

Interactive Haptics and Display for Viscoelastic Solids

by

Jeffrey Lawrence Schoner

In partial satisfaction of the requirements for the degree of

Master of Computer Science (M.Sc.)
Universität des Saarlandes

Completed under the supervision of

Dr. Jochen Lang
Prof. Dr. Hans-Peter Seidel

at the

Max-Planck-Institut für Informatik
Computer Graphics Group
Stuhlsatzenhausweg 85
66123 Saarbrücken
Germany

begun May 5, 2003
finished November 5, 2003

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Saarbrücken, den 5. November, 2003

Jeffrey Lawrence Schoner

Abstract

This thesis describes a way of modeling viscoelasticity in deformable solids for interactive haptic and display purposes. The model is based on elastostatic deformation characterized by a discrete Green's function matrix, which is extended using a viscoelastic add-on. The underlying idea is to replace the linear, time-independent relationships between the matrix entries, force, and displacement with non-linear, time-dependent relationships. A general framework is given, in which different viscoelastic models can be devised. One such model based on physical measurements is discussed in detail. Finally, the details and results of a system that implements this model are described.

Acknowledgements

I would like to thank Professor Hans-Peter Seidel and the International Max Planck Research School (IMPRS) for funding, computing, and personal resources; Jochen Lang for guiding me through this topic and for providing constructive criticism; Stefan Brabeć for spending days getting the haptics software to mostly work under Linux; Sheila Kasprzyk for teaching me to not dread writing and my family for not being too upset that I completed this so far away from them.

Contents

| | |
|--|------------|
| Eidesstattliche Erklärung | i |
| Abstract | ii |
| Acknowledgements | iii |
| Contents | iv |
| List of Figures | vii |
| List of Tables | ix |
| Notes | x |
| 1 Introduction | 1 |
| 2 Related Work | 3 |
| 2.1 Particle System Methods | 4 |
| 2.2 Physics-based Continuous Methods | 6 |
| 2.2.1 Finite Element Based Methods | 7 |
| 2.2.2 Boundary Element Based Methods | 8 |
| 2.2.3 Modal Methods | 9 |
| 2.3 Other Haptics-Focused Work | 10 |
| 2.4 Acquisition of Deformable Models | 11 |
| 3 Particle Systems | 12 |
| 3.1 Primitives | 12 |
| 3.1.1 Particles | 13 |
| 3.1.2 Unary Forces | 14 |
| 3.1.3 Springs | 14 |
| 3.1.4 Damping Forces | 15 |
| 3.2 Constraints | 16 |

| | | |
|----------|---|-----------|
| 3.2.1 | Simple Local Constraints | 17 |
| 3.2.2 | Simple Global Constraints | 17 |
| 3.3 | Forming Systems | 18 |
| 3.3.1 | Setup | 18 |
| 3.3.2 | Simulating | 20 |
| 4 | Solving Differential Equations | 24 |
| 4.1 | Analytical Solutions | 25 |
| 4.2 | Numerical Solutions | 27 |
| 4.2.1 | Discrete Approximations of Derivatives | 27 |
| 4.2.2 | Explicit (Euler) Methods | 30 |
| 4.2.3 | Implicit (Reverse Euler) Methods | 32 |
| 5 | Discrete Green's Function Matrices | 34 |
| 5.1 | Behavior at Non-Vertex Locations | 36 |
| 5.1.1 | Barycentric Coordinates | 37 |
| 5.2 | The Boundary Element Method | 39 |
| 5.2.1 | Boundary Integrals | 41 |
| 5.3 | Direct Measurement | 42 |
| 5.4 | Mesh Subdivision Considerations | 43 |
| 6 | Haptics Fundamentals | 46 |
| 6.1 | Basics | 47 |
| 6.2 | Higher-Level Constructs | 48 |
| 6.3 | Handling Elastic Deformable Objects | 49 |
| 7 | Modeling Viscoelasticity | 54 |
| 7.1 | Relating Elasticity to a DGFm | 54 |
| 7.2 | Developing a Model | 56 |
| 7.2.1 | Kelvin-Voigt Model | 58 |
| 7.2.2 | Improved Damped Model | 59 |
| 7.2.3 | Other Models | 63 |
| 7.3 | Extension into Three Dimensions | 65 |
| 7.3.1 | Mathematical Description | 65 |
| 7.3.2 | Determining Stiffness Tensors from the DGFm | 67 |
| 7.3.3 | Determining Damping Tensors | 70 |
| 7.3.4 | Using Acquired Measurements | 71 |
| 7.4 | Optimizations | 72 |
| 7.4.1 | Basic Precomputations | 72 |
| 7.4.2 | Taking Advantage of the Zero Force Case | 75 |
| 7.5 | Haptic Simulation | 76 |

| | |
|-------------------------------------|-----------|
| <i>CONTENTS</i> | vi |
| 8 Implementation Details | 79 |
| 8.1 Object Models | 79 |
| 8.2 Supporting Hardware | 80 |
| 8.3 Supporting Software | 80 |
| 8.4 Results | 82 |
| 9 Conclusion and Future Work | 85 |
| Bibliography | 87 |

List of Figures

| | | |
|-----|---|----|
| 3.1 | Spring meshes are often used in cloth simulation. | 19 |
| 3.2 | Visual data structure representation of part of a particle system. Arrows represent pointers. | 20 |
| 4.1 | A spring with spring constant k in parallel with a dashpot with damping constant b | 25 |
| 5.1 | a trivial discrete Green's function matrix for a four point mesh (tetrahedron) | 36 |
| 5.2 | Hat functions around vertices 0, 1, and 2, demonstrate linear blending in one dimension. | 38 |
| 5.3 | Loop subdivision pattern. | 44 |
| 6.1 | Example of surface contact point (SCP), surface collision point and proxy location. The dotted line is the proxy path over time. The solid line is the undeformed surface. The dashed line is the deformed surface. | 50 |
| 7.1 | This graph shows the displacement produced from constant force ($t \in [0, 2.5]$, $f = 1.0$) followed by no force using the single spring and one dashpot model. $k = 1.0$, $b = 0.5$, $h = 0.01$ | 60 |
| 7.2 | This graph shows the force produced from constant velocity ($t \in [0, 1]$, $\dot{u} = 1.0$) followed by 0 velocity using the single spring and one dashpot model. $k = 1.0$, $b = 0.5$, $h = 0.01$ | 60 |
| 7.3 | Schematic of the two spring, one dashpot model. | 61 |
| 7.4 | This graph shows the force produced from constant velocity ($t \in [0, 1]$, $\dot{u} = 1.0$) followed by rest using the two spring and one dashpot model. $k_1 = k_2 = 2.0$, $b = 0.5$, $h = 0.01$ | 64 |

| | | |
|-----|---|----|
| 7.5 | This graph shows the displacement produced from constant force ($t \in [0, 2.5]$, $f = 1.0$) followed by no force using the single spring and one dashpot model. $k_1 = k_2 = 2.0$, $b = 0.5$, $h = 0.01$ | 64 |
| 7.6 | This graph shows the force produced from constant velocity ($t \in [0, 1]$, $\dot{u} = 1.0$) followed by 0 velocity using the double spring and one dashpot model with various values for k_1 and k_2 . $h = 0.01$, $b = 1.0$ | 69 |
| 7.7 | This graph shows the displacement produced from constant force ($t \in [0, 2.5]$, $f = 1.0$) followed by no force using the double spring and one dashpot model with various values for k_1 and k_2 . $h = 0.01$, $b = 1.0$ | 69 |
| 8.1 | “hemisphere” model | 81 |
| 8.2 | “tiger” model | 81 |
| 8.3 | Viscoelastic software in use with haptics. | 83 |

List of Tables




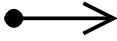
| | | |
|-----|--|----|
| 7.1 | This table shows the relation between two vertices i and j , specified by their corresponding DGFM entry ($\Xi_{i,j}$ or i -th row, j -th column). Entries take the form known constraint value applied to $j \rightarrow$ unknown constraint value produced on i | 55 |
| 8.1 | Overview of properties of the two models used. | 80 |
| 8.2 | PHANTOM® Desktop specifications. Adapted from Sens-Able's™ product documentation. | 82 |
| 8.3 | Timing results of the two models. | 83 |

Notes

Throughout the thesis, the following are used (unless otherwise noted) for mathematical expressions:

- a small dot over a variable represents the first derivative with respect to time, e.g., $\dot{\mathbf{u}} = \frac{d\mathbf{u}}{dt}$
- scalar quantities or functions are printed in the normal font (s)
- vector quantities or functions are lower case in bold font (\mathbf{v})
- matrix quantities or functions are upper case in bold font (\mathbf{M})

For schematic diagrams, the following symbols (with corresponding label meaning in parentheses) are used:

-  is used for a particle (mass)
-  is used for a spring (spring coefficient)
-  is used for a dashpot (damping coefficient)
-  is used to show a measured distance or vector (distance)

FireGL™ is used under license and is a trademark of ATI Technologies, Inc. in the United States and other countries. SensAble™, PHANTOM®, and GHOST® are trademarks or registered trademarks of SensAble Technologies, Inc. in the United States and in other countries. Intel and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. Linux is a registered trademark of Linus Torvalds.

Chapter 1

Introduction

The desire to animate non-rigid, solid objects in the realm of computer graphics flows directly from the contribution of said objects to perceived realism due to their omnipresence in the world. Most often their physical behavior is described by complicated systems of differential equations. The computationally expensive nature of solving such systems to determine behavior has in turn historically led to various simplifications, resulting in an inverse tradeoff between solution time and physical accuracy.

One such simplification germane to this thesis is the introduction of simple quasi-static elastic models. The key assumption made in such models is that once forces are applied or released, the shape of the object changes and comes to rest immediately. Therefore, effects involving oscillation or deformations delayed in time are not present. Furthermore, they are by definition incapable of modeling inelastic behavior such as fracture or permanent deformations arising (in varying degrees) from the use of realistic materials. Nonetheless, because the deformation is determined by a function without regard to time or irreversible changes in the object, simulation can typically be performed more quickly.

With many objects oscillation effects are minimal and the elastic assumption of reversible deformations is perfectly reasonable so long as forces remain below a certain threshold. However, removing time delayed deformation and restoration from models for such “soft” damped objects such as human flesh or plush toys results in clearly less realistic behavior. Adding back this dynamic component to a plain elastic model leads to what is known as a viscoelastic model. Simple models for viscoelastic solids based upon a combination of ideal springs and dashpots have been developed and named for the classical physicists Maxwell, Voigt, and Kelvin [1]. Recent work by Lang, Pai, and Woodham [2] has produced real-world physical measurements consistent with a spring-dashpot viscoelasticity model for overdamped objects such as plush toys and artificial limbs.

This thesis describes a physically based method for simulating such highly damped viscoelastic objects for real-time display and haptic interaction. Because it is designed as a drop-in extension for elastostatic behavior as given by a discrete Green’s function matrix, it provides added realism without requiring complicated tuning or new physical measurements of the model. At its core, the approach combines a discrete Green’s function matrix with aspects of a particle system, resulting in a model with the control of the former, but with the more realistic look and feel arising from the non-linear motion associated with the latter. Furthermore, the slight reduction in computation time is minimal compared to the added realism provided.

Chapter 2

Related Work

The field of three dimensional computer graphics has since its inception concerned itself with modeling and displaying the real world within the computational constraints imposed by hardware. With the constant increase of computing power, every year certain degrees of reality become modelable that were not before. The state of the art in the field of deformable objects has closely followed this trend, starting with the early approaches of Terzopoulos et al. [3] and Pentland and Williams [4]. A survey of many approaches for modeling deformable objects for computer graphics purposes has been written by Gibson and Mirtich [5]. However, since that time other methods from physics and engineering have been adapted for use in interactive and non-interactive computer simulation of deformable bodies, making it somewhat dated.

All methods mentioned in this section attempt in some way to model the physical behavior of deformable solids. However, at the same time they all involve approximations, which defy the true real-world nature of these objects. These approximations include the use of surface meshes or volume tetrahedralizations to describe the object geometry, simplification of more

complicated, but accurate equations for describing behavior, and the use of inexact, numerical methods to solve continuous equations. In many cases, the errors introduced by these are perfectly acceptable, because they are smaller than the error introduced by poor characterization of the problem, are not detectable by sight or touch, or are detectable but not perceived as particularly unnatural or inappropriate.

2.1 Particle System Methods

Methods based on particle systems have been used effectively in the simulation of all kinds of materials for all sorts of purposes within the field of computer graphics. In fact, such methods have been one of the most dominant ones used in computer graphics, because of the broad range of phenomena and objects that can be realistically modeled with them. Moreover, these methods have simple origins in classical physics, which allow them to be understood by anyone with rudimentary knowledge of calculus-based mechanics. Finally, particle systems generally scale well in that large, complex systems with several interacting objects can be built hierarchically from many primitive pieces.

One of the first appearances in the field was for use in facial modeling [6], followed later by use in the animation of explosions along with fire and smoke for use in a feature film [7]. Possibly the most successful usage of such methods has been in the animation of cloth [8, 9, 10, 11, 12]. The most basic idea underlying cloth simulation is to create a planar grid of particles and then connect each particle to its neighbors with simple linear springs. For a given small time step, external forces to be applied to these particles are determined through collision detection and combined

with internal forces arising from the system itself. The combined forces are then used to compute a solution to a set of ordinary differential equations (ODEs), ultimately determining the change in location and velocity of the particles.

Another use of particle systems more related to this thesis is modeling of human organs for surgery simulation. Unlike cloth, which can almost always be modeled as having no thickness, human organs are inherently volumetric. A critical aspect for surgery simulation is the ability to efficiently model incisions as topological changes to the object. One approach described by Groß et al. [13, 14] involves using a particle system consisting of many mass points distributed throughout the volume of a homogeneous organ (such as a liver) that are connected by damped springs. Although they do not provide empirical results from a real software system, they claim the topological changes can be easily implemented by removing certain springs and that the system can be simulated in real time on modern hardware.

One important problem particularly poignant for modeling volumetric objects with particle systems is that the systems cannot easily be controlled. Obtaining desired behavior still often means manually setting and tuning potentially thousands of masses and constants. Furthermore, certain nice global properties are not just hard to control, but actually impossible to model. Real materials have a property known as compressibility, which essentially dictates the amount of volume preserved under deformation. It has been shown that it is impossible to model (and therefore control) this property in a simple spring-mass particle system [15]. Other major potential problems with these methods are accuracy and mathematical stability [16]. Finally, although much recent work has improved the robustness of

solutions [17, 18], accuracy still remains correlated with computation time.

2.2 Physics-based Continuous Methods

Two methods from the world of engineering in particular have found broad based use in the simulation of deformable objects: the finite element method (FEM) and the boundary element method (BEM). In general, both methods can be employed to find a discrete solution to a continuous problem consisting of a set of partial differential equations. Because both have found widespread use in numerous fields and produced copious amounts of related literature, they are each generally recognized as distinct, interdisciplinary fields within applied mathematics and engineering.

The key field from which computer graphics applications for these two methods have come is that of structural engineering. Much of the use of the FEM and BEM in this field is concerned with the behavior of large man-made and geological structures. In essence, the basic underlying theory used for simulating a large bridge can also be used to simulate smaller objects such as a human liver or a toy.

For computer graphics purposes, a number of key advantages for these methods exist. First, unlike particle system based methods, these methods allow control of global properties such as compressibility and general stiffness through parameters such as the Poisson ratio and Young's modulus, which can result in more accurate global behavior. Second, because in general these methods require some pre-computation, the computational burden is shifted away from being done in real-time haptic or display loops, where fast computation time is critical. However, while both are quite powerful and can be applied to and reformulated for various computer graphics

problems, this power comes at the price that both are significantly more complicated to understand than other classes of methods for deformable object simulation.

2.2.1 Finite Element Based Methods

Many introductory textbooks on applied mathematics geared towards engineering students discuss the basics of the FEM (e.g., [19]). Additionally, many more specific texts (e.g., [20]) exist that are solely devoted to the subject.

In order to use the method for the purposes of simulating the behavior of a solid object, the volume of the object must be sectioned into discrete, finite, three dimensional elements (often tetrahedrons). On each of these individual elements stress (σ) and strain (ϵ) tensors (here, 3-by-3 matrices) can be specified or computed as needed. These in turn can be related to the traction¹ ($\mathbf{p} = \sigma \mathbf{n}$, where \mathbf{n} is the surface normal) and deformation ($\mathbf{u} = \epsilon \mathbf{n}$) occurring at the surface of the object.

