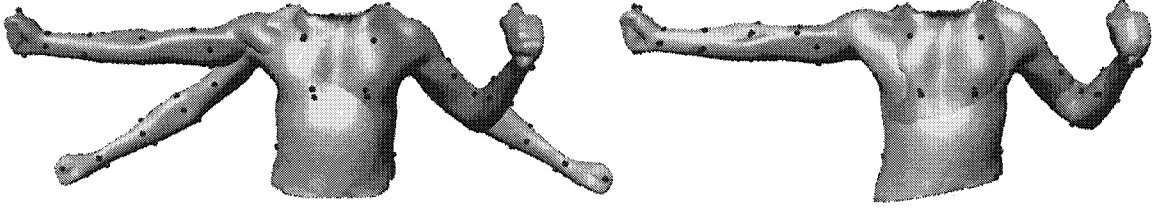


Figure 6.2 In this example, a skeletal pose is computed that deforms the gray surface to fit the blue surface. The gray body is a physical simulation constrained to the motion of the skeleton. The red spheres indicate user-selected feature points that guide the optimization.

where ω is a user-specified smoothing parameter. The first cost term penalizes mismatch between Γ and $\bar{\Gamma}$:

$$C_{surf}(\boldsymbol{\eta}) = \frac{1}{N_{verts}} \sum_{k=1}^{N_{verts}} dist(\mathbf{v}_k, \bar{\Gamma})^2,$$

where N_{verts} is the number of vertices in Γ , $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{N_{verts}}\}$ are the vertices of Γ , and $dist$ measures the minimum distance between a point and a surface. The second cost term encourages smoothness of the displacement field $\mathbf{d}(\mathbf{x})$:

$$C_{smooth}(\boldsymbol{\eta}) = \frac{1}{2} \sum_{k=1}^3 \int_{\Omega} |\nabla d_k|^2 dV,$$

where d_k is the k -th component of $\mathbf{d}(\mathbf{x})$. The cost function C is a function of $\boldsymbol{\eta}$ by virtue of Equation (6.5), which draws a correspondence between \mathbf{q}_F , the state of the character, and $\mathbf{Q}_F(\boldsymbol{\eta})$, the rig forces acting on the character.

If the deformed surface $\bar{\Gamma}$ is created by a connectivity-preserving edit of the original surface, we can use a simpler version of C_{surf} that takes advantage of the correspondence between vertices:

$$C_{surf}(\boldsymbol{\eta}) = \frac{1}{N_{verts}} \sum_{k=1}^{N_{verts}} |\mathbf{v}_k - \bar{\mathbf{v}}_k|^2,$$

where $\{\bar{\mathbf{v}}_1, \bar{\mathbf{v}}_2, \dots, \bar{\mathbf{v}}_{N_{verts}}\}$ are the vertices of $\bar{\Gamma}$.

In some cases, we would like to match surfaces that are not expected to match at every point. For example, we may want to match an isolated arm to an arm belonging to a full human body. In

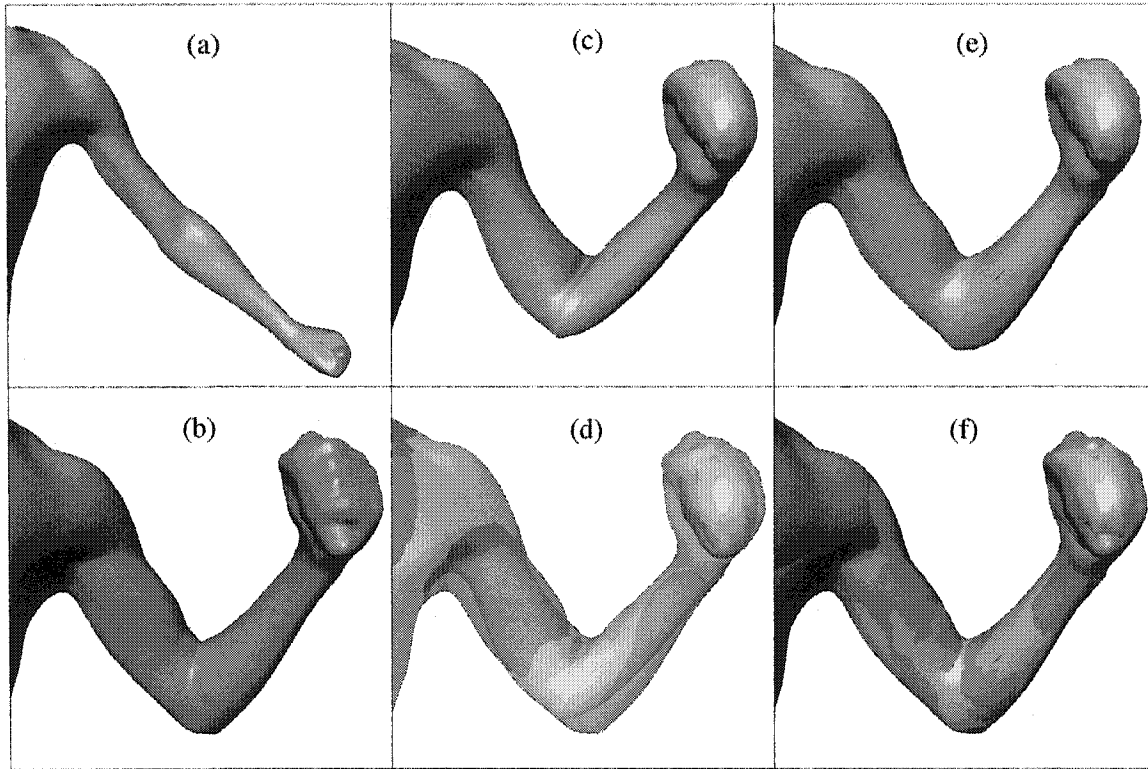


Figure 6.3 A surface deformation rig for a bent arm. (a) A scanned arm in its rest configuration. The arm has been instrumented with a skeleton and control lattice for elastic simulation. (b) The target surface, which was also acquired by scanning. We want to compute the forces that approximately produce the target surface when applied to the simulated arm shown in (a). (c) The static equilibrium shape of the arm in (a) posed to match (b) as well as possible. (d) A comparison between (b) and (c). (e) The bent arm after applying an optimized surface deformation rig that uses forces to produce the target shape. (f) A comparison between (b) and (e).