O'Brien et al. have used this volumetric approach within computer graphics successfully for non-interactive animation of fracture [21, 22]. Knowing internal stresses is critical in this case, as they determine where an object breaks and therefore how its topology changes. However, in many other cases, where the topology of an object does not change, determining these values can lead to superfluous computations. Furthermore, because solving for these internal values in a complex model can be computationally expensive, the method's use in such cases may become confined to offline animation and CAD (computer aided design). Maintaining interactive display frame rates (above 30Hz), let alone interactive haptic feedback rates

¹Traction is a quantity defined as force over area, giving it the same units as pressure.

(above 1000Hz) is for many objects not possible with current hardware.

A number of approaches have employed simplifications of the FEM, resulting in enough dramatic reduction of computational complexity that a higher level of real-time interaction is possible. Bro-Nielsen and Cotin have presented a method that relies on a technique known as condensation or Guyan reduction [23], where a volumetric FEM model is simplified to deal only with the boundary of an object. This results in reducing the complexity of a volumetric problem to that of a surface problem, while still maintaining volumetric behavior. Additionally, they have effectively integrated their method into a surgery simulation system, providing empirical proof of display and simulation at interactive rates. Another such simplified approach that has been presented by Zhuang [24] uses a quadratic stress-strain relation, specialized collision constraints, and non-uniform meshing to reduce the computational complexity of calculating solutions. This approach was also employed in a real-time haptic simulator [25].

Debunne et al. have developed a FEM based method, which provides animation of viscoelastic solids at an adaptive but guaranteed frame rate [26]. This is made possible through the novel use of multiresolution methods.

2.2.2 Boundary Element Based Methods

The definitive texts on theory and engineering applications for the BEM are those of Brebbia et al. [27, 28]. The method's defining advantage is that because problems only need to be discretized on their boundary instead of throughout their volume, their entire solution is characterized just in terms of the object boundary. Using these methods eliminates dealing with internal tetrahedrons as would be required with the FEM or the need to

scatter mass points throughout an object's volume with regard to a particle system. As was touched upon in the section on FEM based approaches, this is particularly attractive for purposes involving display and haptic interaction, as boundary representations such as meshes are already widely used in such applications. However, this also has the side effect that, because it is no longer possible to characterize the internal volume of an object, the method is not fully capable at modeling heterogeneous materials. Naturally, this approach does not lend itself well to certain problems such as simulation of fracture where internal stresses and strains are needed. Finally, in certain cases finding a solution may be highly complex or not even possible.

The work of James and Pai forms the most recent implementation of BEM methods for computer graphics. They have devised an efficient method for linear elasticity and implemented it in a real interactive haptics system called ArtDefo [29]. Unlike many others, their method handles changing boundary conditions efficiently through the use of capacitance matrix algorithms. Furthermore, they have introduced the pressure mask, which is a helpful abstraction of haptic contact [30].

2.2.3 Modal Methods

Modal methods take a somewhat different approach to dealing with deformable solids. The underlying idea is to decompose the general behavior of an object into a finite number of vibrational modes in some way. Much work unrelated to computer graphics has been done in the field of sound and vibration (e.g., [31]). In computer graphics, it was first laid out by Pentland and Williams [4]. In their implementation, they precompute these modes by diagonalizing the mass, damping, and stiffness matrices

derived using the FEM. Modes whose frequencies are too high to be seen are discarded. They claim that using their approach is up to two orders of magnitude faster than using the plain FEM alone.

Although first presented in computer graphics in 1989, a recent renaissance in these methods has occurred. Hauser, Chen, and O'Brien have extended it to be more robust for interactive manipulation [32]. O'Brien, Cook, and Essl have found uses for it in the related field of computational sound synthesis [33]. Instead of discarding high frequency modes, any modes with frequencies outside the range of human hearing (20Hz to 20,000Hz) are discarded. James and Pai have also devised a way of rendering the deformations from modes almost entirely within graphics hardware [34], thereby adding dynamic deformation to models at almost no cost to the main processor. Finally, Basdogan has developed uses for it in the field of interactive surgery simulation [35].

2.3 Other Haptics-Focused Work

Quite a body of work exists within the field of haptics. Although much of the literature is not specifically concerned with deformable objects or interactive computer graphics, the methods discussed therein are still applicable in a general sense. Due to demanding update rates for haptics (1000Hz is generally agreed to be sufficient) and the complexity of the algorithms needed (such as collision detection), interactive haptics has been limited by scene complexity.

Zilles and Salisbury did some early work without graphic interaction [36] that still addresses basic issues with haptics. One piece of work, which has combined haptics and graphics is an interactive system by Rusipini,

Kolarov, and Khatib [37]. Both of these methods are concerned with the already difficult problem of haptic display of rigid objects. Although not the first to do so, very recent work by Barbagli, Salisbury, and Prattichizzo specifically tackles the inclusion of deformable objects in a haptics system [38].

Although not strictly from the field of computer graphics, a good deal of practical work with haptics and deformable models has been done in the construction of surgery simulators (e.g., [39, 40, 41]).

2.4 Acquisition of Deformable Models

All above mentioned methods are concerned with haptic and visual applications, once a model has been constructed with some sort of parameters. Determining such global parameters as Poisson ratio, Young's modulus or other characterizations of these such as Lamé constants can be done by human trial and error (adjust until it looks or feels "right") or by real physical measurements of a material sample. In particular, work concerning the latter approach for determining these kinds of properties of human tissue for surgery simulation has been done [42, 43, 44, 45, 46].

Alternately, work has been done concerning global determination of linear deformation from physical measurements of real world objects [2, 47]. Their approach involves the use of a robotic probe to produce deformations on real objects at discrete surface points. The robotic probe then measures the force exerted back at those points, while cameras are used to measure deformation occurring at other points on the surface.

Chapter 3

Particle Systems

Physically based methods attempt to determine the behavior of complex objects by using principles from physics to perform a mathematical simulation of the objects. Methods based on particle systems combine simple, idealized components such as particle masses and springs together to model a larger, more complex system. A concise and practical introduction to the subject is the evolving SIGGRAPH course entitled “Physically Based Modeling”. Notes are available for many of its offerings (the latest as of this writing in 2003 [16]). Here, an introduction primarily to aspects relevant to the ultimate thrust of this thesis will be provided.

3.1 Primitives

Particle systems are built from the ground up out of simple primitives. Since virtually any kind of physical relationship can be formed into a primitive, an almost unlimited number of them can be conceived. Only the most basic ones will be presented here, so that the reader may get a taste for what typical primitives are like and how they need to be characterized for

use within a larger system. With this understanding, it should be relatively easy to see how additional primitives can be added.

3.1.1 Particles

The simplest component used in physically based modeling is the idealized particle mass (hereafter referred to as a particle). Every particle has a fixed scalar mass (m), a variable vector location in space (\mathbf{u}) and moves according to a vector velocity (\mathbf{v} or $\dot{\mathbf{u}}$)¹. One of the most basic formulas from physics,

$$\mathbf{f} = m\ddot{\mathbf{u}}$$

governs the motion of a given particle, where the vector quantity \mathbf{f} is the force being applied to it and the vector quantity $\ddot{\mathbf{u}}$ is its acceleration. Since we are concerned only with computer simulation of realistic behavior, the units of the above quantities can be arbitrary so long as they are consistent. All of the above mentioned vector quantities typically are three dimensional (consistent with 3D computer graphics), although higher and lower dimensions are possible for certain uses.

While particles are the most basic elements for physical simulation, conceptually they can do nothing interesting on their own. The most that can be done for now is to create several particles, give them initial velocities and watch them travel statically through an infinite space. Particles only react, when forces are applied. Furthermore, for simplicity, it is typically assumed (and will be here) that particles do not collide with one another and therefore that no collision forces are present.

¹For simulation of solids at the macroscopic level, kinetic energy can be treated solely expressed as velocity. Certain more complicated models (such as those sometimes used for fluid dynamics) may divide the kinetic energy into velocity and temperature. This is however, beyond the scope of this thesis and will not be further discussed.

3.1.2 Unary Forces

The simplest forces that can be added to a system are known as unary forces. The most familiar (and commonly used) one is gravity. A unary force simply is one that is exerted with equal direction and intensity on all particles (or a subset thereof). This gives the trivial equation,

$$\mathbf{f} = \mathbf{g}$$

where \mathbf{f} is the force exerted on the particle and \mathbf{g} is a fixed, vector value. This addition still does not make for a particularly interesting system on its own. Particles are now not just capable of moving constantly in an infinite space, but also of accelerating into one direction infinitely.

3.1.3 Springs

Adding binary forces or idealized springs (hereafter referred to simply as springs) is one of the simplest ways to produce interesting forces upon particles. A spring relates the distance between two particles (let us call them i and j) with a force exerted upon the same two particles (in opposite directions). The equation,

$$\mathbf{f} = -k(|\mathbf{l}| - l_r) \tag{3.1}$$

governs this relation, where \mathbf{f} is the force vector on particle i , $-\mathbf{f}$ is the force vector on j , k is a non-negative scalar spring constant (units in terms of force over distance) representing stiffness (larger is more stiff), \mathbf{l} is the current vector distance between the two particles ($i - j$), and l_r is the so-called “rest length” of the spring (the distance between the two that will

cause the spring to produce no force on the particles).

3.1.4 Damping Forces

Galileo's concept of inertia, expressed as Newton's first law of motion tells us that a body in motion will only cease to be in motion due to external forces acting on said body. In every day life, this is not obvious, because there are so many forces acting on everything all the time. The introduction of damping into a particle system is a simple (although idealized) way of partially mimicking the observation that everyday bodies eventually come to rest. Forces due to damping are proportional to velocity ($\dot{\mathbf{u}}$ in the case of a particle), but act in the opposite direction, giving the equation,

$$\mathbf{f} = -b\dot{\mathbf{u}}$$

where b is a non-negative scalar damping constant (units are force over velocity). A larger constant will cause the particle to come to rest faster.

This mathematical relation is also that of an idealized dashpot. Such a component has an arbitrary, dynamic length, where the force exerted is only in proportion to the derivative of this length with respect to time. Hence, the faster it is moved, the more force produced, while infinitely slow motion will produce no force. One non-ideal real world equivalent of an idealized dashpot is a valve filled with a viscous fluid that has a small opening that allows the fluid to leave. Depressing the valve slowly causes the volume inside to decrease slowly, and therefore an equal amount of fluid to flow out of the small opening at low pressure. However, to depress the valve quickly, much more force is needed to balance the pressure necessary for the liquid to flow out of the same small opening at the increased

rate.

Damped Springs

We know from our own personal experience with non-ideal elastic objects that after a certain amount of time they also eventually come to rest, due to these everyday forces as well. Damping in a spring is proportional to its velocity of compression or expansion ($\dot{\mathbf{u}}_i - \dot{\mathbf{u}}_j = \mathbf{v}_i - \mathbf{v}_j = \dot{\mathbf{l}}$). Its direction is opposite to this velocity, but acts along the axis of the spring². This leads to the more complicated relation for a damped spring,

$$\mathbf{f} = - \left[k(|\mathbf{l}| - l_r) + b \frac{\dot{\mathbf{l}} \cdot \mathbf{l}}{|\mathbf{l}|} \right] \frac{\mathbf{l}}{|\mathbf{l}|} \quad (3.2)$$

where once again \mathbf{f} is the force on i while $-\mathbf{f}$ is the force on j .

3.2 Constraints

Damped systems composed of particles, unary forces, and binary forces are capable of adding considerable realism to a particle system. In general, they accurately model simple elasticity. However, their addition only allows an interesting conglomeration of particles that float together through infinite space.

Adding constraints to the above mentioned primitives is necessary for typical simulation scenarios. Almost anything that we would like to visually display or haptically render must take place in a finite volume. Constraints make it possible for the whole system or a part of it to be limited in its cross-

²This gives rise to the dot product and final multiplication terms in Equation 3.2 involving the spring velocity ($\dot{\mathbf{l}}$) and the spring orientation ($\frac{\mathbf{l}}{|\mathbf{l}|}$).

ing of narrow or wide boundaries. The simplest local and global constraints will be discussed here, although they are of course not exhaustive.

3.2.1 Simple Local Constraints

The simplest way to constraint a particle is to simply fix its location or “nail it down”. Any forces applied to a particle with such a constraint can simply be ignored. This eliminates the particle from ever having any acceleration, which in turn leads to the desired effect of no velocity or motion. Any particle can simply be flagged or unflagged as fixed at any point in time. This strategy is simple and robust for interactive or offline simulation.

3.2.2 Simple Global Constraints

The simplest global constraint is the application of impermeable barriers or walls. However, the implementation of these constraints is not as simple as for the above local constraints. First, collisions between the barriers and all particles must be detected, which can be computationally expensive with many particles and arbitrary wall shapes. If planar boundaries (especially those parallel to the axes of the coordinate system) are used and if particles are only allowed to be one side of the boundary, the situation becomes much less complicated.

The second issue is the treatment of what happens when a collision is detected. In a real-world physical collision, both objects deform as the kinetic energy of the collision is absorbed. What happens next is highly dependent on the types of materials that are being dealt with. However, in all cases this kinetic energy must be converted into some other kind of kinetic energy (sound, heat, light) or potential energy. In the case of something

composed of material like clay, the deformation may simply remain. The energy has simply resulted in the rearrangement of the object's atoms. In the case of perfectly elastic rigid objects, the deformed state is only temporary. In these kinds of objects, as the original state is restored, new forces (usually opposite the former direction of motion) are generated, which set colliding objects back into motion. For everything in between, some combination of the two occurs. A ball of silly putty will be partially deformed as well as bounce away from the wall. An egg will fracture, resulting in several separate parts with new kinetic and potential energies. Even a highly elastic "superball" produces sound upon collision, which is simply another form of energy being dissipated as sonic vibrations.

For a simple particle system, it is usually not worth the effort and computational cost to simulate collision at the microscopic level. One idealization that can be made is to simply assume the particle is a rigid point mass and the wall is perfectly elastic. Upon collision, a particle retains the same velocity, but in a new direction dependent on the collision angle.

3.3 Forming Systems

In order to make use of these primitives, they must be connected together into an actual particle system.

3.3.1 Setup

In practice, the exact the layout of particles, springs, and dashpots as well as external forces such as gravity and damping is highly dependent on the the object or phenomenon being simulated. Furthermore, certain primitives may be useful for particular applications, but completely useless in

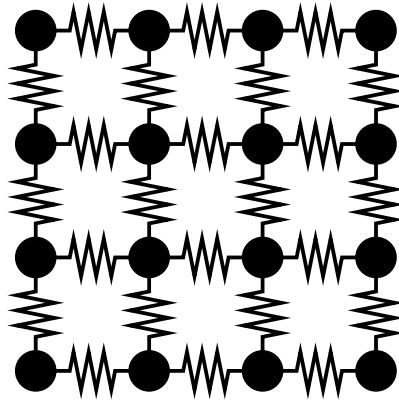


Figure 3.1: Spring meshes are often used in cloth simulation.

others. The nature of particle systems allows for the design of a framework into which virtually any kind of possible force exerting primitive can be devised.

For cloth simulation, a rectangular, planar grid of particles with equal masses is first created (see Figure 3.1). Next, each particle is connected with up to four of its neighboring particles by a spring. Depending on how stretchy a fabric is desired, a spring constant value is chosen that is typically the same for all springs. For this particular application, a special primitive known as an angle spring has been found to be useful in keeping cloth from folding over on itself so easily [17]. Angle springs are usually attached to all pairs of springs, which are connected to a common particle and lie adjacent and perpendicular to each other in the mesh. During simulation, they exert angular forces upon the springs, which in turn exert forces upon the particles.