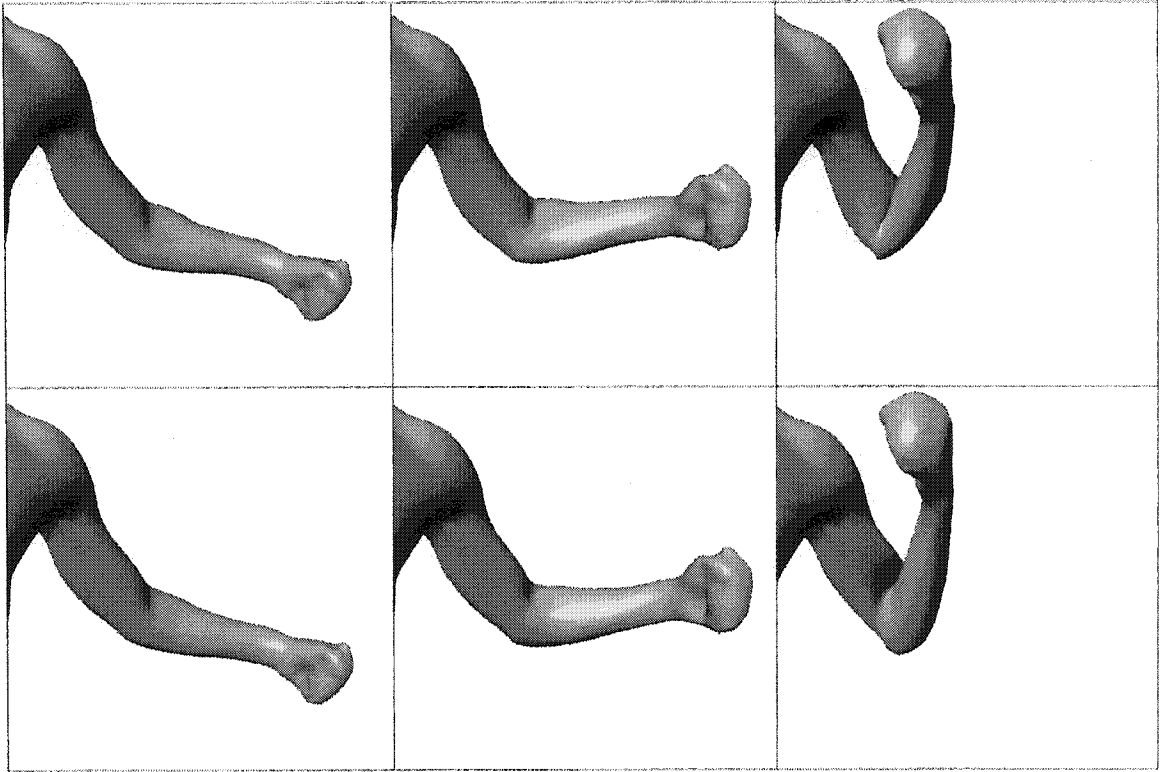


Figure 6.4 This figure illustrates how the rig from Figure 6.3 can produce deformations for other poses by interpolating the forces. The top row shows the simulation without rigging for three different poses. The bottom row shows the same poses simulated with rig forces in effect. Notice that the biceps and the area around the crease of the elbow are much more realistic looking when the rig is used.

cases like this, a lasso tool is used to select the vertices on Γ that are expected to correspond with $\bar{\Gamma}$. In this case we modify the cost function to only include the selected vertices.

Figure 6.3 shows the results of creating a surface deformation rig to match a bent arm. Because the deformation is associated with the arm in a bent pose, we chose the angle of the elbow joint as the control parameter γ associated with the rig. As γ varies from 0 to 1 the force that effects the deformation is gradually introduced. In this manner, intermediate poses of the elbow produce intermediate deformations. Figure 6.4 shows the effect of the rig when the arm is posed differently from the pose of the target data.

6.6 Adaptive Rigging

To make surface deformation rigs match the input surface better, we can increase the resolution of the control lattice. But doing so greatly increases the computational cost of the simulation. We therefore apply the adaptive simulation procedure that was introduced in Section 4.7. Some changes are required to the basic method in order to adapt to the rigging effectively.

The adaptation of Section 4.7 increased the resolution of the simulation where the magnitude of deformation is high. But for surface deformation rigs the goal is to match the input surface, so the adaptation should operate accordingly. We break the problem into two steps as follows:

1. adapt the basis during the optimization of the rig forces, and
2. introduce the appropriate basis functions when the rig is activated.

Step 1 is implemented using the following algorithm:

Algorithm 6.2: Adaptive Rig Optimization

```

for  $l \leftarrow 0$  to  $N_{\text{levels}} - 1$  do
    optimize  $\mathbf{Q}_F$  as in Section 6.5.2
    for  $k \leftarrow 1$  to  $N_{\text{verts}}$  do
1      if  $\text{dist}(\mathbf{v}_k, \bar{\Gamma}) > \epsilon$  then
          activate all level  $l$  basis functions that support  $\mathbf{v}_k$ 
        end
    end
end

```

The quantity ϵ in Line 1 represents a user-defined distance threshold. As in Section 6.5.2, if the surfaces are in correspondence the expression $\text{dist}(\mathbf{v}_k, \bar{\Gamma})$ in Line 1 can be replaced by $|\mathbf{v}_k - \bar{\mathbf{v}}_k|$.

The result of Algorithm 6.6 is that the i -th surface deformation rig has an associated basis \mathcal{B}^i that supports the approximation of the input surface that was used to create the rig. Step 2 is performed by setting $\mathcal{B}_A \leftarrow \mathcal{B}_A \cup \mathcal{B}^i$ whenever the i -th rig is active ($\gamma_i \neq 0$). If the rig is not active, basis functions may be deactivated according to the procedure discussed in Section 4.7.

6.7 Retargetting Surface Deformation Rigs

Deformed surfaces can be difficult to construct, and sometimes a deformation is available for one model but not for another similar model. For these reasons we would like to transfer rigs between characters. In this section we discuss the transfer of surface deformation rigs from one character to another. We accomplish this by creating a mapping between the two characters and then using the map to transfer forces.

6.7.1 Optimal Mapping

Suppose that we have a character whose domain is Ω with surface $\Gamma = \partial\Omega$ and skeleton S , and that the character has been parameterized by a cell complex K in the manner described in Chapter 4, i.e., there is a homeomorphism $\mathbf{r} : K \rightarrow \tilde{\Omega} \supset \Omega$. Our goal is to create a mapping from Ω to a new character $\bar{\Omega}$ with surface $\bar{\Gamma} = \partial\bar{\Omega}$ and skeleton \bar{S} . We approximate such a mapping by repositioning the vertices of the control lattice to form a homeomorphism $\bar{\mathbf{r}} : K \rightarrow \bar{\tilde{\Omega}} \supset \bar{\Omega}$ using the trilinear basis. We can then form a trilinear mapping between the characters as follows:

$$\mathbf{h} : \tilde{\Omega} \rightarrow \bar{\tilde{\Omega}} : \mathbf{x} \mapsto \bar{\mathbf{r}}(\mathbf{r}^{-1}(\mathbf{x})).$$

Our goal is to choose $\bar{\mathbf{r}}$ so that $\mathbf{h}(\Omega) \approx \bar{\Omega}$.

We reposition the vertices of K using an optimization procedure that minimizes a cost function of the form

$$C(\mathbf{h}) = \omega_{skel}C_{skel}(\mathbf{h}) + \omega_{surf}C_{surf}(\mathbf{h}) + C_{distort}(\mathbf{h}). \quad (6.7)$$

The first term encourages the skeletons to match by penalizing the difference $\bar{S} - \mathbf{h}(S)$. The second term (defined in Section 6.5.2) matches the surfaces by penalizing the difference $\bar{\Gamma} - \mathbf{h}(\Gamma)$. The last term penalizes distortion in the mapping. The user defined weights ω_{skel} and ω_{surf} determine how much each term contributes to the total cost. For efficiency, and to avoid local minima, we perform the optimization in four stages.

In the first stage, we attempt to find a good starting guess for our (nonlinear) optimization pro-

cedure. We grossly align the models by minimizing the cost function

$$C_1(\mathbf{h}) = C_{skel}(\mathbf{h}) = \frac{1}{N_b} \sum_{k=1}^{N_b} |\mathbf{b}_k - \bar{\mathbf{b}}_k|^2 \quad (6.8)$$

while allowing the lattice only seven degrees of freedom for translation, rotation and scale. Here \mathbf{b}_k and $\bar{\mathbf{b}}_k$, $1 \leq k \leq N_b$, denote the vertices of S and \bar{S} respectively.

In the second stage, we allow deformation to occur and attempt to match up the skeletons while minimizing distortion of the mapping. We accomplish this by minimizing the cost function

$$C_2(\mathbf{h}) = C_{skel}(\mathbf{h}) + C_{distort}(\mathbf{h}).$$

Our distortion penalty was chosen to be scale, rotation, and translation invariant:

$$C_{distort}(\mathbf{h}) = \frac{1}{N_a} \sum_{k=1}^{N_a} (a_k - \bar{a}_k)^2,$$

where a_k and \bar{a}_k , $1 \leq k \leq N_a$, are the set of face angles in the cell complexes, \bar{K} and K , respectively, as embedded in \mathbb{R}^3 by the mappings \mathbf{r} and $\bar{\mathbf{r}}$. Each face angle measures the angle between two edges of the lattice that belong to the same cell and have a vertex in common.

In the third stage, in addition to the skeleton, we attempt to match the surface. Instead of matching the whole surface, we begin by matching user-selected feature points using the cost function $C_3(\mathbf{h}) = C_2(\mathbf{h}) + \omega_f C_{features}(\mathbf{h})$. This stage, the only one that requires user interaction (to select the feature points), is optional but drastically speeds up the optimization.

In the fourth and final stage we match all of the surface points, instead of only the feature points, using the cost function of Equation (6.7).

Due to the coarseness of the cell complex K , the above procedure does not usually produce a perfect answer. In difficult areas, such as underarm creases, it is necessary to manually adjust a few control points to ensure that the surface is properly embedded in the subdivided solid.