When simulating volumetric solids a slightly different approach is taken [13, 14]. Typically, a series of particles are distributed throughout the volume of the solid. The masses of these particles are varied in order to accurately model the density of the solid in different locations. These particles

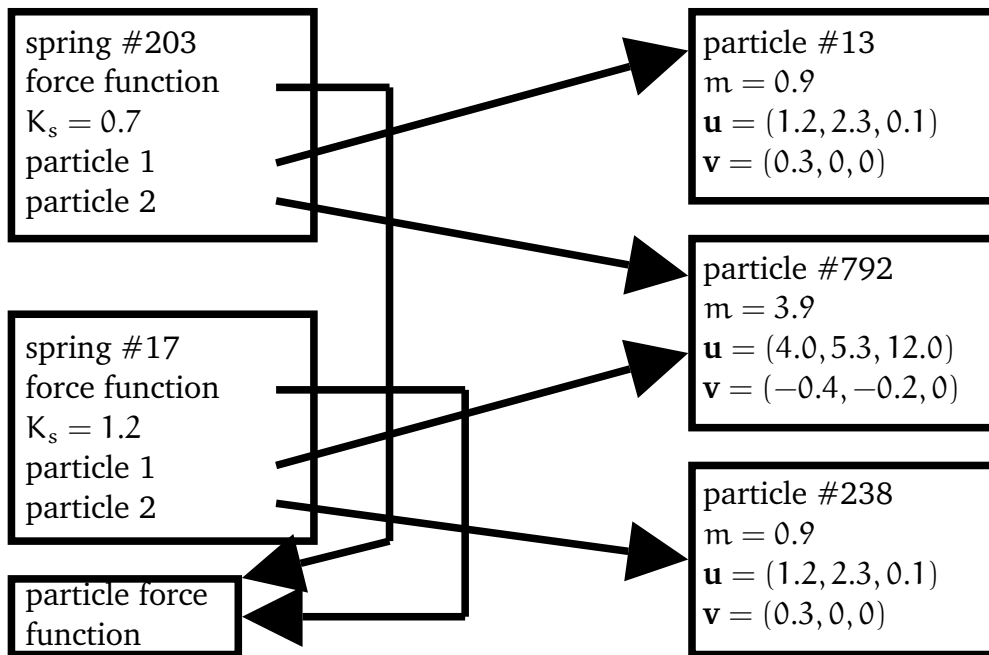


Figure 3.2: Visual data structure representation of part of a particle system. Arrows represent pointers.

are then connected with neighboring particles by springs that are tuned to produce the correct stiffness of the solid. More information on both of these examples can be found in references discussed in Section 2.1.

3.3.2 Simulating

Software for simulating a particle system may contain many parts specific to the type problem being solved. Here, the discussion will be limited to the general core of any such software: the simulation loop and necessary supporting data structures.

Using data structures that enforce modularity results in a more robust particle system simulator by allowing the addition of new kinds of force producing elements (springs, dashpots, walls, etc.) without signifi-

cant rewriting of code. One effective way of achieving this is by producing a unified data structure for force producing elements. Its necessary contents are

- parameters for determining force (e.g., spring coefficient, damping coefficient, rest length),
- pointers to other force producing elements or particles, whose dynamic state will be used at run time to determine how much force is applied and to where,
- a function pointer, which is called with this element itself and possibly the time as parameters and which has the effect of computing and applying the appropriate forces to the appropriate particles.

Additionally, a separate structure is needed, which contains the mass (m), location (\mathbf{u}), and velocity (\mathbf{v}) of a particle. Each particle instance and each force producing element instance are placed into an array or list of particles or force producing elements, respectively. See Figure 3.2 for a visual example.

Before starting the loop, only the starting time, t , must be set³. The idea behind the loop is to determine the position of all particles as well as other necessary state information at discrete locations in time. The algorithm for the loop follows.

1. Add to or remove from the system any particles or force producing elements as necessary.
2. Calculate the forces exerted on all particles.

³A common value for the starting time is 0, but any arbitrary value will work. Furthermore, keeping track of the current time is not necessary at all in cases where the forces exerted are time independent. In such cases, step 5 of the algorithm can be left out.

- (a) Iterate through all of the particles, setting the force accumulation fields of each to zero.
 - (b) Iterate through all force producing elements, computing the forces each exerts on any particles, given the current time and state of the system. Add these forces to the corresponding particle force accumulation fields.
3. From these particle forces, compute the acceleration vector for each particle i by using $\mathbf{a}_i = \frac{\mathbf{f}_i}{m_i}$.
 4. For each particle, update its velocity (\mathbf{v}_i) by using its acceleration (\mathbf{a}_i) and its location (\mathbf{u}_i) by using its velocity (\mathbf{v}_i). This essentially involves solving the ordinary differential equations (ODEs),

$$\begin{aligned}\frac{d\mathbf{u}_i}{dt}(t) &= \mathbf{v}_i(t) \\ \frac{d\mathbf{v}_i}{dt}(t) &= \mathbf{a}_i(t)\end{aligned}$$

for $\mathbf{u}_i(t)$ and $\mathbf{v}_i(t)$, respectively.

Typically, these equations must be solved numerically. Because the functions are arbitrary, finding analytical solutions proves to not be feasible. For these specific equations arising in acceleration driven particle systems, the SIGGRAPH course notes discuss many practical methods in detail. Chapter 4 of this thesis discusses more general methods for solving ODEs, which are applicable to these equations as well.

5. Advance the time t by the desired amount. This can be increased by a constant amount every time. More advanced methods with adaptive time steps are also possible (see SIGGRAPH course notes).

6. Repeat loop until simulation is finished.

Particle systems (as well as software supporting them if designed correctly) truly provide an extensible means of simulating physical objects (and phenomena) through hierarchical layering of simple components. Just as more advanced primitives can be added for more robust behavior, so can more advanced methods for numerically solving ODEs be added for better accuracy and stability. In the next chapter, we will examine some basic ways of solving ODEs found in particle system components.

Chapter 4

Solving Differential Equations

Any solution to a differential equation (such as those presented in Chapter 3) must be discretized before it can be used in a digital computer. Two main possibilities exist in approaching this issue: solve the differential equation analytically, then discretize the solution or discretize the differential equation, then solve it numerically.

Here only the discretization of these problems with respect to time will be discussed, since the equations from particle systems deal only with derivatives with respect to time. The scalar h will be used to denote the current time step size in the general case. Sometimes, it is useful to adjust the time step to obtain a solution more efficiently or as a result of real time simulation (the time between samples may not be constant). In these cases, h can be viewed as a function of time being evaluated at the current point in the simulation.

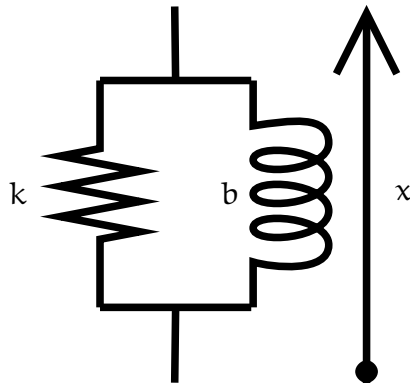


Figure 4.1: A spring with spring constant k in parallel with a dashpot with damping constant b .

4.1 Analytical Solutions

It is non-trivial to solve most differential equations analytically. Most importantly, because many do not have closed form solutions, analytic approaches cannot be relied on for solving general problems. Fortunately, for some specific simple particle relations, such analytical solutions can be easily derived. Let us examine two particles connected by a spring with spring constant k in parallel with a dashpot with damping constant b (see Figure 4.1) in one dimension. Further, let us add the condition that the particles are at some distance x from each other and that one of the points is fixed at 0. The position of the free particle (x) is given by,

$$f = kx + b\dot{x}.$$

where f is the force being applied. If we consider only the case where $f = 0$, the equation can be rewritten as,

$$kx(t) = -b\dot{x}(t).$$

Next, we assume that $x(t)$ has the form De^{-Ct} and therefore,

$$\dot{x}(t) = -DCe^{-Ct}.$$

Substituting these back into the above equation gives,

$$\begin{aligned} Dke^{-Ct} &= bDCe^{-Ct} \\ k &= bC \\ C &= \frac{k}{b} \\ x(t) &= De^{-\frac{k}{b}t}. \end{aligned}$$

Now, we can treat this as an initial value problem where $x(0)$ is the initial distance x_0 ,

$$\begin{aligned} x(0) &= De^0 \\ x(0) &= x_0 \\ x_0 &= D \\ x(t) &= x_0e^{-\frac{k}{b}t} \end{aligned}$$

Such a solution can now be directly evaluated at any desired t . From a computational perspective, this solution is, however, not necessarily as ideal as it may at first seem. If we assume the $-\frac{k}{b}$ term is evaluated beforehand only once, evaluation at every time step requires two multiplications and one function call (for the exponent). It will be seen later, that the numerical solution to this same system can be done more efficiently.

Other techniques exist for the solution of more complicated differential equations. Notable of these is the use of the Laplace transform and z-transform. A detailed explanation of these transformations will not be discussed in this thesis, primarily because this method was not employed for our viscoelastic method. However, many engineering and applied math-

ematics textbooks cover this method (e.g., [19]). Essentially, the Laplace transform is used to solve a continuous differential equation, by converting the equation into the frequency domain (much like the Fourier transform). For a closed form solution, the result can be converted back into the time domain. The role of the z-Transform is to discretize the solution arising from the Laplace transform, while it is still in the frequency domain. This allows for avoiding a closed form continuous solution to the original differential equation.

4.2 Numerical Solutions

For many kinds of problems, no closed form analytical solution may exist. Furthermore, even in cases where such a solution does exist, it may be complicated enough that we are willing to instead use a more quickly computable approximation that can be determined by a general method. A number of books covering the subject have been written (e.g., [48, 49]), which the reader may consult for a much more complete explanation of these and other related methods.

4.2.1 Discrete Approximations of Derivatives

The numeric solution of a differential equation of time is based upon discretizing the derivatives in terms of desired or known functions. Discrete finite differences approximations can be derived from a Taylor series expansion of the desired function around arbitrary relative samplings. Using the expansion around point $x(t-h)$, a simple formula for \dot{x} in terms of $x(t)$

and $x(t - h)$ can be derived:

$$\begin{aligned}x(t - h) &= x(t) - h\dot{x}(t) + O(h^2) \\h\dot{x}(t) &= x(t) - x(t - h) + O(h^2) \\\dot{x}(t) &= \frac{x(t) - x(t - h)}{h} + O(h).\end{aligned}$$

Leaving out the error term $O(h)$, gives,

$$\dot{x}(t) = \frac{x(t) - x(t - h)}{h}. \quad (4.1)$$

Because the error term is linear ($\in O(h)$), the approximation is known as first order.

Deriving an approximation with a higher (better) order or for a different derivative (such as \ddot{x}) requires a more involved process. With a three-point scheme (the function evaluated at t , $t - h$, and $t - 2h$), a Taylor series expansion around all three points is necessary:

$$\begin{aligned}x_0 &= x(t) = x(t) \\x_1 &= x(t - h) = x(t) - h\dot{x}(t) + \frac{1}{2}h^2\ddot{x}(t) + O(h^3) \\x_2 &= x(t - 2h) = x(t) - 2h\dot{x}(t) + 2h^2\ddot{x}(t) + O(h^3).\end{aligned}$$

Because we want to estimate the first derivative, the following equality for our resulting approximation must hold,

$$0 \cdot x(t) + 1 \cdot \dot{x}(t) + 0 \cdot \ddot{x}(t) = \alpha_0 x_0 + \alpha_1 x_1 + \alpha_2 x_2 + E, \quad (4.2)$$

where E is the error due to the approximation. Substituting in the Taylor series expansions of the sampled points, with the higher order terms

becoming E, gives

$$\begin{aligned} & 0 \cdot x(t) + 1 \cdot \dot{x}(t) + 0 \cdot \ddot{x}(t) \\ = & (\alpha_0 + \alpha_1 + \alpha_2)x(t) - h(\alpha_1 + 2\alpha_2)\dot{x}(t) + h^2\left(\frac{1}{2}\alpha_1 + 2\alpha_2\right)\ddot{x}(t) . \end{aligned}$$

This linear system can be solved by comparing coefficients. Rewritten in matrix form (with the h terms brought to the right),

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & -1 & -2 \\ 0 & \frac{1}{2} & 2 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{1}{h} \\ \frac{0}{h^2} \end{pmatrix}$$

solutions $\alpha_0 = \frac{3}{2h}$, $\alpha_1 = -\frac{2}{h}$, and $\alpha_2 = \frac{1}{2h}$ are obtained. After substituting back into the right side of Equation 4.2 (ignoring the error term) gives our final approximation,

$$\dot{x}(t) = \frac{3}{2h}x_0 - \frac{2}{h}x_1 + \frac{1}{2h}x_2 = \frac{3x(t) - 4x(t-h) + x(t-2h)}{2h}. \quad (4.3)$$

It is important to analyze the order of error of any approximation. If we substitute the full Taylor series expansion of each sampled point into Equation 4.2, the error bound becomes,

$$\begin{aligned} \dot{x}(t) + E &= \frac{3}{2h}x(t) - \frac{2}{h}x(t) + 2\dot{x}(t) - h\ddot{x}(t) \\ &+ \frac{1}{2h}x(t) - \dot{x}(t) + h\ddot{x}(t) + O(h^2) \\ &= 0 \cdot x(t) + 1 \cdot \dot{x}(t) + 0 \cdot \ddot{x}(t) + O(h^2) \\ E &\in O(h^2) \end{aligned}$$

which means Equation 4.3 is a second order approximation. Taking more samples typically results in higher approximation orders, which in turn leads to less error as $\forall k > 0, \forall h \in (0, 1), h^{k+1} < h^k$. However, it also re-

sults in more complicated mathematical formulas for the approximations, which can increase simulation time. For example, the first order approximation mentioned in Equation 4.1 can be computed with one subtraction and one division (or multiplication if h is constant and can be inverted before hand). However, the second order approximation in Equation 4.3 requires four multiplications, one subtraction, and one addition or if h is constant and necessary precomputations are made, three multiplications, one subtraction, and one addition.

4.2.2 Explicit (Euler) Methods

Let us examine the same simple differential equation initial value problem from Section 4.1,

$$kx(t) = -b\dot{x}(t) \quad x(0) = x_0$$

For the numeric solution of any differential equation we want to know the value of x after the next time step, $x(t + h)$. The simplest way to construct the numeric solution to this is to substitute an approximation like the one derived in Equation 4.1 for the derivative term(s), giving

$$\begin{aligned} kx(t) &= -b \left(\frac{x(t+h) - x(t)}{h} \right) \\ kx(t) - \frac{b}{h}x(t) &= -\frac{b}{h}x(t+h) \\ \left(k - \frac{b}{h} \right) x(t) &= -\frac{b}{h}x(t+h) \\ -\frac{h}{b}x(t) \left(k - \frac{b}{h} \right) &= x(t+h) \\ x(t+h) &= x(t) \left(1 - \frac{kh}{b} \right) \end{aligned}$$

This is known as an *explicit* method of solution. The above equation has the effect of causing repeated multiplication by $(1 - \frac{kh}{b})$ as time marches on in our simulation. This is simply the discrete equivalent of the exponential decay seen in the analytical solution.

Like any other numeric solution, those obtained through an explicit method are just approximations. However, explicit methods suffer from certain particularly severe approximation issues. In this example, the major issue arises most visibly in the case where $|1 - \frac{kh}{b}| > 1$. In this case, there are two possible outcomes (depending on the sign of the term): $x(t)$ grows constantly larger or $x(t)$ oscillates between negative and positive with constantly larger absolute values. As time marches on, this dampened spring will get larger (oscillating or simply going in one direction), all while no force is being applied. From the analytical solution, the expected outcome should be a gradual approach to zero. Therefore, the approximation errors may not be simple inaccuracies, but rather cause dramatic unrealistic behavior.

Fortunately, this problem is avoidable. For any arbitrary k and b , an h can be chosen such that $|1 - \frac{kh}{b}|$ remains less than 1. In order to guarantee even vaguely realistic behavior for a damped spring with a large spring constant, an extremely small time steps must be used. This has the highly undesirable effect of making the simulation take much longer than would be necessary if accuracy was only dictated by time step size. Because a high spring constant represents a stiff spring, such differential equations are known as “stiff differential equations”.

However, it is worth pointing out that if stability is not a concern in the case of non-stiff equations, this explicit solution can be calculated very quickly. Assuming b and k are constant and that their quotient is deter-

mined once ahead of time, each step can be computed with two multiplications (one if h is constant as well) and one subtraction.

4.2.3 Implicit (Reverse Euler) Methods

In the explicit (or Euler) solution methods, $x(t)$ is used along with a forward derivative approximation ($\dot{x}(t) = \frac{1}{h}(x(t+h) - x(t))$). With implicit methods, the derivative approximation is the same, but because $x(t+h)$ is used instead of $x(t)$, the approximation can now be viewed as a backward one:

$$\begin{aligned} kx(t+h) &= -b \left(\frac{x(t+h) - x(t)}{h} \right) \\ \left(k + \frac{b}{h} \right) x(t+h) &= \frac{b}{h} x(t) \\ x(t+h) &= \frac{\frac{b}{h}}{k + \frac{b}{h}} x(t). \end{aligned}$$

Like the explicit method solution for this problem, the solution here can be characterized as a repeated multiplication by a constant. However, do we still face the problem of that constant becoming larger than 1 or smaller than -1 ? If we make the perfectly reasonable assumption of $b \geq 0$ and $k \geq 0$,

$$\begin{aligned} 1 &\geq \left| \frac{\frac{b}{h}}{k + \frac{b}{h}} \right| \\ k + \frac{b}{h} &\geq \frac{b}{h} \\ k &\geq 0, \end{aligned}$$

the answer to that question is no (well, at least as long as a reasonable spring coefficient is used). Although larger time steps still mean less accuracy, time steps of arbitrary size will not cause the stability problems present in the explicit method.