6.7.2 Force Transfer

Once the mapping \mathbf{h} from Ω to $\overline{\Omega}$ has been established, we can use it to transfer a generalized force \mathbf{Q} (or $\boldsymbol{\eta}$, recall Equation (6.6)) from Ω to $\overline{\Omega}$. Transferring the generalized forces directly between different physical systems produced unintuitive results, so instead we convert the generalized forces to displacements, transfer the displacements, and then convert back to generalized forces. In doing so we take advantage of the fact that our pose-dependent linear system draws a correspondence between generalized forces and displacements.

Consider a generalized force \mathbf{Q}_F acting on the character whose domain is Ω . We can convert it to a displacement \mathbf{d} by first applying Equation (6.5) to compute the generalized coordinates \mathbf{q}_F and then applying the basis expansion in Equation (3.4). The laws of vector transformation (see, e.g., [5]) dictate that the transformed displacement $\overline{\mathbf{d}}$ satisfies the equation

$$\overline{\mathbf{d}}(\overline{\mathbf{x}}) = \mathbf{J}(\mathbf{x})\mathbf{d}(\mathbf{x}).$$

where $\mathbf{J}(\mathbf{x}) = \frac{\partial \mathbf{h}}{\partial \mathbf{x}}(\mathbf{x})$. We approximate this transformation by transforming the coefficients:

$$\overline{\mathbf{q}}_i = \mathbf{J}(\mathbf{x})\mathbf{q}_i.$$

Although \mathbf{h} is not differentiable at \mathbf{x}_i , we can estimate \mathbf{J}_i at the vertex \mathbf{x}_i of the lattice by computing the affine map that minimizes the energy function

$$E(\mathbf{J}_i) = \sum_{j=1}^N (\mathbf{J}_i \mathbf{e}_{ij} - \overline{\mathbf{e}}_{ij})^2, \quad (6.9)$$

where \mathbf{e}_{ij} and $\overline{\mathbf{e}}_{ij}$ are edge vectors surrounding vertex i in K and \overline{K} , respectively, as embedded in \mathbb{R}^3 by \mathbf{r} and $\overline{\mathbf{r}}$. If there is an edge between vertex i and vertex j in K , then $\mathbf{e}_{ij} = \mathbf{r}_j - \mathbf{r}_i$. Otherwise, $\mathbf{e}_{ij} = \mathbf{0}$. Minimizing Equation (6.9) reduces to solving a 3×3 linear system. The method of computing an affine map by minimizing Equation (6.9) is commonly referred to as the Procrustes algorithm [12, 40]. Once we have computed $\overline{\mathbf{q}_F}$ as above, Equation (6.5) can be applied again to produce the generalized force $\overline{\mathbf{Q}_F}$ for the target character.

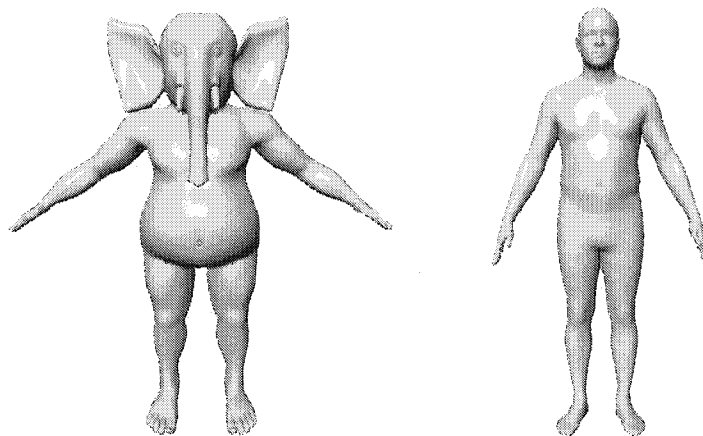


Figure 6.5 Input surfaces for rig transfer: Ganesh (left) and Mean Man (right).

6.8 Results

Some of the results of our rigging system have already been shown in Figures 6.1, 6.3, and 6.4. Two additional input surfaces that were used to test the system appear in Figure 6.5. The model of “Ganesh” on the left was designed using geometric modeling software. The model on the right, “Mean Man”, is the average of a set of scans of men (see [4]). The Ganesh character was instrumented with three rigs which were then transferred to the Mean Man character. One of the rigs was derived from a third model, the arm scan demonstrated in Figures 6.3 and 6.4. Another rig, representing the flexing of the chest, was derived from a surface deformation designed by an artist. A third rig, effecting a breathing motion of the torso, was created using a force field template³.

Figures 6.6 and 6.7 demonstrate the transfer of rigging between characters. In Figure 6.6, the rig derived from the bent arm scan (Figure 6.3) has been transferred to the Ganesh character. It is noteworthy that the thin human arm on which the rig is based and the Ganesh arm behave quite differently when using the unrigged model. The Ganesh arm is short, fat, and cartoon-like, so the deficiencies of the simplified physical model are not as apparent as for the thin human arm as in Figure 6.3 (c) and (d). Despite their significant differences, the rig produces a similar effect on

³This rig based on a force field template was transferred by sampling its generalized force and turning it into a surface deformation rig of the form given in Equation (6.6). It was then transferred using the method described in Section 6.7.

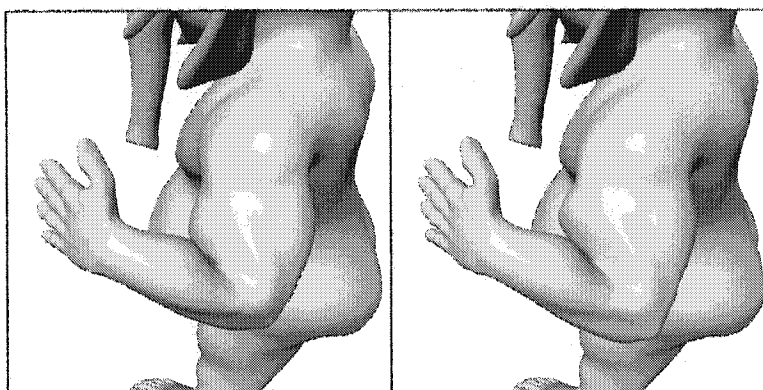


Figure 6.6 The Ganesh arm with rigging transferred from the arm scan in Figure 6.3. On the left is the unrigged arm (in a bent pose) and on the right is the rigged arm. On Ganesh the rig causes similar effects as on the original arm model: the elbow extends, becoming less rounded, and the biceps bulges).

the Ganesh arm as on the scanned arm. The biceps bulges in an appropriate place and the elbow extends, counteracting an unrealistic contraction.

The transfer of a chest flex rig is shown in Figure 6.7. The chest flex rig was created using a target deformation created using geometric modeling software. The target shape is shown in image (f). The deformations that result from using the optimized surface deformation rig are shown in image (b), using a coarse control lattice and image (e), using a finer control lattice (the coarse control lattice subdivided once). Notice that as the control lattice gets finer, the resulting rig forces produce an effect that is closer to the target surface. Images (h) and (i) show the effects of the chest flex rig after having been transferred to the Mean Man character. It produces a plausible deformation despite the fact that the characters are very different.

Figure 6.8 demonstrates frames of an animation in which all three rigs are in effect (although only the chest flex rig is really noticeable in still frames). The animation in row (c) uses 2 levels of basis functions and produces a chest flex that is very much like the target surface. In row (d), a belt constraint has also been applied, demonstrating that our system supports constraints along with rigging. Although we have not done extensive timing studies, the cost of simulation is about equally split between overhead (e.g., bookkeeping, rendering) and solving the sparse linear system at each timestep.