The price paid for this new ability, however, is that our formula is somewhat more complicated. Making the same assumptions as with the explicit solution above, one multiplication, one addition, and one division. If it is further assumed that the time step size is constant, everything collapses to just one multiplication per step, as in the explicit approach. Therefore, for constant time step sizes, nothing has really been lost and everything gained. For variable time step sizes, the situation is not terribly worse, since on modern hardware the difference in execution time between a division and multiplication is fairly narrow. Still, the time saved by the ability to take arbitrary sized steps outweighs any additional costs from the division.

Particle system components whose simulation is aided by the methods given in this chapter provide a physically based approach to deformable object modeling. However, there are other ways of characterizing such behavior. One such way is through the use of discrete Green's function matrices, which will be discussed next.

Chapter 5

Discrete Green's Function

Matrices

A discrete Green's function matrix (or DGFM) linearly relates the traction and displacement of nodes on a discrete mesh for a given boundary configuration. In this thesis, the vertices in a polygonal mesh representing the surface of an object will be treated as these nodes. Each of the n vertices has either a displacement constraint or a traction constraint. Following the notation used by James [30], from the set Λ , consisting of all the vertices in the mesh, two exclusive subsets are defined: Λ_u , which consists of the vertices with displacement constraints and Λ_v , which consists of those with traction constraints. A given DGFM is specific to these two constraint sets as well as the object's mesh and properties. The actual constraint values are three-dimensional real valued vectors. Let $\bar{\mathbf{u}}_i$ denote either the known displacement constraint or $\bar{\mathbf{p}}_i$ the known traction constraint on vertex i , depending on the corresponding constraint type of i .

Given these constraints, our goal is to solve for the unknown traction for displacement constrained nodes and for the unknown displacement for

traction constrained nodes. This leads us to the notation of either \mathbf{p}_i for the unknown traction (if displacement constrained) or \mathbf{u}_i for the unknown displacement (if traction constrained) of node i . These values can be combined into block vectors of n dimension,

$$\mathbf{v}_j = \begin{cases} \mathbf{p}_j & : j \in \Lambda_u \\ \mathbf{u}_j & : j \in \Lambda_v \end{cases}$$

and

$$\bar{\mathbf{v}}_j = \begin{cases} \bar{\mathbf{u}}_j & : j \in \Lambda_u \\ \bar{\mathbf{p}}_j & : j \in \Lambda_v \end{cases}$$

With these new n length block vectors it is easy to produce a linear relationship between known and unknown quantities,

$$\Xi \bar{\mathbf{v}} = \mathbf{v} \tag{5.1}$$

where Ξ is an $n \times n$ DGFM containing 3-by-3 block entries. Solving for the unknown quantities is as simple as performing a matrix multiplication. The element from the i -th row and j -th column of the matrix forms the additive influence the known j -th element of $\bar{\mathbf{v}}$ has on the unknown i -th element of \mathbf{v} .

To better understand the meaning of the entries of a DGFM, in Figure 5.1 an extremely simple example has been constructed for a tetrahedron. Let us assume that vertices 1, 2, and 3 are traction constrained, while 4 is displacement constrained. In the simplest case, if we apply a traction of 1 unit in the x direction on vertex 1, zero traction on vertices 2 and 3, and

$$\Xi = \begin{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \end{pmatrix} & \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{4} & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{pmatrix}$$

Figure 5.1: a trivial discrete Green's function matrix for a four point mesh (tetrahedron)

zero displacement on vertex 4, the resulting $\bar{\mathbf{v}}$ is,

$$\left((1 \ 0 \ 0)^T \ (0 \ 0 \ 0)^T \ (0 \ 0 \ 0)^T \ (0 \ 0 \ 0)^T \right)^T$$

multiplying with Ξ gives \mathbf{v} ,

$$\left((1 \ 0 \ 0)^T \ (0 \ 0 \ 0)^T \ (0 \ 0 \ 0)^T \ (0 \ 0 \ \frac{1}{4})^T \right)^T$$

This result can be interpreted as the application of 1 unit of traction in the x direction, yields a displacement of 1 unit in the same direction on vertex 1 as well as a traction of $\frac{1}{4}$ unit on vertex 4.

5.1 Behavior at Non-Vertex Locations

So far only the relation between tractions and displacements on specific discrete vertices of the mesh has been given. For most sorts of interesting

simulation, it is necessary to define what the behavior should be for other surface points, which still lie on the surface, but between mesh vertices. There are various ways of handling this case, each of which must be taken into account when generating or acquiring a correct DGM. Commonly (and for this thesis as well), the behavior of a point on a particular face of the surface mesh will be the sum of the behavior of each vertex of said face scaled by its respective barycentric coordinate with respect to the original point in question. This allows for a smoothly interpolated, continuous description of the behavior at all points on the surface of the object.

5.1.1 Barycentric Coordinates

Let \mathbf{p} be the point on the triangular face consisting of points \mathbf{q}_1 , \mathbf{q}_2 , and \mathbf{q}_3 , from which the barycentric coordinates β_1 , β_2 , and β_3 respectively will be determined. First, for our problem is necessary that if point \mathbf{p} coincides with any of the vertices \mathbf{q}_i , β_i is defined to be 1 and the other coordinates are defined to be 0. This condition guarantees that none of the values at known vertex points will be disturbed. Second, we would like smooth interpolation of values for all points that do not lie on a mesh vertex. This guarantees a smooth function over of the entire mesh surface, by linearly combining the values. In Figure 5.2, the one dimensional case is presented. For a given point between vertex 0 and 2, its barycentric coordinates (which are conceptually nothing more than weights) are determined by the value of the corresponding hat functions at the point of interest. The behavior of any such point is then given as the sum of the behavior at two of the points multiplied by their corresponding weight.

Because the problems discussed in this thesis take place in three dimensions, this approach must be extended to deal with that case. In three

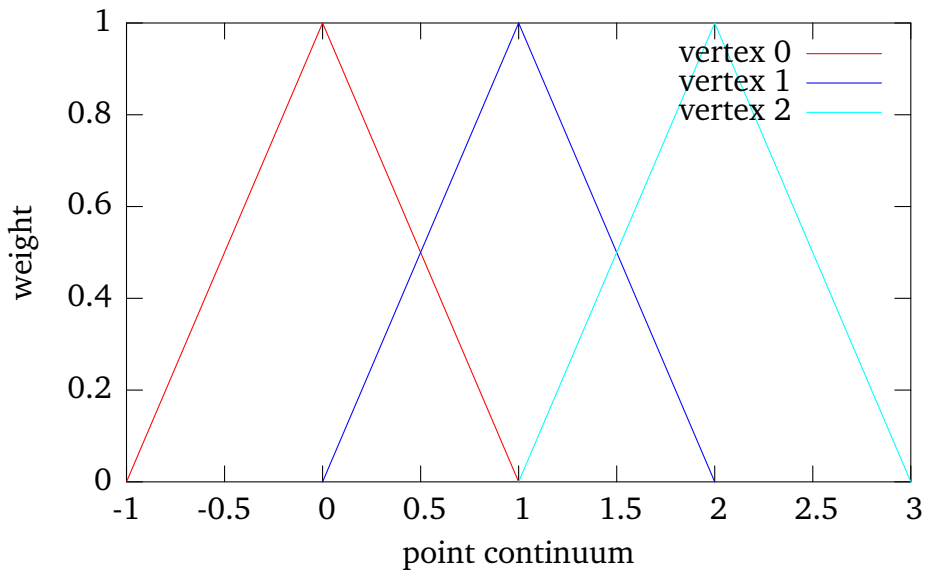


Figure 5.2: Hat functions around vertices 0, 1, and 2, demonstrate linear blending in one dimension.

dimensions, using similar reasoning as found in the one dimensional case, it would seem that we would need four barycentric coordinates to characterize an arbitrary point. However, because we are only interested in the case where point \mathbf{p} is coplanar to the enclosing triangle face, one degree of freedom is eliminated from the system and therefore in this case we can get along with just three coordinates. Unfortunately, this assumption of coplanarity may not hold due to numerical imprecision arising in common floating point hardware. Therefore, it is necessary to couch the solution to this problem in terms of four coordinates (β_1 , β_2 , β_3 , and ω). However, as long as the planarity constraint violation is minimal, the extra coordinate can be ignored after the computation has been completed.

The 3D solution is given in matrix form by

$$\begin{pmatrix} \mathbf{q}_{1,x} & \mathbf{q}_{2,x} & \mathbf{q}_{3,x} & 1 \\ \mathbf{q}_{1,y} & \mathbf{q}_{2,y} & \mathbf{q}_{3,y} & 1 \\ \mathbf{q}_{1,z} & \mathbf{q}_{2,z} & \mathbf{q}_{3,z} & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \omega \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix},$$

where $\mathbf{q}_{a,b}$ represents the b -th component of \mathbf{q}_a . Generally ω should be 0 in the case where point \mathbf{p} is perfectly coplanar with \mathbf{q}_1 , \mathbf{q}_2 , and \mathbf{q}_3 . If it is not perfectly coplanar due to imprecision, it should still be very close to zero.

This new system can be easily inverted, so long as all of the vertices of the face are distinct. Additionally, it's not strictly necessary to perform an entire matrix-vector multiplication between the inverse of the matrix and the right-hand vector. Generally, we are only interested in β_1 , β_2 and β_3 , as ω should be 0 except in the cases mentioned above. Therefore, multiplication with the last row of the inverted matrix to determine ω is not strictly necessary.

5.2 The Boundary Element Method

Determining a physically consistent DGFM by hand even for a simple tetrahedral solid is impractical. For any sort of interesting object of higher complexity it might as well be impossible. There exist a number of automated ways for computation of a DGFM, one of which is based on the BEM. In this thesis only enough information on the BEM for construction of a DGFM for a triangular mesh will be provided. For a more detailed explanation of it, the standard text on this broad subject (of which elastostatics is only one

of many application areas) by Brebbia et al. [27] is quite comprehensive.

The notation here continues to follow that of James. As mentioned in the general introduction to DGFMs above, a matrix is only valid for the specific Λ_u and Λ_v constraint sets. Any change in these constraints from one type to the other, requires recomputation of a DGF, which is not possible without further information. Let us take a broader view of the linear relation present in a DGF and characterize the system in a more general manner, with all constraints as unknowns. Such a linear relation can be seen in,

$$0 = \mathbf{H}\mathbf{u} - \mathbf{G}\mathbf{p} = \sum_{j=1}^n h_{i,j}\mathbf{u}_j - \sum_{j=1}^n g_{i,j}\mathbf{p}_j$$

where \mathbf{p} , \mathbf{u} , and n have the same meaning as above (unknown tractions and displacements, and a known number of nodes respectively), while \mathbf{G} and \mathbf{H} (formed from the 3-by-3 block entries $g_{i,j}$ and $h_{i,j}$ respectively) are derived from the mesh and some parameters representing material properties.

From \mathbf{G} and \mathbf{H} , two new matrices \mathbf{A} and $\bar{\mathbf{A}}$ can be constructed based upon the specific boundary constraints applied. The arrangement of their columns follows a similar process as the boundary condition values in \mathbf{v} and $\bar{\mathbf{v}}$ above. Their definitions are given in terms of their columns:

$$\mathbf{A}_{:j} = \begin{cases} -\mathbf{G}_{:j} & : j \in \Lambda_u \\ \mathbf{H}_{:j} & : j \in \Lambda_v \end{cases}$$

and

$$\bar{\mathbf{A}}_{:j} = \begin{cases} \mathbf{H}_{:j} & : j \in \Lambda_u \\ -\mathbf{G}_{:j} & : j \in \Lambda_v \end{cases}$$

Finally, it can be shown that our DGFM for these constraint sets is defined according to,

$$\Xi = -\mathbf{A}^{-1}\bar{\mathbf{A}}$$

This of course allows for the recomputation of a new DGFM in the case of boundary constraints, so long as \mathbf{G} and \mathbf{H} are known. Unfortunately, performing the computation using the above formula is prohibitively expensive for interactive display and especially haptics of interesting meshes. This stems from the cost of explicitly inverting the large dense matrix \mathbf{A} . At least one solution to this problem has been proposed that allows efficient computation for real-time haptic interaction [29, 30].

5.2.1 Boundary Integrals

The boundary element integrals for the computation of \mathbf{G} and \mathbf{H} will not be provided here. Brebbia [27] gives a fairly detailed account of the integrals necessary for calculating these values. However, for a triangular mesh some of these integrals have singularities and cannot be used. The appendix of James' and Pai's ArtDefo paper [29] discusses in some detail how to get around this problem.

It is worth noting here that these integrals contain parameters for such global properties as compressibility (the Poisson ratio) and overall stiffness (Young's modulus). The Poisson ratio is a scalar value that is often written with the Greek letter ν . A ratio of 0 means the material is fully compressible, while a ratio of 0.5 means the material is fully incompressible. For example, rubber is a substance with a ratio near 0.5, wood is closer to 0, and most metals are somewhere in the middle. Young's modulus is a positive scalar proportional to stress over strain that is typically written with the

Latin letter E [50]. Traditionally, these values have been measured physically from a material sample (see related work in Section 2.4 for details).

5.3 Direct Measurement

It is also possible to determine the parts of a DGFM necessary for most haptic and visual purposes of real world objects through automated measurement as opposed to idealized calculations on meshes or tetrahedralizations.

The approach of Lang et al. [2] involves the use of a robotic arm and cameras to measure the observed deformation when forces are applied to a point on the surface of the model. Many measurements are taken with deformation being applied from various directions. It is important to note that a complete DGFM is not always produced. In this specific approach, a displacement is applied to locations corresponding to non-fixed vertices of the mesh. The probe making the displacement then measures the force exerted at that specific vertex. This is ultimately used to compute the diagonal elements of the DGFM. Other entries of the DGFM are computed by using cameras to measure observed displacement resulting from the displacement. Hence, traction can be known only at traction constrained vertices, while displacement can be known at all vertices. No attempt is made to measure traction at the displacement constrained vertices, which are fixed.

Because virtually all real world objects exhibit time-dependent or non-linear deformation behavior, the techniques used must make necessary approximations of actual behavior for constructing the DGFM of an object. One such behavior is that of stress relaxation, which forms the basis of material discussed in Chapter 7.

5.4 Mesh Subdivision Considerations

In certain cases it may be necessary or useful to use a finer mesh for smoother appearance or feel. This requires a new DGFM to match the new geometry of the refined mesh and the behavior of the original mesh. With regard to DGFMs that have been computed through use of the BEM, the obvious approach is to simply perform subdivision then recompute a new DGFM from the refined mesh. Running the BEM on a mesh can be a time consuming process, so it would be nice to have some sort of faster approximation. With a DGFM that has been physically acquired, the simplest approach would be to reacquire the DGFM at a finer resolution. This presents two possible issues though. First, the robot arm may not be able to take more precise measurements due to its own physical limitations. Second, it may not be practical to acquire a new DGFM because of time constraints.

For these reasons it is convenient to perform subdivision on the existing mesh and to interpolate reasonable values for the new DGFM. Although only Loop subdivision [51] will be specifically discussed here, it is possible to use a variety of subdivision schemes here in an analogous manner.

A modified full algorithm for performing Loop subdivision based on work by Pai et al. [47] follows.

1. Allocate enough space for the new DGFM: $v + e$ by $v + e$, 3 by 3 where v is the number of vertices and e the number of edges in the original mesh. Copy the corresponding values from original the DGFM into this new one, leaving other entries blank for now.
2. Allocate enough space for the boundary condition types of the new DGFM: $v + e$. Copy the corresponding values from the original condi-

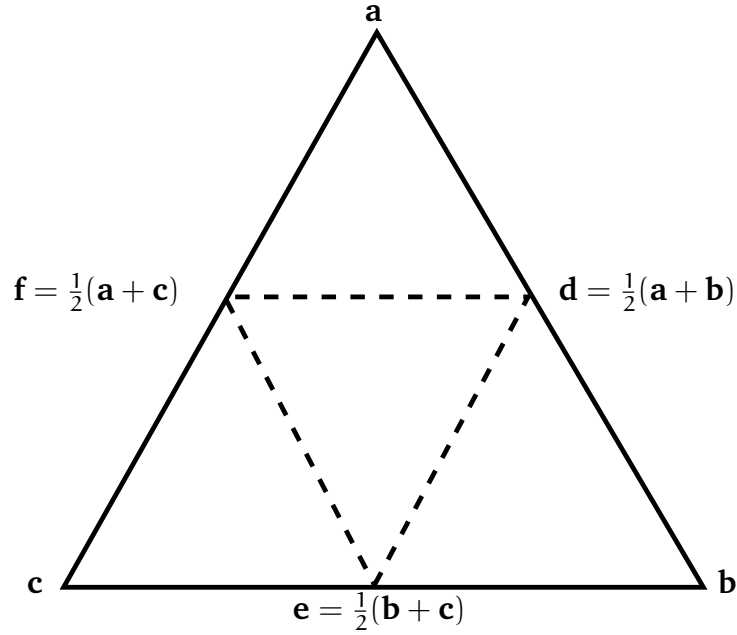


Figure 5.3: Loop subdivision pattern.

tions into this new one, leaving the other entries unset.