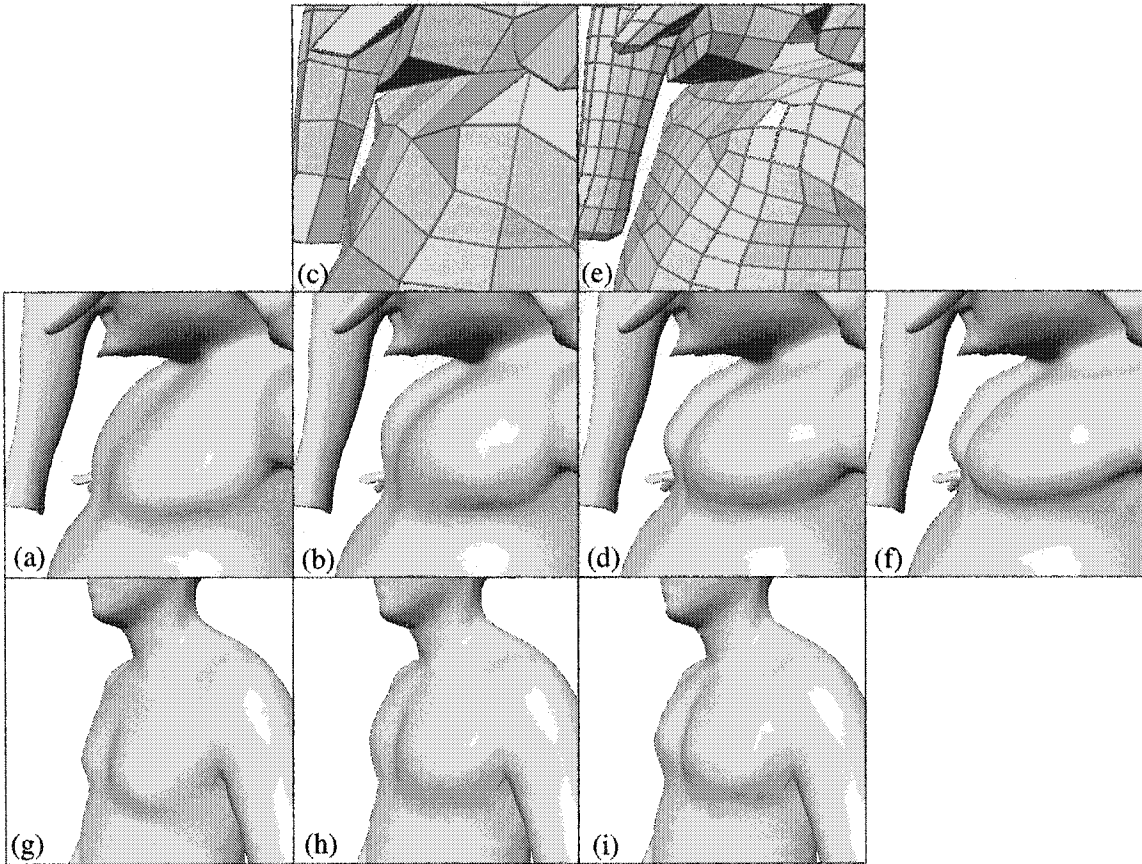


Figure 6.7 A surface deformation rig effecting a chest flex. (a) The undeformed chest of Ganesh. (b) A chest flex using the coarse control lattice shown in (c). (d) The chest flex optimized for the finer control lattice shown in (e). (f) The target deformation used to create the rig. (g) The undeformed Mean Man model to which the chest flex rig was transferred. (h) The chest flex using a control lattice of the same coarse resolution as (c). (i) The chest flex using a control lattice of the resolution of (e).

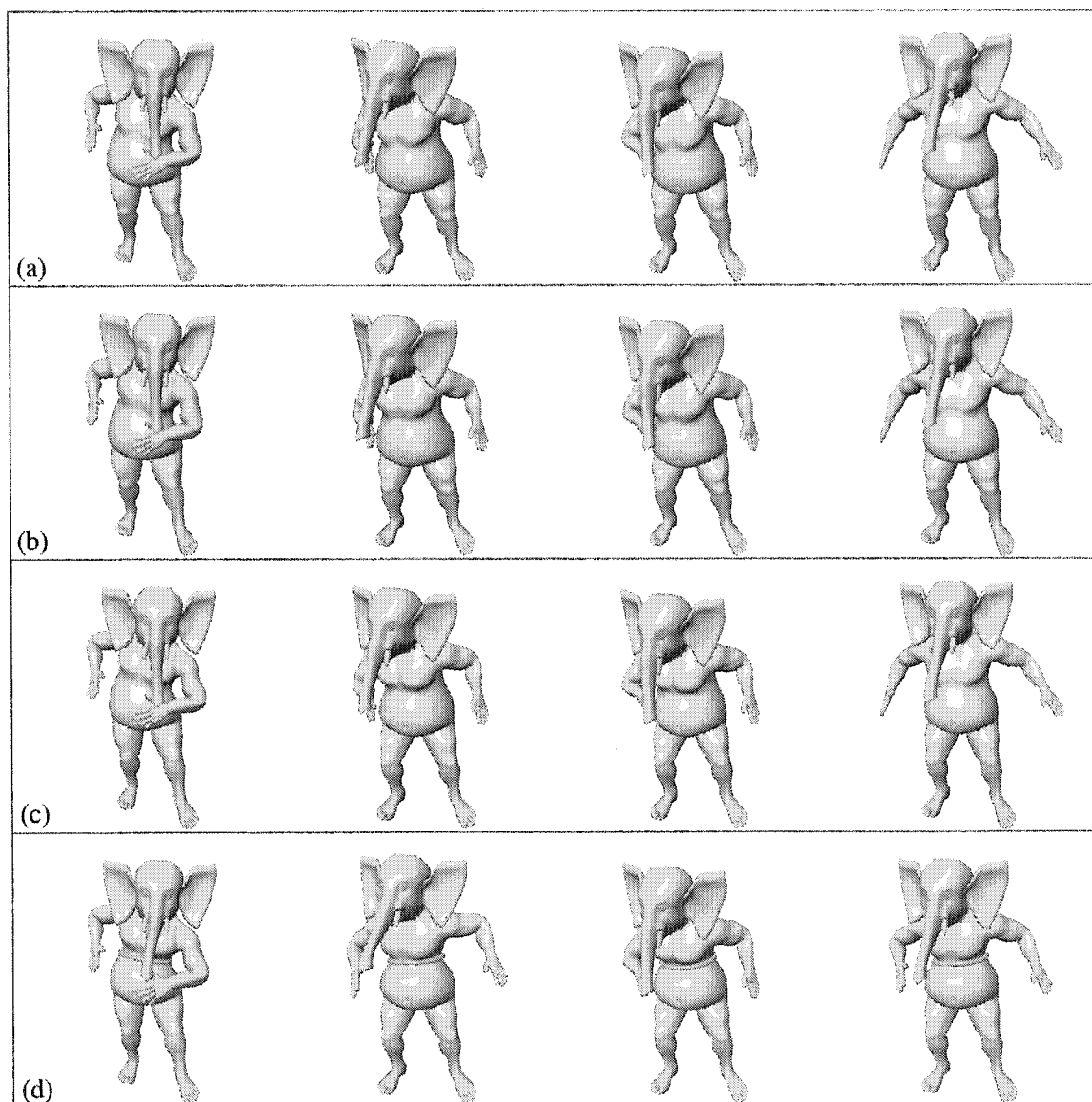


Figure 6.8 Animations demonstrating a chest flex rig. (a) Animation with no force-based rigging. (b) Animation using a chest flex rig optimized for a coarse control lattice. (c) The chest flex rig using a finer control lattice. (d) Animation using the chest flex on a fine control lattice and a belt constraint.

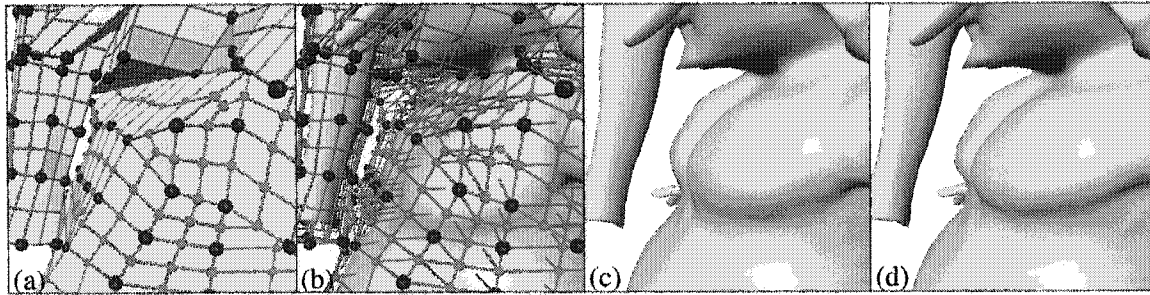


Figure 6.9 Adaptive simulation of the chest flex rig. (a) The surface of the control lattice at rest. Red spheres represent active level 0 basis functions and green spheres represent active level 1 basis functions. Notice that outside the chest region the level 1 basis functions are inactive. The active basis functions are being shown on the rest configuration for clarity, although they were selected during adaptive optimization of the rig forces. (b) The volumetric control lattice. (c) The chest flex resulting from adaptive simulation using the basis functions shown in (a) and (b). (d) The target surface.