3. Subdivide each triangle in the mesh, keeping track of all the new points (\mathbf{d} , \mathbf{e} , and \mathbf{f} in Figure 5.3) and their combination of the original points (\mathbf{a} , \mathbf{b} , and \mathbf{c}). To determine the boundary condition type of a new vertex, examine its two parent vertices. If both have the same condition type, the new vertex takes that type as well. If not, the new vertex takes a fixed displacement constraint type¹.
4. For each new vertex V added in the subdivision step 2, where I and J are its parent vertices, set the row $\Xi_{V,\cdot} := \frac{1}{2}(\Xi_{I,\cdot} + \Xi_{J,\cdot})$

¹This only works for display and haptic purposes as discussed in this thesis. It does not work in general for the case where a DGFM may be multiplied by a non-zero displacement constraint value. In this thesis, we make the stricter constraint that all displacement constrained points are fixed and thus have a zero displacement constraint value always. This, in turn allows the bogus values placed in the tensor to not have an effect on the outcome, so long as this tighter constraint is kept. This appears later in more detail in Section 7.1.

5. For each new vertex V added in the subdivision step 2, where I and J are its parent vertices, set the entry for each x ,

$$\Xi_{x,V} := \begin{cases} \frac{1}{2} (\Xi_{x,I} + \Xi_{x,J}) & x \neq V \\ \frac{1}{2} (\Xi_{I,I} + \Xi_{J,J}) & x = V \end{cases}$$

This separation of cases preserves the stronger response present in the diagonal matrix entries, which describe the behavior at a given node occurring when that same node is acted upon.

6. Perform averaging on all vertices on the mesh as many times as desired.
7. Repeat entire process as desired.

It is worth noting that the refinement does not need to be done the same in both dimensions of the DGFM. For example, it can be useful to only subdivide the display mesh and not the haptic mesh. In such a case, the new matrix only requires a dimension of $v + e$ by v and step 5 can be ignored.

The DGFM is an efficient way of characterizing the linear deformation behavior of an object. As we will see in the next chapter, it also works well with regard to interactive haptic and visual applications.

Chapter 6

Haptics Fundamentals

Haptic is an adjective defined as “of or relating to the sense of touch; tactile” [52]. The term *haptics* in the context of this thesis refers to the field of computing, where methods for input and output of touch are dealt with. The material comprising this chapter is based on work described in Chapter 2 and the documentation for SensAble’s™ PHANTOM® Desktop haptic device and its associated GHOST® SDK (software development kit) [53].

All modern computer users are familiar with motion reliant input devices such as keyboards and mice. In particular, a mouse takes motion input in two dimensions by allowing the user to arbitrarily move the device across a plane with his or her hand. What distinguishes haptic devices from devices like mice is that haptic devices are capable of generating output forces as well, which can be felt by the user. Because this thesis deals with the simulation of three dimensional objects, the haptic devices discussed in this thesis are those that are capable of obtaining input and producing output forces in three dimensions as well.

6.1 Basics

Much like with 3D computer graphics software, software interacting with haptic devices often runs in a high speed loop, in which input is monitored or taken, necessary computations are made, and finally output is rendered. However, unlike a graphics loop, which must run at a minimum rate between 30Hz and 60Hz for the best user experience, a haptic loop needs to run at around 1000Hz [54]. After overhead is taken into account, this of course limits the amount of time that can be spent on one cycle to less than one one-thousandth of a second.

At any point in time, the user may be holding the device at an arbitrary position with an arbitrary orientation. Typically, a haptic device operates in a particular volume, which can be characterized by the standard 3D Cartesian coordinate system (x , y , and z axes). First, the position of the device is converted into a 3D point in the global coordinate space. This vector quantity is known as the *proxy* and forms the most fundamental component of input. Because it is so important, it is usually updated on each discrete time step taken by the haptic loop.

On the output side, over any discrete time step a force can be assigned to be rendered. An arbitrary force is generated by combining forces from several motors that are oriented in different directions and may even be in different parts of a segmented device. Although the constraints upon this force are dependent on the specific device being used, one constraint is universal: the device cannot render force above a certain threshold. The motors in the device can only provide so many newtons of force. The software interface to the device typically provides some means of notification that this threshold has been reached or exceeded and a means of specifying what should happen in such a case.

6.2 Higher-Level Constructs

Developing any sort of non-trivial piece of haptic software by reading the proxy position and controlling forces directly can be cumbersome. For this reason, higher-level SDKs have been developed that make implementation in most common cases significantly easier. One such SDK is the above mentioned GHOST® SDK developed for SensAble Technology's product line. Because this software package enjoys broad usage for haptics in computer graphics and because it was used in the implementation of the work that this thesis describes, discussion here will be specific to it. Nonetheless, the issues discussed here are mostly applicable to other similar SDKs.

The GHOST® SDK relies on the scene graph construct common to graphics applications. Static, rigid objects such as primitive volumetric quadrics (spheres, cones, tori, etc.) and triangular meshes as well as untouchable transformation nodes (scaling, translation, rotation, etc.) can be added. Another particularly useful component provided is built-in collision detection. More complicated dynamic constructs are also provided, which allow the user, for example, to move objects in the scene. Typically, no end-developer code needs to be added directly to the haptic loop for these cases.

Now that scene geometry and collision detection have been abstracted away, abstractions for force can be examined. When an object and the proxy come in contact with each other, a reactive force needs to be generated. Because force is a 3D vector quantity, it can be considered to have two components: direction and magnitude. Force direction can be determined from the normal of the surface at the collision point, which can be retrieved from geometric data structures in the SDK. Magnitude is, however, not quite as straightforward. A perfectly ideal rigid object (conceptually, this can be thought of as an infinitely stiff surface) will exert an infinite

amount of force in the collision.

Because obviously infinite force cannot be rendered, it is often reasonable to instead dictate that all objects are at least slightly non-rigid and thus have a non-infinite stiffness. A relatively large stiffness coefficient (dependent on the capabilities and nature of the haptic device) can be used for all rigid objects. The force magnitude exerted in this case, $|\mathbf{f}|$, is given by,

$$|\mathbf{f}| = k|\mathbf{u}| \quad (6.1)$$

where k is the stiffness coefficient and \mathbf{u} is the vector formed by the proxy location and what is known as the surface contact point (SCP). The SCP is the point on the surface closest to the proxy. Using the SCP instead of the point of collision for determining \mathbf{u} allows for more accurate behavior¹. The quantity $|\mathbf{u}|$ then is the distance the proxy has penetrated the surface by. This has the effect that as the surface is penetrated more deeply, the force exerted rises linearly. The GHOST® SDK uses this stiffness coefficient k as the primary means of setting the amount of force response for scene objects.

6.3 Handling Elastic Deformable Objects

The reader should recognize Equation 6.1 as similar to that of a standard linear spring (Equation 3.1 with $l_r = 0$) and reminiscent of that governing the DGFM (Equation 5.1). In fact, all three somehow relate force to displacement in a linear manner. Because of this affinity, handling linear

¹For example, using the SCP allows the user to slide over the surface and get a reasonable effect. If the collision point were used in this case, reactive force would be based on the distance from first collision. Sliding over the surface would result increasing force, which would give the feeling of handling an extremely sticky surface.

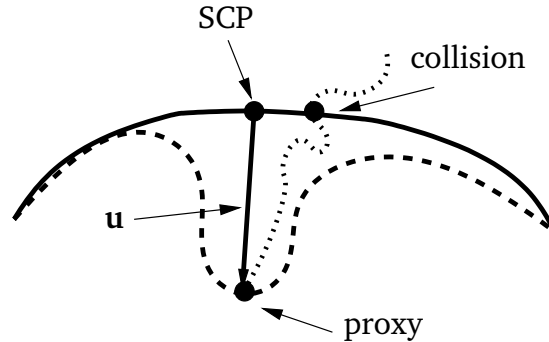


Figure 6.1: Example of surface contact point (SCP), surface collision point and proxy location. The dotted line is the proxy path over time. The solid line is the undeformed surface. The dashed line is the deformed surface.

deformable objects as described by a DGFM is fairly straightforward. In each step of the haptic loop, the following occurs (see Figure 6.1 for a depiction of terms):

1. Determine if contact with the object is occurring. If not, nothing more needs to be done and the remaining steps can be skipped.

In the case of a point contact model, collision occurs when the proxy has penetrated the object's surface.

2. Compute the stiffness through use of the DGFM and collision details.

With a point contact model on a triangle mesh, one simple approach can be done with the proxy location ($\mathbf{u}_{\text{proxy}}$), the SCP (\mathbf{u}_{SCP}), and the latter's barycentric coordinates.

- (a) Let the vertices of the triangle enclosing the SCP comprise the set S . Let S_t be the subset consisting of the vertices that are traction constrained and S_d be the subset consisting of the vertices that are displacement constrained. ($S_d \cup S_t = S$, $S_d \cap S_t = \{\}$)

For each $\mathbf{v}_n \in S_t$, its corresponding diagonal DGFM entry is in-

verted to produce an ad hoc stiffness tensor:

$$\mathbf{K}_n = \boldsymbol{\Xi}_{n,n}^{-1}.$$

For each $\mathbf{v}_m \in S_d$, a stiffness scalar is set directly²:

$$k_m = k_{\text{rigid}}.$$

This maximal stiffness, k_{rigid} should be the relatively large stiffness coefficient used for rigid objects. In the GHOST® SDK this is 1.

- (b) These tensors and coefficients are then averaged together appropriately with their corresponding barycentric coordinate values (β_i) as weights to produce an interpolated stiffness coefficient at the SCP

$$k_{\text{SCP}} = \sum_{\mathbf{v}_i \in S_t} \beta_i \left| \alpha_i \mathbf{K}_i \frac{\mathbf{u}}{|\mathbf{u}|} \right| + \sum_{\mathbf{v}_i \in S_d} \beta_i k_i,$$

where $\mathbf{u} = \mathbf{u}_{\text{proxy}} - \mathbf{u}_{\text{SCP}}$ and α_i is the area of node i (for traction to force conversion).

3. Set the current stiffness coefficient as determined in step 2.

While the above steps are performed to determine the effects of the deformation local to the area of contact at each haptic time step, the global deformation of the object needs to be determined for display in the graphics loop at its lower rate. This process can be broken down as follows:

²Once again, the assumption is made that all displacement constrained vertices are actually fixed. Therefore, they are not allowed to deform at all and should behave like a rigid object. However, it is necessary to handle the case where enclosing vertices have mixed conditions. Hence, these scalars will be used to interpolate a smooth transition between fixed and non-fixed parts of the object mesh.

1. Determine tractions acting on the object.

For a point contact model, the process is similar to step 2 for the haptic handling. Therefore, the same notation and definitions there hold here as well.

- (a) For each $\mathbf{v}_n \in S_t$, solve the following linear system to obtain the local traction vector applied that vertex, \mathbf{p}'_n ,

$$\Xi_{n,n} \mathbf{p}'_n = \mathbf{u}$$

- (b) Let the overall traction vector be defined as,

$$\forall i, i \in \{0, 1, \dots, n-1, n\}, \mathbf{p}_i = \begin{cases} \beta_i \mathbf{p}'_i & \text{if } \mathbf{p}'_i \text{ exists} \\ 0 & \text{otherwise} \end{cases}$$

2. Calculate the resulting deformation.

For a point contact model, this is done by performing a dense matrix, sparse vector multiply to determine the displacement vectors for all vertices, assuming any displacement constrained vertices have a zero displacement constraint.

3. Draw all polygons in the mesh using the displaced locations of all vertices obtained by adding the corresponding deformation vectors and undeformed vertex locations.

Handling the haptic rendering and the display separately allows the quantities to be largely computed independently of one another. Because nothing needs to be written into the same location by both processes, synchronization is not a problem. This allows these processes to be run in separate threads at the necessary rates without risk of data corruption.

Now that much of the background material for this thesis has been given, the problem of handling the more complicated case of viscoelasticity will be presented next.

Chapter 7

Modeling Viscoelasticity

A standard elastic model behaves independently of time; deformation is only a function of traction or forces applied. For example, when using an elastic model and forces are released from an object, the object will instantly return to its undeformed state. A viscoelastic model goes beyond this by making deformation a function of time as well, allowing more realistic gradual deformation and restoration to occur.

In this chapter, a flexible approach to viscoelasticity that combines the advantages of a DGFM (precomputability, more accurate global behavior and simpler global parameters for control) with those of particle systems (ability to model secondary motion) is presented. The crux of the approach is to replace the underlying linear, elastic equations of a DGFM with non-linear, viscoelastic equations inspired by particle system primitives.

7.1 Relating Elasticity to a DGFM

A simple linear elastic system characterized by a DGFM can be conceptually viewed as set of linear springs. Mathematically, a DGFM relates the

| | | type of constraint on vertex j | |
|-----------------------------------|--------------|--------------------------------|-----------------------------|
| | | traction | displacement |
| type of constraint on vertex i | traction | tract. \rightarrow displ. | displ. \rightarrow displ. |
| | displacement | tract. \rightarrow tract. | displ. \rightarrow tract. |

Table 7.1: This table shows the relation between two vertices i and j , specified by their corresponding DGFm entry ($\Xi_{i,j}$ or i -th row, j -th column). Entries take the form known constraint value applied to $j \rightarrow$ unknown constraint value produced on i .

known and unknown quantities at all n vertices through linear combination. However, the relationship dictated by a given DGFm entry for its two corresponding nodes falls into one of four possible classes, depending on the boundary conditions of the nodes. The conditions for these four classes as well as their resulting relationship can be seen in Table 7.1. Of the four classes, only two directly lend themselves well to a linear spring, because only these two involve traction and displacement. Ideally, handling just one case would provide the most simplicity.

Fortunately, by enforcing additional stricter constraints, all but one of the four cases can be safely ignored. When simulating for haptics and display purposes, we are generally only interested in obtaining the displacement for the vertices that are force constrained. Although the traction present on displacement constrained vertices can be calculated, the quantity is not needed for our purposes. Furthermore, the only vertices that have been designated displacement constrained for our purposes are those that have been fixed in their undeformed location. This stricter constraint results in the use of a zero vector for the displacement constraint value on these vertices, which in turn results in no contribution to the final results ($\Xi_{i,j} \cdot \mathbf{0} = \mathbf{0}$). Ultimately, this allows us to ignore any displacement constrained vertex, leaving only the case where both vertices are traction constrained. Mathematically, the DGFm in this case can be reformulated

without the columns or rows associated with these fixed points. With regard to implementation, those entries of the DGM can be discarded or simply ignored.

The displacement of vertex i in this one case follows the relation,

$$\mathbf{u}_i = \sum_j \Xi_{i,j} \mathbf{p}_j,$$

where $\Xi_{i,j}$ is the 3-by-3 matrix from the i -th row and j -th column of the DGM. The individual terms of the sum are equivalent to the equation describing a three dimensional spring, where the rest length of the spring is zero and a three dimensional tensor is used instead of a scalar for the spring constant (see Equation 3.1 for the one dimensional case). This connection can perhaps be more clearly seen when the j -th term inside the summation ($\mathbf{u}_{i,j}$) is rewritten with respect to force instead of traction as,

$$\mathbf{f}_{i,j} = \underbrace{\alpha_j \Xi_{i,j}^{-1}}_{\mathbf{K}_{i,j}} \mathbf{u}_{i,j} \quad (7.1)$$

where $\mathbf{K}_{i,j}$ is known as the stiffness tensor and α_j is the area of the j -th node.

7.2 Developing a Model

The underlying idea for the viscoelastic model in this thesis is to replace these spring-like traction-displacement relations with other non-linear ones, inspired by compositions of springs and dashpots. This provides the ability to simulate viscoelastic behavior in various ways, while retaining the DGM for primary deformation calculation.