The use of adaptive simulation for rigging is demonstrated in Figure 6.9. Notice that the result of the adaptively simulated chest flex using 2 levels of basis functions (Figure 6.9 (c)) is visually identical to the non-adaptive simulation using 2 levels of basis functions (Figure 6.8 (d)). But the computational cost of the adaptive simulation is significantly less. The adaptive simulation required about 0.18 seconds per simulation step, while the non-adaptive simulation required 1.2 seconds. The discrepancy is of course due to the fact that the adaptive simulation requires fewer degrees of freedom (2637 as compared to 9957 for the non-adaptive simulation).

Summary

By augmenting our framework with force-based rigging, the animator gains additional control over the shape of the character. At the same time, the dynamic nature of the simulation is maintained, and external forces such as gravity and constraints can still be used. The forces used for rigging can come from a library of templates or can be derived from surface deformations, and rigs can be transferred from one character to another.

Chapter 7

CONCLUSIONS AND FUTURE WORK

This dissertation has introduced a new character animation framework that unifies dynamic elastic simulation, skeleton-driven deformation, and shape interpolation. In doing so, we present a new way of looking at character animation. The shapes that a character takes on are determined jointly by the dynamical system and the guidance of the animator. Animations of this kind are intrinsically interactive; they can respond to environmental conditions that did not have to have been anticipated by the animator. Physical constraints and external forces can be added without changing the rigging or rig control functions created by the animator.

While we have focused on interactive animations, our framework would also be useful for animations computed off-line. In general, this system can be used to produce animations that automate physical dynamics while providing shape control to the animator. We believe that this work is a step toward a future in which character animation is treated as a physical system under the influence of the animator via abstract controls. Such an approach offers the significant advantage that the animator needs only to control the character at a very high level of abstraction, for example, by directing the character to smile or step to the left. These directions will naturally effect the resulting shape and motion, but the details of motion will be determined automatically by physical laws.

The remainder of this chapter discusses areas of future work that we believe would be fruitful to pursue.

7.1 Improving the Physical Model

One of the techniques central to this work is to estimate dynamic behavior using a simple physical model and correct its shortcomings by introducing additional force fields. The reason that a biceps bulge does not automatically occur in our system is that the physical model is not sufficiently complex; we do not model the internal anatomical structure that would produce a bulge. Instead, force

fields are used to effect the bulge.

This raises the question: what physical model should be used? The pose-dependent linear elasticity model seems to provide a good balance between efficient computation and plausible deformation, but it is a very simple physical model. It seems likely that other points in the trade-off space may also be appealing.

The most obvious deficiency of our physical model is the lack of collision and contact handling. The deformations that occur in real creatures are highly dependent on contact between surfaces, especially where creases are formed near a joint such as along a finger. In our system, rig forces can be used to emulate the effects of collisions to some degree, but when interpolation occurs between forces there is no guarantee that intersection will not occur. Handling contact would slow down the simulation of course, but the resulting deformations would be more realistic.

7.2 *Geometric Mechanical Components*

In this work geometry and dynamics are loosely coupled. Other than the surface of the character, no geometry is created or updated; no internal structure is maintained in order to determine the shape of the character in different configurations. The separation of these two concepts was beneficial because it allowed shape control using the simple abstraction of force fields. It also enabled rigging to be transferred from a character to a similar one simply by mapping force fields.

But separating mechanical structure from its effect on shape has disadvantages too. Consider a nostril, whose dilation behavior is clearly related to its shape. In our system, if the mapping between characters maps a nostril to a nostril then the mechanical behavior of the nostril can be transferred. But what about two characters that cannot be meaningfully mapped to each other, such as a human and a horse? It would still be nice to transfer the mechanical behavior between them.

If the nostril is to be transferred independent of the structure of the rest of the character, it should be put in a library. But what cues in the force field tell us how to deploy it? Shape seems to give the right cues, correlating what the user cares about, shape, with the mechanical behavior.

It seems very promising to approach the problem of rigging physically based characters using a library of components that have both a geometric and mechanical form. Some of the many interesting questions involved are enumerated below.

1. *Extracting Components* – Suppose we are given the surface description of an entire horse and a second surface that is identical except that a nostril is dilated. Can we extract a mechanical component that encapsulates both the mechanical behavior of the nostril as well as its geometric form? Where does the nostril end and the rest of the horse begin? How can we encode the way in which the nostril fits together with its surroundings?
2. *Deploying Components* – If we are given a new surface which has nostrils, how can we fit the nostril in the component library to the new surface? Can the system detect the nostril of the input and automatically unify the component with the input?
3. *Building Characters Directly* – With a library of components, do we even need input geometry? Can components be connected together to form a complete character? How can the relationships between components be represented in the library. How can the shape of a component be edited by the user while maintaining its mechanical properties?

7.3 *Learning and Interpolating Physical Properties*

The rigging system presented in this work provides shape control in the context of physical simulation. Forces are interpolated to effect shape change. A natural extension of the system would be to allow physical properties to be interpolated, in addition to forces.

Consider the bulging biceps that was used as an example in Chapter 6. Our rigging creates the appropriate shape for the bulging muscle but does not impact the underlying dynamics. When a real muscle bulges it becomes more taut, deforming less easily. In the elastic framework this corresponds to a change in the shear modulus G (and possibly Poisson's ratio ν). More realistic results could be achieved if a rig affected G in addition to introducing forces.

The challenge in augmenting our framework so that rigs affect G is to provide the user with the means to design or acquire the physical parameters of the system. For shape control using forces we had the advantage that the static equilibrium state can be used when the user is interactively configuring the rigs. For physical properties that affect the dynamics of the system, new methods of presenting the impact of the user's decisions will be needed.

Acquiring physical properties from actual objects or characters is also a significant challenge. It is well known how to acquire the shape of a character using a surface scanner. The skeletal motion of

a character can also be inferred using motion capture equipment. But acquiring dynamic properties and parameters for animation is an area that has not been thoroughly explored for the purpose of character animation.

BIBLIOGRAPHY

- [1] Marc Alexa. Linear combination of transformations. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2002)*, 21(3):380–387, 2002.
- [2] Alias/Wavefront. *Using Maya: Character Setup*. 2000.
- [3] Brett Allen, Brian Curless, and Zoran Popović. Articulated body deformation from range scan data. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2002)*, 21(3):612–619, 2002.
- [4] Brett Allen, Brian Curless, and Zoran Popović. The space of human body shapes: reconstruction and parameterization from range scans. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2003)*, 22(3):587–594, 2003.
- [5] Rutherford Aris. *Vectors, Tensors, and the Basis Equations of Fluid Mechanics*. Dover Publications, 1962.
- [6] Amaury Aubel and Daniel Thalmann. Realistic deformation of human body shapes. In *Proceedings of Computer Animation and Simulation 2000*, pages 125–135, 2000.
- [7] Fabrice Aubert and Dominique Bechmann. Volume-preserving space deformation. *Computers and Graphics*, 21(5):625–639, 1997.
- [8] Chandrajit Bajaj, Scott Schaefer, Joe Warren, and Guoliang Xu. A subdivision scheme for hexahedral meshes. *The Visual Computer*, 18:409–420, 2002.
- [9] R. E. Bank. *Hierarchical Bases and the Finite Element Method*, volume 5 of *Acta Numerica*, pages 1–43. Cambridge University Press, Cambridge, 1996.

- [10] David Baraff and Andrew Witkin. Dynamic simulation of non-penetrating flexible bodies. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):303–308, July 1992.
- [11] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proceedings of SIGGRAPH 98*, pages 43–54, July 1998.
- [12] P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and machine Intelligence*, 14(2):239–258, February 1992.
- [13] C. Börgers and O. Widlund. On finite element domain imbedding methods. *SIAM J. Num. Anal.*, 27(4):963–978, 1990.
- [14] Morten Bro-Nielsen and Stephane Cotin. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. *Computer Graphics Forum*, 15(3):57–66, August 1996.
- [15] N. Burtnyk and M. Wein. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *Communications of the ACM*, 19(10):564–569, 1976.
- [16] Samuel R. Buss and Jay P. Fillmore. Spherical averages and applications to spherical splines and interpolation. *ACM Transactions on Graphics*, 20(2):95–126, 2001.
- [17] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyu Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(6):1190–1208, 1994.
- [18] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. Interactive skeleton-driven dynamic deformations. *ACM Transaction on Graphics (Proceedings of ACM SIGGRAPH 2002)*, 21(3), 2002.
- [19] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. A multiresolution framework for dynamic deformations. In *Proceedings of the 2002 ACM SIGGRAPH Symposium on Computer Animation*, pages 41–47, 188, 2002.

- [20] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10:350–355, September 1978.
- [21] George Celniker and Dave Gossard. Deformable curve and surface finite-elements for free-form shape design. *Computer Graphics*, 25(4):257–266, July 1991.
- [22] John E. Chadwick, David R. Haumann, and Richard E. Parent. Layered construction for deformable animated characters. *Computer Graphics (Proceedings of SIGGRAPH 89)*, 23(3):243–252, July 1989.
- [23] David T. Chen and David Zeltzer. Pump it up: Computer animation of a biomechanically based model of muscle using the finite element method. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):89–98, July 1992.
- [24] Fehmi Cirak and Michael Ortiz. Fully c^1 -conforming subdivision elements for finite deformation thin-shell analysis. *International Journal for Numerical Methods in Engineering*, 51(7):813–833, July 2001.
- [25] Fehmi Cirak, Michael Ortiz, and Peter Peter Schröder. Subdivision surfaces: a new paradigm for thin-shell finite-element analysis. *International Journal for Numerical Methods in Engineering*, 47(12):2039–72, April 2000.
- [26] Sabine Coquillart. Extended free-form deformation: A sculpturing tool for 3d geometric modeling. *Computer Graphics*, 24(4):187–193, 1990.
- [27] Gilles Debunne, Mathieu Desbrun, Alan Barr, and Marie-Paule Cani. Interactive multiresolution animation of deformable models. *Computer Animation and Simulation '99*, September 1999.
- [28] Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H. Barr. Dynamic real-time deformations using space & time adaptive sampling. *Proceedings of SIGGRAPH 2001*, 2001.

- [29] Tony DeRose, Michael Kass, and Tien Truong. Subdivision surfaces in character animation. *Proceedings of SIGGRAPH 98*, pages 85–94, August 1998.
- [30] Mathieu Desbrun, Peter Schröder, and Al Barr. Interactive animation of structured deformable objects. *Graphics Interface '99*, pages 1–8, June 1999.
- [31] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Dynamic free-form deformations for animation synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):201–214, July–September 1997.
- [32] Y. C. Fung. *A First Course in Continuum Mechanics*. Prentice-Hall, 1977.
- [33] Steven J. Gortler and Michael F. Cohen. Hierarchical and variational geometric modeling with wavelets. *1995 Symposium on Interactive 3D Graphics*, pages 35–42, April 1995.
- [34] Steven J. Gortler, Peter Schröder, Michael F. Cohen, and Pat Hanrahan. Wavelet radiosity. In *Proceedings of SIGGRAPH 93*, pages 221–230, August 1993.
- [35] Jean-Paul Gourret, Nadia Magnenat Thalmann, and Daniel Thalmann. Simulation of object and human skin deformations in a grasping task. *Computer Graphics (Proceedings of SIGGRAPH 89)*, 23(3):21–30, July 1989.
- [36] Seth Green, George Turkiyyah, and Duane Storti. Subdivision-based multilevel methods for large scale engineering simulation of thin shells. In *Proceedings of the Seventh ACM Symposium on Solid Modeling and Applications*, pages 265–272, 2002.
- [37] E. Grinspun, P. Krysl, and P. Schröder. Charms: A simple framework for adaptive simulation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2002)*, 21(3):281–290, 2002.
- [38] Michael Hauts and Olaf Etzmuss. A high performance solver for the animation of deformable objects using advanced numerical methods. *Computer Graphics Forum (Proceedings of Eurographics 2001)*, 20(3), 2001.

- [39] Gentaro Hirota, Renee Maheshwari, and Ming C. Lin. Fast volume-preserving free form deformation using multi-level optimization. In *Proceedings of the Fifth ACM Symposium on Solid Modeling and Applications*, pages 234–245, 1999.
- [40] Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4(4):629–642, April 1987.
- [41] William M. Hsu, John F. Hughes, and Henry Kaufman. Direct manipulation of free-form deformations. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):177–184, July 1992.
- [42] Doug L. James and Dinesh K. Pai. Artdefo - accurate real time deformable objects. *Proceedings of SIGGRAPH 99*, pages 65–72, August 1999.
- [43] R. M. Koch, M. H. Gross, F. R. Carls, D. F. von Büren, G. Fankhauser, and Y. Parish. Simulating facial surgery using finite element methods. *Proceedings of SIGGRAPH 96*, pages 421–428, August 1996.
- [44] Koji Komatsu. Human skin model capable of natural shape variation. *The Visual Computer*, 3(5):265–271, 1988.
- [45] Paul G. Kry, Doug L. James, and Dinesh K. Pai. Eigenskin: Real time large deformation character skinning in hardware. In *Proceedings of the 2002 ACM SIGGRAPH Symposium on Computer Animation*, pages 153–159, 2002.
- [46] J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of SIGGRAPH 2000*, pages 165–172, 2000.
- [47] Xuetao Li, Tong Wing Woon, Tiow Seng Tan, and Zhiyong Huang. Decomposing polygon meshes for interactive applications. In *ACM Symposium on Interactive 3D Graphics*, pages 35–42, 2001.

- [48] Michael Lounsbery, Tony D. DeRose, and Joe Warren. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics*, 16(1):34–73, January 1997.
- [49] A. E. H. Love. *A Treatise on the Mathematical Theory of Elasticity*. Dover Publications, 1944.
- [50] Ron MacCracken and Kenneth I. Joy. Free-form deformations with lattices of arbitrary topology. *Computer Graphics (Proceedings of SIGGRAPH 96)*, pages 181–188, 1996.
- [51] Nadia Magnenat-Thalmann, Richard Laperrière, and Daniel Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Graphics Interface '88*, pages 26–33, 1988.
- [52] G.I. Marchuk, Y.A. Kuznetsov, and A.M. Matsokin. Fictitious domain and domain decomposition methods. *Sov. J. Numer. Anal. Math Modeling*, 1, 1986.
- [53] K. McDonnell and H. Qin. Dynamic sculpting and animation of free-form subdivision solids. In *Proceedings of the Conference on Computer Animation*, pages 126–133. IEEE Press, 2000.
- [54] Kevin T. McDonnell and Hong Qin. FEM-based subdivision solids for dynamic and haptic interaction. In *Proceedings of the Sixth Symposium on Solid Modeling and Application*, pages 312–313. ACM Press, 2001.
- [55] Dimitri Metaxas and Demetri Terzopoulos. Dynamic deformation of solid primitives with constraints. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):309–312, July 1992.
- [56] Eric N. Mortensen and William A. Barrett. Intelligent scissors for image composition. In *Proceedings of SIGGRAPH 95*, pages 191–198, 1995.
- [57] Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler. Stable real-time deformations. In *Proceedings of the 2002 ACM SIGGRAPH Symposium on Computer Animation*, pages 49–54, 189, 2002.