Several models for viscoelasticity can be devised and used within this framework. The general goals for useful models:

1. The behavior resulting from the model should look and feel physically more realistic than a simple elastic model.
2. The model should be primarily governed by the DGFm in that after tractions/forces on the object have stabilized, the deformation should also stabilize by approaching the deformation produced by a corresponding simple linear model. Mathematically, this can be written more precisely with the new relation \mathbf{g} as,

$$\forall \mathbf{i}, \forall \mathbf{c}_k \in \mathbb{R}^3, \lim_{t \rightarrow \infty, \forall \mathbf{p}_k, \mathbf{p}_k \rightarrow \mathbf{c}_k} \sum_j \mathbf{g}(\Xi_{i,j}, t, \mathbf{p}_j) = \sum_j \Xi_{i,j} \mathbf{p}_j. \quad (7.2)$$

3. The relation should be somewhat continuous in that the term

$$\sum_j \mathbf{g}(\Xi_{i,j}, t, \mathbf{p}_j) \quad (7.3)$$

should not have large discontinuities over time. In the discrete time implementation, the term should not have jumps disproportionate to the input parameters (time step size, traction/force, DGFm entries) as time passes.

Furthermore, should such a jump be unavoidable, it is better to have the jump happen visually rather than haptically. This flows from the fact that display rates are usually much slower than haptic rates. The human visual system sees a large jump between frames as simply fast motion in the animation. A jump in the haptic output, however, will be felt as unnatural.

4. The model should be adjustable by a few intuitive key parameters. Manual tuning of all n^2 relations (or a large subset thereof) should not be required. Nevertheless, it should remain possible to tune all relations in an automated manner, such as one that relies on measurements acquired physically.
5. Simulating the object with the model should be fast enough to satisfy interactive haptic and display rates (approximately 1000Hz and 30Hz, respectively).

7.2.1 Kelvin-Voigt Model

Perhaps the simplest model to tackle these goals is the use of damped spring in place of the standard spring. By adding damping to the elastic deformation, a basic viscoelastic model is created. This is also known as the Kelvin-Voigt model. The one dimensional model (see Figure 4.1 for the schematic) is described by the differential equation,

$$f = ku + b\dot{u}$$

where b is the damping coefficient and k the spring coefficient. This continuous equation in turn must be discretized with respect to time. Using an implicit scheme, we obtain two relations:

$$\begin{aligned} f(t+h) &= ku(t+h) + b \left(\frac{u(t+h) - u(t)}{h} \right) \\ &= \left(k + \frac{b}{h} \right) u(t+h) - \frac{b}{h} u(t) \end{aligned} \quad (7.4)$$

$$u(t+h) = \frac{f(t+h) + \frac{b}{h} u(t)}{k + \frac{b}{h}}, \quad (7.5)$$

solving for force in terms of known displacement and solving for displacement in terms of known force, respectively.

It can be easily seen that this model provides basic viscoelasticity. When Equation 7.5 is considered with $f(t) = 0$, $u(t + h)$ is simply $u(t)$ multiplied by some constant $\frac{b}{k + \frac{b}{h}} \leq 1$. As pointed out earlier in various parts of Chapter 4, this is nothing more than a discrete approximation of exponential decay to zero. This of course satisfies the above mentioned constraint of the rest behavior approaching that dictated by the plain DGFM. See Figure 7.1 for the resulting displacement in this case and in the application of a constant force.

Now, let us examine the case where the damped spring is compressed at a constant velocity for only a finite amount of time, after which the velocity is reduced immediately to zero and thus the displacement is stopped. The force here starts out as linearly increasing, which is perfectly reasonable. Furthermore, after the displacement stops, the force drops to that, which would be exerted only by the spring. Again, this is consistent the condition layed out above where behavior should tend towards that of the plain DGFM. However, this transition is not smooth, but rather a vertical drop in force, which can be clearly seen in Figure 7.2. This of course, is not consistent with real observed physical behavior. Furthermore, such a jolting drop in force would feel particularly unnatural when rendered to a haptic device.

7.2.2 Improved Damped Model

One way to eliminate this problem induced by an abrupt change in the velocity, is to separate the system into two parts. In one part, an abrupt change should not cause such discontinuous behavior because that part will

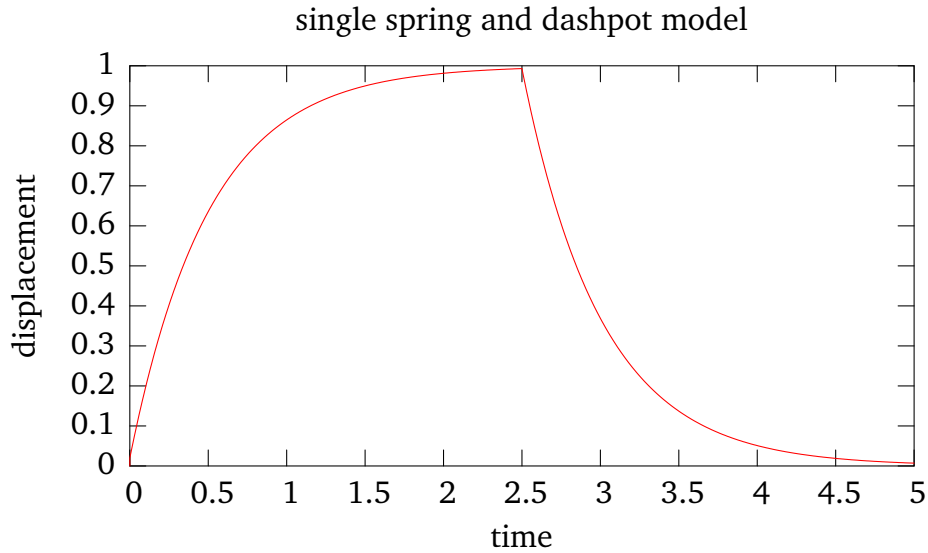


Figure 7.1: This graph shows the displacement produced from constant force ($t \in [0, 2.5]$, $f = 1.0$) followed by no force using the single spring and one dashpot model. $k = 1.0$, $b = 0.5$, $h = 0.01$

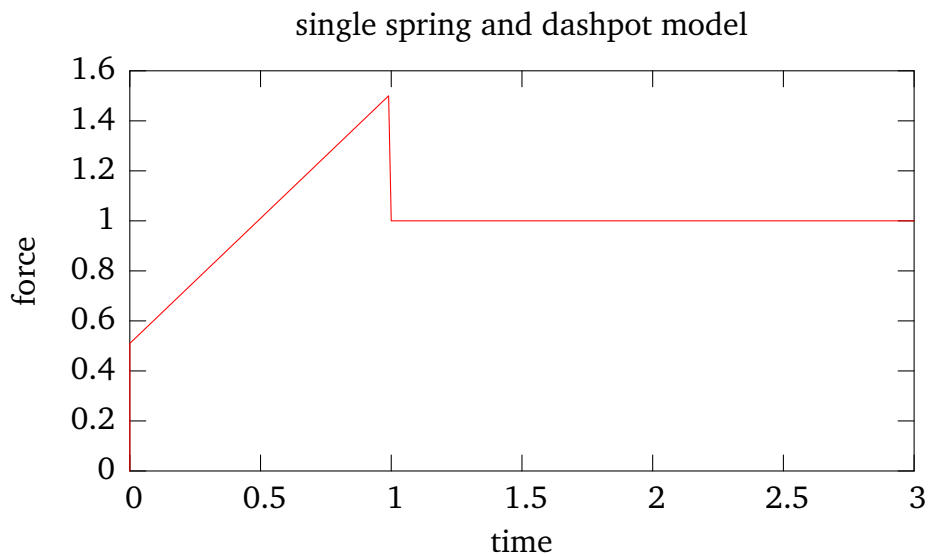


Figure 7.2: This graph shows the force produced from constant velocity ($t \in [0, 1]$, $\dot{u} = 1.0$) followed by 0 velocity using the single spring and one dashpot model. $k = 1.0$, $b = 0.5$, $h = 0.01$

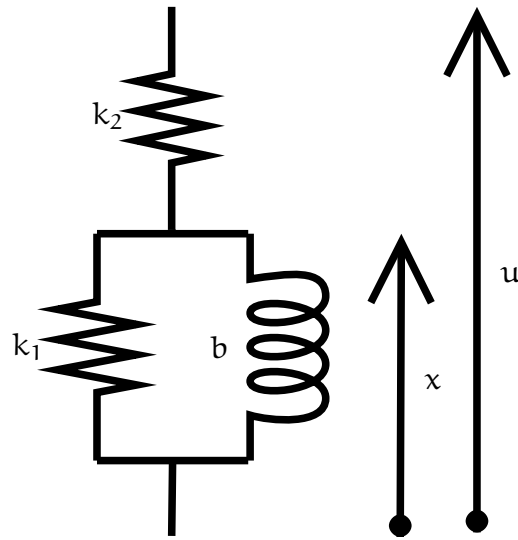


Figure 7.3: Schematic of the two spring, one dashpot model.

not be influenced by the velocity. The second part should still be affected by this problem, but will be attached to the new first part and therefore be buffered from these effects. Such a model can be devised consisting of two springs, connected in sequence, where the one not at the point of control lies in parallel to a dashpot (see Figure 7.3 for the schematic). The one dimensional relationship between force and displacement is given by the compound equation,

$$f = k_2(u - x) = k_1x + b\dot{x}, \quad (7.6)$$

where k_1 and k_2 are the two spring coefficients and b is the damping coefficient.

Using implicit schemes, the displacement in terms of force is given by,

$$u(t+h) = \frac{f(t+h)}{k_2} + x(t+h) \quad (7.7)$$

$$\begin{aligned} x(t+h) &= \frac{f(t+h) - b \left(\frac{x(t+h) - x(t)}{h} \right)}{k_1} \\ &= \frac{f(t+h) + \frac{b}{h} x(t)}{k_1 + \frac{b}{h}} \end{aligned}$$

and the force in terms of displacement is given by,

$$\begin{aligned} f(t+h) &= k_2(u(t+h) - x(t+h)) & (7.8) \\ x(t+h) &= u(t+h) - \frac{k_1 x(t+h) + b \left(\frac{x(t+h) - x(t)}{h} \right)}{k_2} \\ &= \frac{k_2 u(t+h) + \frac{b}{h} x(t)}{k_1 + k_2 + \frac{b}{h}}. \end{aligned}$$

In the case where $f(t) = 0$, like in the simpler model, $x(t+h)$ experiences nothing but an exponential decay. Unfortunately, under these same conditions $u(t+h) = x(t+h)$, which can cause an abrupt change in displacement. However, this discontinuity is not so serious as the earlier one for two important reasons. First, this governs only the displacement of the spring when there is no contact, which means a haptic device will never directly feel this dramatic change. Second, because of the exponential reduction, the velocity at the point of control will begin very quickly; the first time step is likely to move a great deal already and be indistinguishable to the viewer.

Determining whether this model tends to that of a standard linear spring is slightly more involved than in the previous case, because there are now two spring constant instead of one. Once the system comes to rest, $\dot{x}(t) = 0$. Substituting this into Equation 7.6, gives

$$f = k_2(u - x) = k_1 x$$

$$\begin{aligned}
 x &= \frac{f}{k_1} \\
 f &= k_2 \left(u - \frac{f}{k_1} \right) \\
 &= \frac{k_2}{1 + \frac{k_2}{k_1}} u \\
 &= \frac{k_1 k_2}{k_1 + k_2} u,
 \end{aligned}$$

which is simply the same as a simple spring over the total length u with a coefficient,

$$k = \frac{k_1 k_2}{k_1 + k_2}. \quad (7.9)$$

Examining the problem case from the previous model (where a constant velocity is applied followed by no change in position) shows that our goal of reducing that jolting movement has been achieved (see Figure 7.4). The curve is smooth enough that the relaxation will be felt over time. Figure 7.5 shows a differently shaped displacement curve, which flows from the fact that only one half of the element is now damped. Finally, this model closely approximates measurements of real highly damped objects that have been taken [2].

7.2.3 Other Models

A virtually infinite number of models can be devised and used in a similar manner. Such models could involve more complicated arrangements of springs and dashpots, the use of more exotic primitives, or the inclusion of a mass term. The equations governing these other models do not even need to be first order ODEs. By approximating higher order derivatives, higher order ODEs can be numerically solved within this framework as well.

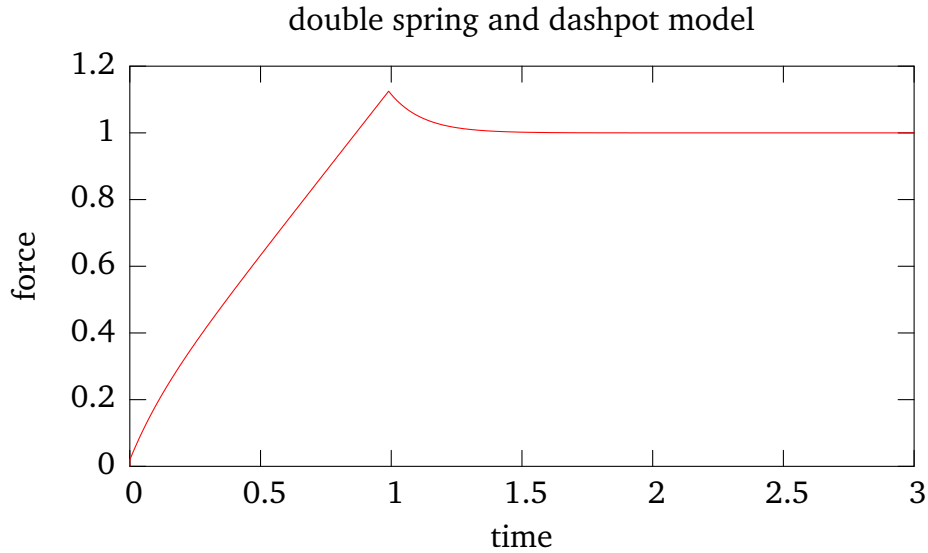


Figure 7.4: This graph shows the force produced from constant velocity ($t \in [0, 1]$, $\dot{u} = 1.0$) followed by rest using the two spring and one dashpot model. $k_1 = k_2 = 2.0$, $b = 0.5$, $h = 0.01$

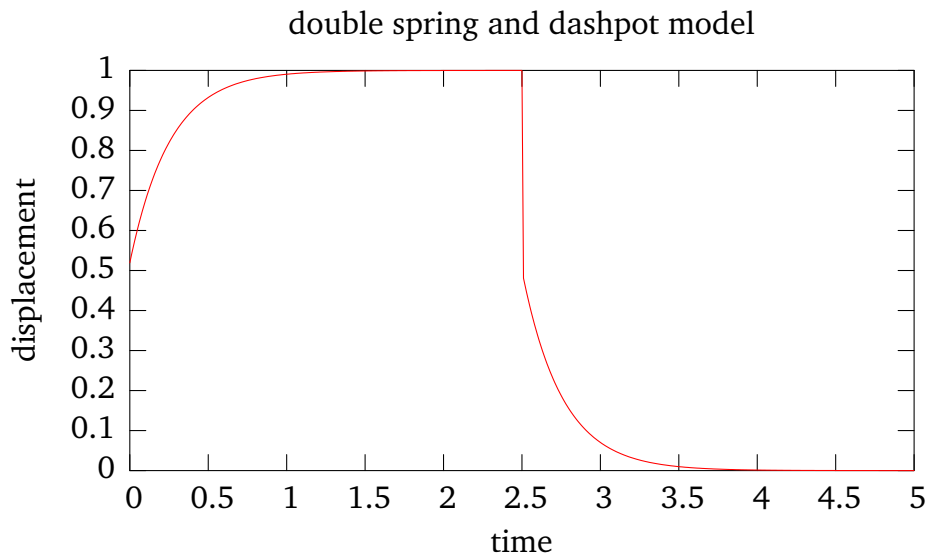


Figure 7.5: This graph shows the displacement produced from constant force ($t \in [0, 2.5]$, $f = 1.0$) followed by no force using the single spring and one dashpot model. $k_1 = k_2 = 2.0$, $b = 0.5$, $h = 0.01$

7.3 Extension into Three Dimensions

All the models discussed so far have been one dimensional to make their analysis simpler. Of course using these with a real DGFEM requires that they act in three dimensions. Moreover, the DGFEM entries are not one dimensional scalars (equivalent to spring coefficients), but in fact 3-by-3 matrices (equivalent to stiffness tensors). Next, the extension to three dimensions of the just mentioned improved damped model will be discussed.