- [58] James F. O'Brien and Jessica K. Hodgins. Graphical modeling and animation of brittle fracture. *Proceedings of SIGGRAPH 99*, pages 137–146, 1999.
- [59] J. T. Oden and J. N. Reddy. *An Introduction to the Mathematical Theory of Finite Elements*. John Wiley and Sons, Ltd., New York, London, Sydney, 1982.
- [60] Frederic I. Parke. Parameterized models for facial animation. *IEEE Computer Graphics and Applications*, 2(9):61–68, November 1982.
- [61] Alex Pentland and John Williams. Good vibrations: modal dynamics for graphics and animation. *Computer Graphics (Proceedings of SIGGRAPH 89)*, 23(3):215–222, July 1989.
- [62] G. Picinbono, H. Delingette, and N. Ayache. Real-time large displacement elasticity for surgery simulation: Non-linear tensor-mass model. In *Proceedings of the Third International Conference on Medical Robotics, Imaging and Computer Assisted Surgery: MICCAI 2000*, pages 643–652, 2000.
- [63] John C. Platt and Alan H. Barr. Constraint methods for flexible models. *Computer Graphics (Proceedings of SIGGRAPH 88)*, 22(4):279–288, August 1988.
- [64] P. M. Prenter. *Splines and Variational Methods*. John Wiley and Sons, 1975.
- [65] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C, 2nd. edition*. Cambridge University Press, 1992.
- [66] Hong Qin, Chhandomay Mandal, and Baba C. Vemuri. Dynamic catmull-clark subdivision surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 4(3):215–229, 1998.
- [67] J. N. Reddy. *An Introduction to the Finite Element Method*. McGraw-Hill, 1993.
- [68] Hans Rijpkema and Brian J. Green. Skinning cats and dogs using muscle deformations. In *SIGGRAPH 2001 Conference Abstracts and Applications*, page 262, 2001.

- [69] S. H. Martin Roth, Markus H. Gross, Silvio Turello, and Friedrich R. Carls. A bernstein-bézier based approach to soft tissue simulation. *Computer Graphics Forum*, 17(3):285–294, 1998.
- [70] Ferdi Scheepers, Richard E. Parent, Wayne E. Carlson, and Stephen F. May. Anatomy-based modeling of the human musculature. In *Proceedings of SIGGRAPH 97*, pages 163–172, 1997.
- [71] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. *Computer Graphics*, 20(4):151–160, August 1986.
- [72] A. Shabana. *Dynamics of Multibody Systems*. Cambridge University Press, 1998.
- [73] Karan Singh and Evangelos Kokkevis. Skinning characters using Surface-Oriented Free-Form deformations. In *Proceedings of the Graphics Interface 2000*, pages 35–42, 2000.
- [74] Peter-Pike J. Sloan, Charles F. Rose III, and Michael F. Cohen. Shape by example. In *Proceedings of 2001 Symposium on Interactive 3D Graphics*, pages 135–143, 2001.
- [75] Jos Stam. Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. *Computer Graphics (Proceedings of SIGGRAPH 98)*, 32:395–404, August 1998.
- [76] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann, San Francisco, CA, 1996.
- [77] Marek Teichmann and Seth Teller. Assisted articulation of closed polygonal models. In *Computer Animation and Simulation '98*, pages 87–101, 1998.
- [78] Demetri Terzopoulos and Kurt Fleischer. Deformable models. *The Visual Computer*, 4(6):306–331, December 1988.
- [79] Demetri Terzopoulos and Kurt Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. *Computer Graphics (Proceedings of SIGGRAPH 88)*, 22(4):269–278, August 1988.

- [80] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. *Computer Graphics (Proceedings of SIGGRAPH 87)*, 21(4):205–214, July 1987.
- [81] Demetri Terzopoulos and Hong Qin. Dynamic nurbs with geometric constraints for interactive sculpting. *ACM Transactions on Graphics*, 13(2):103–136, April 1994.
- [82] Demetri Terzopoulos and Andrew Witkin. Physically based models with rigid and deformable components. *IEEE Computer Graphics and Applications*, 8(6):41–51, November 1988.
- [83] Dennis Turner and Sebastian Marino. Dynamic flesh and muscle simulation: Jurassic park III. In *SIGGRAPH 2001 Conference Abstracts and Applications*, page 173, 2001.
- [84] Joe Warren and Henrik Weimer. *Subdivision Methods for Geometric Design*. Morgan Kaufmann Publishers, 2002.
- [85] Keith Waters. A muscle model for animating three-dimensional facial expression. *Computer Graphics (Proceedings of SIGGRAPH 87)*, 21(4):17–24, July 1987.
- [86] Henrik Weimer and Joe Warren. Subdivision schemes for fluid flow. *Proceedings of SIGGRAPH 99*, pages 111–120, August 1999.
- [87] Jane Wilhelms and Allen Van Gelder. Anatomically based modeling. In *Proceedings of SIGGRAPH 97*, pages 173–180, 1997.
- [88] Andrew Witkin, Michael Gleicher, and William Welch. Interactive dynamics. *1990 Symposium on Interactive 3D Graphics*, 24(2):11–21, March 1990.
- [89] Andrew Witkin and William Welch. Fast animation and control of nonrigid structures. *Computer Graphics (Proceedings of SIGGRAPH 90)*, 24(4):243–252, August 1990.
- [90] David Zeltzer. Task-level graphical simulation: Abstraction, representation, and control. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 3–33, 1991.

Appendix A

DERIVATIVES OF ELASTIC POTENTIAL

The generalized forces associated with elastic deformation correspond to the derivatives of the elastic potential U . Its derivative with respect to the c -th generalized coordinate, for $1 \leq c \leq N$, is

$$\begin{aligned}
 \frac{\partial U}{\partial \mathbf{q}_c} = & \sum_{a=1}^N (2\mathbf{A}_{ca}^1 \mathbf{q}_a + \mathbf{A}_{ac}^2 \mathbf{q}_a + B_{ac} \mathbf{q}_a) \\
 & + \sum_{a=1}^N \sum_{b=1}^N (2\mathbf{q}_b (\mathbf{q}_a \cdot \mathbf{C}_{abc}^1) + (\mathbf{q}_a \cdot \mathbf{q}_b) \mathbf{C}_{cab}^1) \\
 & + \sum_{a=1}^N \sum_{b=1}^N (\mathbf{q}_a (\mathbf{q}_b \cdot \mathbf{C}_{acb}^2) + \mathbf{q}_a (\mathbf{q}_b \cdot \mathbf{C}_{cab}^2) + (\mathbf{q}_a \cdot \mathbf{q}_b) \mathbf{C}_{bac}^2) \\
 & + \sum_{a=1}^N \sum_{b=1}^N \sum_{d=1}^N (\mathbf{q}_d (\mathbf{q}_a \cdot \mathbf{q}_b) D_{abcd}^1 + \mathbf{q}_a (\mathbf{q}_d \cdot \mathbf{q}_b) D_{adcb}^2), \tag{A.1}
 \end{aligned}$$

where

$$\begin{aligned}
 \mathbf{A}_{ab}^1 &= \int_{\Omega} \frac{4G\nu}{1-2\nu} \left(\frac{\partial \phi_a}{\partial \mathbf{x}} \otimes \frac{\partial \phi_b}{\partial \mathbf{x}} \right) dV, \\
 \mathbf{A}_{ab}^2 &= \int_{\Omega} 4G \left(\frac{\partial \phi_a}{\partial \mathbf{x}} \otimes \frac{\partial \phi_b}{\partial \mathbf{x}} \right) dV, \\
 B_{ab} &= \int_{\Omega} 4G \left(\frac{\partial \phi_a}{\partial \mathbf{x}} \cdot \frac{\partial \phi_b}{\partial \mathbf{x}} \right) dV, \\
 \mathbf{C}_{abc}^1 &= \int_{\Omega} \frac{4G\nu}{1-2\nu} \frac{\partial \phi_a}{\partial \mathbf{x}} \left(\frac{\partial \phi_b}{\partial \mathbf{x}} \cdot \frac{\partial \phi_c}{\partial \mathbf{x}} \right) dV, \\
 \mathbf{C}_{abc}^2 &= \int_{\Omega} 4G \frac{\partial \phi_a}{\partial \mathbf{x}} \left(\frac{\partial \phi_b}{\partial \mathbf{x}} \cdot \frac{\partial \phi_c}{\partial \mathbf{x}} \right) dV, \\
 D_{abcd}^1 &= \int_{\Omega} \frac{4G\nu}{1-2\nu} \left(\frac{\partial \phi_a}{\partial \mathbf{x}} \cdot \frac{\partial \phi_b}{\partial \mathbf{x}} \right) \left(\frac{\partial \phi_c}{\partial \mathbf{x}} \cdot \frac{\partial \phi_d}{\partial \mathbf{x}} \right) dV, \text{ and} \\
 D_{abcd}^2 &= \int_{\Omega} 4G \left(\frac{\partial \phi_a}{\partial \mathbf{x}} \cdot \frac{\partial \phi_b}{\partial \mathbf{x}} \right) \left(\frac{\partial \phi_c}{\partial \mathbf{x}} \cdot \frac{\partial \phi_d}{\partial \mathbf{x}} \right) dV. \tag{A.2}
 \end{aligned}$$

Note that \mathbf{A}_{ab}^i , ($i = 1, 2$), is a 3×3 matrix, \mathbf{C}_{abc}^i is a 3-vector, and B_{ab} and D_{abcd}^i are scalar quantities.

The stiffness matrix (the Hessian of U), which is comprised of the second derivatives of U , is then

$$\begin{aligned}
\frac{\partial^2 U}{\partial \mathbf{q}_c \partial \mathbf{q}_d} = & 2\mathbf{A}_{cd}^1 + \mathbf{A}_{dc}^2 + IB_{dc} \\
& + \sum_{a=1}^N (2I(\mathbf{q}_a \cdot \mathbf{C}_{adc}^1) + 2\mathbf{q}_a \otimes \mathbf{C}_{dac}^1 + 2\mathbf{C}_{cda}^1 \otimes \mathbf{q}_a) \\
& + \sum_{a=1}^N (\mathbf{I}(\mathbf{q}_a \cdot \mathbf{C}_{dca}^2) + \mathbf{q}_a \otimes \mathbf{C}_{acd}^2 + \mathbf{I}(\mathbf{q}_a \cdot \mathbf{C}_{cda}^2)) \\
& + \sum_{a=1}^N (\mathbf{q}_a \otimes \mathbf{C}_{cad}^2 + \mathbf{C}_{dac}^2 \otimes \mathbf{q}_a + \mathbf{C}_{adc}^2 \otimes \mathbf{q}_a) \\
& + \sum_{a=1}^N \sum_{b=1}^N (\mathbf{I}(\mathbf{q}_a \cdot \mathbf{q}_b) D_{abcd}^1 + 2(\mathbf{q}_b \otimes \mathbf{q}_a) D_{adcb}^1) \\
& + \sum_{a=1}^N \sum_{b=1}^N (\mathbf{I}(\mathbf{q}_a \cdot \mathbf{q}_b) D_{dacb}^2 + (\mathbf{q}_a \otimes \mathbf{q}_b) D_{abcd}^2 + (\mathbf{q}_a \otimes \mathbf{q}_b) D_{adcb}^2), \quad (\text{A.3})
\end{aligned}$$

where $1 \leq c \leq N$, $1 \leq d \leq N$, and \mathbf{I} is a 3×3 identity matrix.

Appendix B

REVIEW OF TRILINEAR FUNCTIONS

A *trilinear function* on the standard unit cube $C^3 = \{\mathbf{x} = (x, y, z) : 0 \leq x, y, z \leq 1\}$ is a function of the form

$$f(x, y, z) = a_0 + a_1x + a_2y + a_3z + a_4xy + a_5xz + a_6yz + a_7xyz.$$

The function f is determined by its values at the vertices of C^3 : let $\hat{\phi}(s)$ denote the *hat function*

$$\hat{\phi}(s) = \begin{cases} 1 - |s| & \text{for } |s| \leq 1 \\ 0 & \text{for } |s| > 1. \end{cases}$$

and let $\phi_0(x, y, z) = \hat{\phi}(x)\hat{\phi}(y)\hat{\phi}(z)$. Then

$$f(\mathbf{x}) = \sum_{0 \leq i, j, k \leq 1} f_{i, j, k} \phi_0(x - i, y - j, z - k),$$

where $f_{i, j, k} = f(i, j, k)$. It is easy to check that trilinear functions satisfy the following interpolation or *hexahedral subdivision* rules:

- (i) The value of f at the midpoint of an edge of C^3 is the average of its values at the endpoints of the edge.
- (ii) The value of f at the centroid of a face of C^3 is the average of its values at the corners of the face.
- (iii) The value of f at the centroid of C^3 is the average of its values at the eight vertices of C^3 .

If we subdivide the unit cube into 8 sub-cubes in the standard way, we can use these subdivision rules to determine the value of f at the vertices of each sub-cube. Repeatedly subdividing and applying the subdivision rules yields the value of f at each diadic point $(i/2^J, j/2^J, k/2^J)$ of C^3 .

Because the diadic points are dense in C^3 , the subdivision rules completely determine f from its values at the vertices of C^3 . More generally, starting with values of a function at the vertices of the standard cubic tiling of \mathbb{R}^3 and applying the subdivision rules to each cubic cell determines a piecewise trilinear function on \mathbb{R}^3 .

We can generalize this construction to define *piecewise trilinear* functions on any cell complex in which the vertices of each 3-cell of K have valence 3. Starting with the values of f at the vertices of K , we infer its values at the centroid of every edge, face and 3-cell of K . This gives values of f at every vertex of the refined complex K_1 obtained by subdivision (see [50] for details). Because the vertices of each 3-cell of K have valence 3, the subdivided complex K_1 has only hexahedral cells, so after one subdivision, the subdivision process behaves just as for cubes in \mathbb{R}^3 .

There is a corresponding nested sequence of function spaces

$$V_0 \subset V_1 \subset V_2 \subset \dots$$

defined on K . To define V_J , subdivide J -times to obtain the complex K_J and specify values at each vertex of K_J . The subdivision rules then determine a function on all of K . Thus, each function in V_J , for $J = 0, 1, 2, \dots$, is determined by its values at the vertices of K .

Appendix C

ESTIMATING THE ROTATION IN A DISPLACEMENT FIELD

Here we show how to estimate to first order the rotation present in a displacement field. A rotation of θ radians about the axis \mathbf{u} , with $\|\mathbf{u}\| = 1$, can be represented as the matrix

$$\mathbf{R} = \mathbf{u}\mathbf{u}^T + \cos \theta (\mathbf{I} - \mathbf{u}\mathbf{u}^T) + \sin \theta \mathbf{A},$$

where \mathbf{I} is a 3×3 identity matrix and

$$\mathbf{A} = \begin{pmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{pmatrix}.$$

For small θ , where $\sin \theta \approx \theta$ and $\cos \theta \approx 1$, we have

$$\mathbf{R} \approx \tilde{\mathbf{R}} = \mathbf{I} + \theta \mathbf{A}.$$

A displacement field that effects the rotation $\tilde{\mathbf{R}}$ can be written as

$$\mathbf{d}(\mathbf{x}) = \tilde{\mathbf{R}}\mathbf{x} - \mathbf{x} = \theta \begin{pmatrix} u_2x_3 - u_3x_2 \\ u_3x_1 - u_1x_3 \\ u_1x_2 - u_2x_1 \end{pmatrix}.$$

Taking the curl of the above expression results in $\nabla \times \mathbf{d} = 2\theta\mathbf{u}$. Thus, given a displacement field $\mathbf{d}(\mathbf{x})$, the rotation that it contains can be estimated to be $\frac{1}{2}\|\nabla \times \mathbf{d}\|$ radians about the axis $\nabla \times \mathbf{d}$. Note that the irrotational component of \mathbf{d} will not interfere with the above approximation because it is naturally curl free.

VITA

Steve Capell was born in Portland, Oregon and grew up on a farm on Chehalem Mountain surrounded by a hundred cows and seven siblings. He attended Portland Community College for two years and then went on to receive his Bachelor of Science degree from Portland State University. After taking a break to do some traveling, he attended graduate school at Georgia Institute of Technology, but left to pursue a career in video game development at Turner Interactive and later Big Fun Development. He eventually returned to academia; at the University of Washington he earned a Master of Science degree and in 2004 was awarded his Doctor of Philosophy. He now lives in a peaceful bungalow in Kharagpur, India with his wife Rimli Sengupta.