7.3.1 Mathematical Description

Local 3D Model

Displacement in terms of force for the improved damped model (Equation 7.7) can be written in three dimensions for each individual element (i, j) as,

$$\begin{aligned} \mathbf{u}_{i,j}(t+h) &= \mathbf{K}_2^{-1} \mathbf{f}_j(t+h) + \mathbf{x}_{i,j}(t+h) \\ \mathbf{x}_{i,j}(t+h) &= \left(\mathbf{K}_{1(i,j)} + \frac{1}{h} \mathbf{B}_{i,j} \right)^{-1} \left(\mathbf{f}_j(t+h) + \frac{1}{h} \mathbf{B}_{i,j} \mathbf{x}_{i,j}(t) \right), \end{aligned} \quad (7.10)$$

and traction in terms of displacement (in 1D, Equation 7.8) as,

$$\begin{aligned} \mathbf{f}_{i,j}(t+h) &= \mathbf{K}_{2(i,j)} (\mathbf{u}_j(t+h) - \mathbf{x}_{i,j}(t+h)) \\ \mathbf{x}_{i,j}(t+h) &= \left(\mathbf{K}_{1(i,j)} + \mathbf{K}_{2(i,j)} + \frac{1}{h} \mathbf{B}_{i,j} \right)^{-1} \left(\mathbf{K}_{2(i,j)} \mathbf{u}_j(t+h) + \frac{1}{h} \mathbf{B}_{i,j} \mathbf{x}_{i,j}(t) \right) \end{aligned} \quad (7.11)$$

where the (i, j) subscripted quantities are the stiffness tensors ($\mathbf{K}_{1(i,j)}$ and $\mathbf{K}_{2(i,j)}$ corresponding to k_1 and k_2 in the 1D model, respectively), damping tensors ($\mathbf{B}_{i,j}$ corresponding to b), and state information ($\mathbf{x}_{i,j}(t)$) for the effect of node j on node i .

Global 3D Model

Equation 7.10 can be placed into the form of Equation 7.3 by adding a summation over j ,

$$\mathbf{u}_i(t+h) = \sum_j \mathbf{K}_{2(i,j)}^{-1} \mathbf{f}_j(t+h) + \mathbf{x}_{i,j}(t+h), \quad (7.12)$$

where $\mathbf{x}_{i,j}(t+h)$ is the same as in the local model. A global model for Equation 7.11 can be determined in a similar manner. However, this is not needed for implementation, so it will left out.

There are no traction or DGFM terms present in these equations, because traction can be couched in terms of force and stiffness tensors in terms a corresponding DGFM entry. Although the traction-force relationship ($\mathbf{f} = \alpha \mathbf{p}$) is clear and has been discussed, that of the stiffness-DGFM is not as straightforward. Combining a 3D extension of the one dimensional case presented in Equation 7.9 and Equation 7.1, the following must to hold to guarantee stiffness equivalence,

$$\alpha_j \Xi_{i,j}^{-1} = \mathbf{K}_{i,j} = (\mathbf{K}_{1(i,j)} + \mathbf{K}_{2(i,j)})^{-1} \mathbf{K}_{1(i,j)} \mathbf{K}_{2(i,j)}. \quad (7.13)$$

Clearly, there are many combinations of $\mathbf{K}_{1(i,j)}$ and $\mathbf{K}_{2(i,j)}$ that can satisfy this equation for a given $\Xi_{i,j}$. Thus, it is not completely possible to write Equation 7.12 in terms of traction and DGFM entries, without losing generality. The more qualitative meaning of these terms, as well as automatic ways of determining them will be discussed next.

7.3.2 Determining Stiffness Tensors from the DGFM

Although there are many possible solutions to Equation 7.13, there seems to be no advantage to having $\mathbf{K}_{1(i,j)}$ and $\mathbf{K}_{2(i,j)}$ acting in different directions or even in a direction different than that of the inverted DGFM entry. Furthermore, it should be mathematically simpler to only have to solve for tensors that are multiples of the inverted DGFM entry. For these reasons, only this specific case will be examined.

Perhaps the simplest way to find a valid solution is to look at the degenerate case where $\mathbf{K}_{1(i,j)}$ and $\mathbf{K}_{2(i,j)}$ are the equal to each other:

$$\begin{aligned}
 \bar{\mathbf{K}}_{i,j} &= \mathbf{K}_{1(i,j)} = \mathbf{K}_{2(i,j)} \\
 \alpha_j \boldsymbol{\Xi}_{i,j}^{-1} &= (\bar{\mathbf{K}}_{i,j} + \bar{\mathbf{K}}_{i,j})^{-1} \bar{\mathbf{K}}_{i,j} \bar{\mathbf{K}}_{i,j} \\
 &= (2\bar{\mathbf{K}}_{i,j})^{-1} \bar{\mathbf{K}}_{i,j}^2 \\
 &= \frac{1}{2} \bar{\mathbf{K}}_{i,j}^{-1} \bar{\mathbf{K}}_{i,j}^2 \\
 \bar{\mathbf{K}}_{i,j} = \bar{\mathbf{K}}_{1(i,j)} = \bar{\mathbf{K}}_{2(i,j)} &= 2\alpha_j \boldsymbol{\Xi}_{i,j}^{-1} \\
 \bar{\mathbf{K}}_{i,j}^{-1} = \bar{\mathbf{K}}_{1(i,j)}^{-1} = \bar{\mathbf{K}}_{2(i,j)}^{-1} &= \frac{1}{2\alpha_j} \boldsymbol{\Xi}_{i,j}.
 \end{aligned}$$

This approach produces stiffness tensors from the DGFM without any other parameters. In practice, this works fine in the case where behavior cannot (or does not need to) be tuned by a user. However, in other cases rough global control of behavior can be useful. To add another degree of control, a more general stiffness ratio, r can be defined.

$$\begin{aligned}
 \mathbf{K}_{1(i,j)} &= \bar{\mathbf{K}}_{i,j} \\
 \mathbf{K}_{2(i,j)} &= r\bar{\mathbf{K}}_{i,j} \\
 \alpha_j \boldsymbol{\Xi}_{i,j}^{-1} &= (\bar{\mathbf{K}}_{i,j} + r\bar{\mathbf{K}}_{i,j})^{-1} \bar{\mathbf{K}}_{i,j} r\bar{\mathbf{K}}_{i,j}
 \end{aligned}$$

$$\begin{aligned}
&= ((1+r)\bar{\mathbf{K}}_{i,j})^{-1} r\bar{\mathbf{K}}_{i,j}^2 \\
&= \frac{r}{1+r} \bar{\mathbf{K}}_{i,j}^{-1} \bar{\mathbf{K}}_{i,j}^2 \\
\bar{\mathbf{K}}_{i,j} = \bar{\mathbf{K}}_{1(i,j)} &= \frac{\alpha_j(1+r)}{r} \bar{\boldsymbol{\Xi}}_{i,j}^{-1} \\
\bar{\mathbf{K}}_{2(i,j)} &= \alpha_j(1+r) \bar{\boldsymbol{\Xi}}_{i,j}^{-1} \\
\bar{\mathbf{K}}_{i,j}^{-1} = \bar{\mathbf{K}}_{1(i,j)}^{-1} &= \frac{r}{\alpha_j(1+r)} \bar{\boldsymbol{\Xi}}_{i,j} \\
\bar{\mathbf{K}}_{2(i,j)}^{-1} &= \frac{1}{\alpha_j(1+r)} \bar{\boldsymbol{\Xi}}_{i,j}
\end{aligned}$$

It is instructive to talk about what this ratio actually means with regard to behavior. Because the force exerted on both springs must be the same, when the element is at rest the more pliable spring will stretch proportionally more than the stiffer spring. In the case where the springs are of the same stiffness, both springs will have equal length when the system is at rest.

A large ratio means that the non-damped spring will be stiffer than the damped spring and therefore occupy less of the length of the entire element; the opposite is true for a small ratio. The longer half of the element will proportionally dominate the overall behavior more. As can be seen for the one dimensional case in Figure 7.6, a larger ratio ($\frac{k_2}{k_1}$ in one dimension) results essentially in more damping-like behavior¹ when compression occurs with constant velocity. Similar effects can be seen in Figure 7.7 as well for the case where a constant force is applied for a time and then completely released. In three dimensions, increasing the ratio does not simply amplify the damping effects, but rather amplifies the damping effects *in the*

¹Damping-like behavior is mentioned here as opposed to actual damping behavior, because changing stiffness constants only changes the effects of damping through the inverted term of $\mathbf{x}_{i,j}(t+h)$, not in the direct damping against $\mathbf{x}_{i,j}(t)$. Therefore, actual damping is not completely affected.

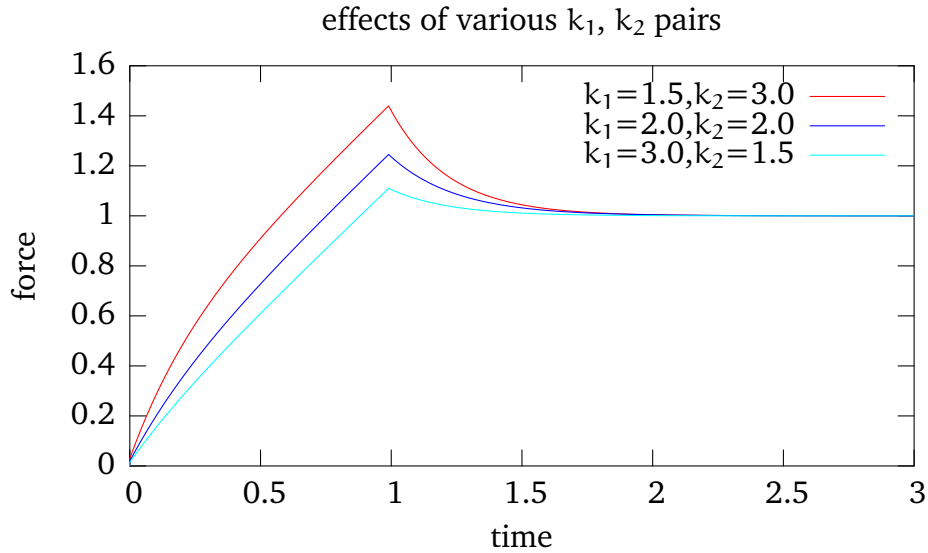


Figure 7.6: This graph shows the force produced from constant velocity ($t \in [0, 1]$, $\dot{u} = 1.0$) followed by 0 velocity using the double spring and one dashpot model with various values for k_1 and k_2 . $h = 0.01$, $b = 1.0$

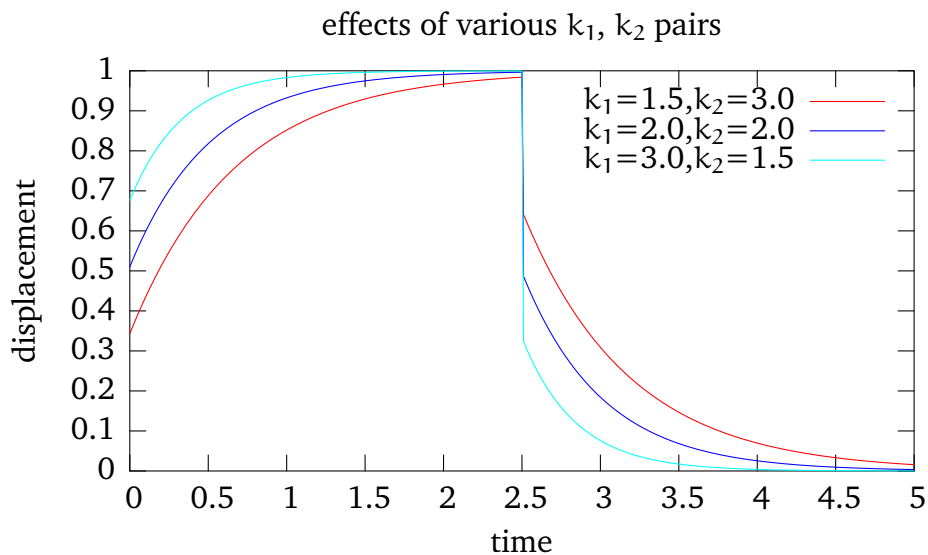


Figure 7.7: This graph shows the displacement produced from constant force ($t \in [0, 2.5]$, $f = 1.0$) followed by no force using the double spring and one dashpot model with various values for k_1 and k_2 . $h = 0.01$, $b = 1.0$

direction of the stiffness tensor. Any effects arising from the damping tensor ($\mathbf{B}_{i,j}$) occur separately in their respective direction.

7.3.3 Determining Damping Tensors

Unlike the stiffness tensors, there is no inherent relation that can be drawn between the damping tensors and the DGFm entries. These tensors can be set to any value and the model will still exhibit exponentially decaying damping. However, these values need to be carefully chosen to obtain realistic behavior. Here, Rayleigh damping is introduced as one simple way of handling this issue.

Rayleigh damping is the case, where the damping tensor can be described as a linear combination of mass and stiffness tensors. Because the model discussed here has no explicit mass term, Rayleigh damping means the damping tensor is nothing but a scaled version of the corresponding stiffness tensor,

$$\mathbf{B}_{i,j} = b\mathbf{K}_{1(i,j)},$$

where b is a damping scalar. In this case, Equation 7.12 can be rewritten as,

$$\begin{aligned} \mathbf{u}_i(t+h) &= \sum_j \mathbf{K}_{2(i,j)}^{-1} \mathbf{f}_j(t+h) + \mathbf{x}_{i,j}(t+h) & (7.14) \\ \mathbf{x}_{i,j}(t+h) &= \left(\mathbf{K}_{1(i,j)} + \frac{b}{h} \mathbf{K}_{1(i,j)} \right)^{-1} \left(\mathbf{f}_j(t+h) + \frac{b}{h} \mathbf{K}_{1(i,j)} \mathbf{x}_{i,j}(t) \right) \\ &= \left(\left(1 + \frac{b}{h} \right) \mathbf{K}_{1(i,j)} \right)^{-1} \left(\mathbf{f}_j(t+h) + \frac{b}{h} \mathbf{K}_{1(i,j)} \mathbf{x}_{i,j}(t) \right) \\ &= \frac{h}{h+b} \mathbf{K}_{1(i,j)}^{-1} \left(\mathbf{f}_j(t+h) + \frac{b}{h} \mathbf{K}_{1(i,j)} \mathbf{x}_{i,j}(t) \right) \\ &= \frac{h}{h+b} \mathbf{K}_{1(i,j)}^{-1} \mathbf{f}_j(t+h) + \frac{b}{h+b} \mathbf{x}_{i,j}(t). \end{aligned}$$

Equation 7.11 (needed for haptics) slightly simplifies to

$$\begin{aligned} \mathbf{f}_{i,j}(\mathbf{t} + \mathbf{h}) &= \mathbf{K}_{2(i,j)} (\mathbf{u}_j(\mathbf{t} + \mathbf{h}) - \mathbf{x}_{i,j}(\mathbf{t} + \mathbf{h})) \\ \mathbf{x}_{i,j}(\mathbf{t} + \mathbf{h}) &= \left(\frac{\mathbf{h} + \mathbf{b}}{\mathbf{h}} \mathbf{K}_{1(i,j)} + \mathbf{K}_{2(i,j)} \right)^{-1} \left(\mathbf{K}_{2(i,j)} \mathbf{u}_j(\mathbf{t} + \mathbf{h}) + \frac{\mathbf{b}}{\mathbf{h}} \mathbf{K}_{1(i,j)} \mathbf{x}_{i,j}(\mathbf{t}) \right). \end{aligned} \quad (7.15)$$

By using this assumption, global damping can be controlled by just this one scalar and still act reasonably.

7.3.4 Using Acquired Measurements

Work has been done recently on the acquisition of elastic, linear deformable models characterized by DGFMs (see Section 2.4 for a more general discussion on acquisition of deformable objects). In the course of work by Lang, Pai, and Woodham [2], measurements were taken, which are consistent with the improved viscoelastic model. The shape of the curves obtained from their measurements closely resemble the shape of the curve in Figure 7.4. Not only does this physically support the use of this model for the realistic simulation of a certain class of solid objects, but it also begs the question of whether the stiffness and damping tensors in the model can be acquired as well.

It certainly seems possible to fit this acquired data to ideal curves in order to approximate the stiffness and damping coefficients in one dimension. Assuming Rayleigh damping would allow a damping tensor to be determined from a combination of the acquired one dimensional damping coefficient and the acquired stiffness tensor. It may also be possible to fit the data into a more general damping tensor, as multiple measurements are taken at each point from different directions. However, because the data for the non-diagonal entries is limited in temporal precision, deter-

mining accurate damping tensors directly is difficult. While the robot arm is precise in measuring the traction being exerted on one of the vertices, the displacement of the other vertex must be measured visually using cameras with limited speed.

7.4 Optimizations

For each needed value in the summation (in total, the number of vertices in the mesh that are force-constrained squared), 3 matrix-vector multiplications, 1 matrix addition, 2 matrix inversions, 1 matrix-scalar division, and 2 vector additions are required. While just nine operations may not seem particularly expensive, it is important to remember that this step needs to be performed several times even for a simple mesh. For example, because the number of operations grows quadratically against the number of force-constrained vertices being used, with 100 of such vertices, this step would need to be done 10,000 times per time step. Taking into account the update rate necessary for decent visual display (30Hz), 2.7 million operations per second are needed. For haptic rates (1000Hz) the total number climbs to 90 million operations (20 million of which are matrix inversions which tend to be far more expensive than the other operations) per second. Because of the massive number of operations that need to be completed in a short amount of time, it would be desirable to find ways of reducing the total number through simplifications or precomputation.

7.4.1 Basic Precomputations

The most obvious way to reduce run-time computation is to have as much of it as possible done before the interactive loop begins. The key way to

recognize precomputable steps is to note which parts of Equation 7.12 can be assumed to be constant throughout the simulation loop.

The first possible precomputation eliminates one run-time matrix inversion and comes only at the expense of either requiring more memory (equal in size to the DGFM) or requiring an additional matrix-scalar division. So long as constant stiffness tensors are used, all $\mathbf{K}_{2(i,j)}$ quantities can be inverted beforehand and stored. Furthermore, as long as each tensor remains a scalar multiple of a constant matrix (typically, the DGFM), it is even possible to change them during the simulation without a high penalty. In this case, the constant matrix needs only to be inverted once in the precomputation stage. Because the inverse of that matrix multiplied by a scalar, $(s\mathbf{M})^{-1}$, is simply the inverted matrix multiplied by the inverse of the scalar: $\frac{1}{s}\mathbf{M}^{-1}$, the only penalty is that of a matrix-scalar division (9 scalar divisions as opposed to a 3-by-3 matrix inversion).

Unfortunately, eliminating the other matrix inversion, $(\mathbf{K}_{1(i,j)} + \frac{1}{h}\mathbf{B}_{i,j})^{-1}$ is only possible when the time step size, h , is constant. In the cases where the update rate can be held constant (or where such temporal inaccuracies do not significantly hinder perceived realism), this term for each (i, j) pair can be summed together, inverted, and stored during the precomputation phase.

Likewise, the $\frac{1}{h}\mathbf{B}_{i,j}$ term appearing towards the end of Equation 7.12 can be divided during the precomputation phase and stored in advance, if h is held constant. However, if one damping tensor is being used for all elements in the model, the term only needs to be computed once for each time step (regardless of the number of vertices), even if h can vary between time steps.

With Rayleigh damping, the last line of Equation 7.14 is rewritten here

with underbraced quantities as,

$$\mathbf{x}_{i,j}(t + h) = \underbrace{\frac{h}{h + b}}_{c_1} \mathbf{K}_{1(i,j)}^{-1} \mathbf{f}_j(t + h) + \underbrace{\frac{b}{h + b}}_{c_2} \mathbf{x}_{i,j}(t).$$

The $\mathbf{K}_{1(i,j)}$ terms can simply be inverted during precomputation. Because h and b are kept constant at least during a time step, c_1 and c_2 can also be precomputed at the beginning of the loop. Additionally, if h and b are constant throughout the entire simulation, the two divisions can be precomputed just once before the simulation loop begins.

Precomputation of parts of Equation 7.11 and its Rayleigh damping equivalent Equation 7.15 are not in general as time saving. For both equations, the inversion can only be saved if h (and b in the case of the latter one) is constant. In this case only two matrix-vector multiplications at runtime would be necessary. For the general case, summing $\mathbf{K}_{1(i,j)}$ and $\mathbf{K}_{2(i,j)}$ during precomputation is possible when h can vary, but will not result in any large scale time savings. Furthermore, because this is a local calculation, typically it only needs to be performed for a small number of vertices (and not for all vertices as with the global calculation). This has the effect that any precomputation is not going to make as large of an impact as in the other equations. Finally, because only the case where $i = j$ is needed for the haptics purposes of this thesis, the number of elements, which need to be stored for precomputation is only linear (as opposed to quadratic as in the other equations) in the number of vertices.

7.4.2 Taking Advantage of the Zero Force Case

During the course of simulation for haptic and display purposes, typically only a few non-zero valued forces will act on the object. If the common case, where forces being applied are zero, can be computed more quickly, savings to computation time may be quite substantial. Substituting the zero vector into $\mathbf{f}_j(t + h)$ of Equation 7.10 for this case and separating Equation 7.12 gives,

$$\begin{aligned} \mathbf{u}_i(t + h) &= \sum_j \mathbf{x}_{i,j}(t + h) + \begin{cases} \mathbf{K}_{2(i,j)}^{-1} \mathbf{f}_j(t + h), & \mathbf{f}_j(t + h) \neq \mathbf{0} \\ \mathbf{0} & \mathbf{f}_j(t + h) = \mathbf{0} \end{cases} \\ \mathbf{x}_{i,j}(t + h) &= \underbrace{\left(\mathbf{K}_{1(i,j)} + \frac{1}{h} \mathbf{B}_{i,j} \right)^{-1}}_{\mathbf{Q}_{i,j}} \underbrace{\left(\frac{1}{h} \mathbf{B}_{i,j} \mathbf{x}_{i,j}(t) + \right.}_{\mathbf{R}_{i,j}} \\ &\quad \left. + \begin{cases} \mathbf{f}_j(t + h), & \mathbf{f}_j(t + h) \neq \mathbf{0} \\ \mathbf{0} & \mathbf{f}_j(t + h) = \mathbf{0} \end{cases} \right) \end{aligned}$$

Even in the case where there are no special conditions on the stiffness or damping tensors, this reduction is still cheaper computationally than using the full formula (only one matrix inversion, one matrix addition, one matrix multiplication, one matrix-vector multiplication, and two scalar divisions). If h is held constant, each $\mathbf{R}_{i,j}$ and $\mathbf{Q}_{i,j}$ can be precomputed, resulting in just one necessary matrix-vector multiplication in the zero case and two in the non-zero case.

With Rayleigh damping the situation is even rosier. Equation 7.14 be-

comes,

$$\begin{aligned} \mathbf{u}_i(t+h) &= \sum_j \mathbf{x}_{i,j}(t+h) + \begin{cases} \mathbf{K}_{2(i,j)}^{-1} \mathbf{f}_j(t+h), & \mathbf{f}_j(t+h) \neq \mathbf{0} \\ \mathbf{0} & \mathbf{f}_j(t+h) = \mathbf{0} \end{cases} \\ \mathbf{x}_{i,j}(t+h) &= \underbrace{\frac{b}{h+b}}_{c_2} \mathbf{x}_{i,j}(t) + \begin{cases} \underbrace{\frac{h}{h+b}}_{c_1} \mathbf{K}_{1(i,j)}^{-1} \mathbf{f}_j(t+h), & \mathbf{f}_j(t+h) \neq \mathbf{0} \\ \mathbf{0}, & \mathbf{f}_j(t+h) = \mathbf{0} \end{cases} \end{aligned}$$

As before, c_2 needs only be computed once at the beginning of the time step. All subsequent computations only require one vector-scalar multiplication, when the force acting on that element is zero.

7.5 Haptic Simulation

Simulating haptics with the viscoelastic model is very similar to how it is done in the elastic model, although much of the work done in the display loop must now be done in the haptic loop to maintain the correct system state. The notation used then in Section 6.3 will also be used here. The modified process for the haptic loop follows.

1. Determine if contact with the object is occurring. If not, nothing more needs to be done and the remaining steps can be skipped.

In the case of a point contact model, collision occurs when the proxy has penetrated the object's surface. However, this is not as simple to do as with an elastic model, because the undeformed state of the object cannot be used here to detect collision. Quantities \mathbf{u} and \mathbf{u}_{SCP} need to reflect the collisions occurring on this deformed surface.

2. Compute the stiffness through use of the equation for determining

local force from displacement and collision details.

- (a) For each $\mathbf{v}_n \in S_t$, \mathbf{u} is used as \mathbf{u}_n and substituted into Equation 7.11 to obtain $\mathbf{f}'_n := \mathbf{f}_{n,n}(t + h)$.

For each $\mathbf{v}_m \in S_d$, follow the same procedure as for the elastic case.

- (b) These forces and coefficients are then averaged together appropriately with their corresponding barycentric coordinate values (β_i) as weights to produce an interpolated stiffness coefficient at the SCP

$$k_{\text{SCP}} = \sum_{\mathbf{v}_i \in S_t} \beta_i \frac{|\mathbf{f}'_i(t + h)|}{|\mathbf{u}|} + \sum_{\mathbf{v}_i \in S_d} \beta_i k_i.$$

3. Set the current stiffness coefficient as determined in step 2.

4. Let the overall force vector be defined as,

$$\forall i, i \in \{0, 1, \dots, n-1, n\}, \mathbf{f}_i = \begin{cases} \beta_i \mathbf{f}'_i & \text{if } \mathbf{f}'_i \text{ exists} \\ 0 & \text{otherwise} \end{cases}$$

5. Calculate the resulting deformation. Use equation 7.12 (or corresponding optimization from Section 7.4) to compute all \mathbf{u}_i as well as $\mathbf{x}_{i,j}$.

Because so much must now be done at every haptic time step to update the state of the deformed object, the display loop consists now of only one step:

1. Draw all polygons in the mesh using the displaced locations of all vertices obtained by adding the corresponding deformation vectors and undeformed vertex locations.

The theory presented in this chapter provides the foundation for a actual simulation of viscoelastic solids. In the next chapter, details of how this material fits into a real haptic and visual system will be given.

Chapter 8

Implementation Details

The above mentioned approaches have been combined into an interactive software system. Empirical experiments demonstrated that this system was effective at modeling viscoelasticity in an efficient manner. For objects that are simple enough to be adequately haptically manipulated and viewed, adding viscoelasticity proved to consume bearable amounts of extra runtime computation and precomputation.

8.1 Object Models

For the experiments, two specific object models were used:

“hemisphere” model Its mesh and various DGFMs (with different material properties) were purely computer generated. The flat bottom of the mesh consists solely of fixed points. It represents one of the simplest shapes for haptic interaction. See figure 8.1.

“tiger” model Its mesh was acquired through physical measurements of a tiger plush toy. Several DGFMs generated through BEM methods were

| model name | “hemisphere” | “tiger” |
|----------------------------------|----------------|---------------|
| Properties | | |
| # of vertices = # fixed + # free | 162 = 111 + 51 | 149 = 54 + 95 |
| # of faces | 320 | 294 |
| # of edges | 480 | 441 |
| BEM derived DGFMs | yes | yes |
| physically acquired DGFMs | no | yes |

Table 8.1: Overview of properties of the two models used.

available. Additionally, a DGFm physically acquired using the ACME Facility at the University of British Columbia [55] was available. In all of these DGFMs, the belly and paws of the tiger have been fixed in place by displacement constraints. See figure 8.2.

8.2 Supporting Hardware

SensAble’s™ PHANToM® Desktop [53] was used with all software produced for this thesis. Specifications for the device are given in Table 8.2. The main workstation used for development and testing contained a 2.8 GHz Intel Pentium 4 processor along with an ATI FireGL™ X1 graphics card and ran Debian GNU/Linux 3.0.

8.3 Supporting Software

Most pieces of the systems that did not need to run fast (precomputation, loading and parsing input files, user interface) were written in Python [56], while time critical code (GL drawing call, deformation computation) was written in C (in most cases) or C++ (in code that directly interfaced with the haptic device). The Python extension PyOpenGL [57] was used for

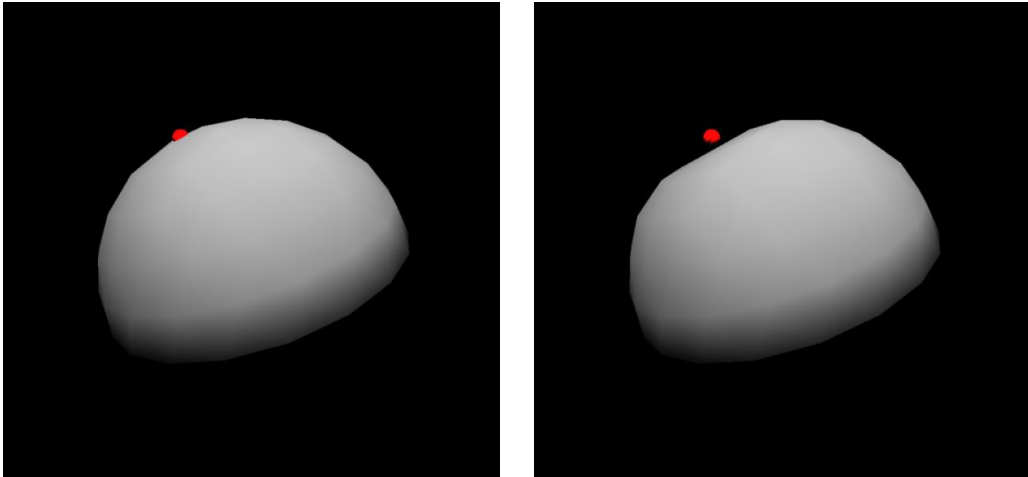


Figure 8.1: “hemisphere” model



Figure 8.2: “tiger” model

| | |
|--------------------------------|---------------------------------|
| workspace | 16 x 13 x 13 cm |
| range of motion | hand movement pivoting at wrist |
| nominal position resolution | 0.02 mm |
| backdrive friction | 0.06 N |
| maximum exertable force | 6.4 N |
| continuous exertable force | 1.7 N |
| stiffness | ≈ 3.16 N/mm |
| inertia (apparent mass at tip) | < 75 g |
| footprint | 18 x 16 cm |
| force feedback | x, y, z |
| position sensing | x,y,z, pitch, roll, yaw |
| supported platforms | Intel-based PCs |

Table 8.2: PHANTOM® Desktop specifications. Adapted from SensAble's™ product documentation.

many OpenGL and GLUT calls occurring outside of the main program loop. Another extension known as Numeric Python [58] provided an array data structure that was efficient and easily accessible in both Python and C. LAPACK (linear algebra package) [59] routines along with optimized BLAS (basic linear algebra subprograms) routines from ATLAS (automatically tuned linear algebra software) [60] were used in C through Fortran calling conventions and more indirectly in Python through routines provided by Numeric Python. Finally, SensAble's™ C++ GHOST SDK [53] was used for interacting with the haptic device.

8.4 Results

For the implementation of the concepts discussed in this thesis two pieces of software were written. First, an interactive linear, elastic simulator was written to provide a starting point for experimenting with more advanced techniques. For both models mentioned earlier in this section, the linear elastic system performed at reasonable haptic and display rates. However,



Figure 8.3: Viscoelastic software in use with haptics.

| model name | “hemisphere” | “tiger” |
|---|--------------|-----------|
| Results | | |
| viscoelastic display rate without haptics ¹ | 425–670Hz | 435–700Hz |
| viscoelastic display rate with elastic haptics ² | 100Hz | 100Hz |
| elastic haptic rate sustainable ³ | yes | yes |

Table 8.3: Timing results of the two models.

using a mesh and DGFM derived through one level of subdivision was not haptically sustainable.

The software was then further extended into another interactive system that implements most of the viscoelastic approach described in Chapter 7. Unfortunately, only the visual portion of the viscoelastic model was fully implemented. Limitations on the built in collision detection of the GHOST SDK prevented full implementation of the haptic portion. Essentially, the

¹Ranges are given, because the rate varies with the number of traction constraints acting on the object at a certain point in time.

²Idiosyncrasies of the GHOST SDK restrict the display update to no higher than 100Hz.

³The GHOST SDK’s reporting of the actual haptic rate is somewhat unreliable. However, if the rate drops below 1000Hz, thus becoming unsustainable, the haptic loop is interrupted and notification is given.

automatic collision detection on polygonal meshes does not feasibly work when the mesh must rapidly deform. Thus, the software system used a viscoelastic display loop and elastic haptic loop instead. Despite this compromise, it is difficult for the user to detect the presence of the different models. Nonetheless, the high visual rates indicate that running the necessary calculations inside the haptic loop are feasible. While the 1000Hz rate is not quite reached for these models, updating the surface spring constant only on every second or third time step still provides an approximate force updated on every cycle. A summary of these rates can be found in Table 8.3.

Chapter 9

Conclusion and Future Work

This thesis has presented the background, theoretical underpinnings, and implementation details of a framework for viscoelastic models. This framework combines the underlying global behavior of a DGFM with the realism of particle systems. Additionally, a specific model based on physical measurements has been devised as an example within this framework. This model is flexible in that it is controlled through as few as one parameter (damping) or as many as thousands (all individual stiffness and damping tensors). The approach is capable of interactive rates for haptics and display with approximate collision detection.

With regard to future work, a number of directions exist. The first would be adding specialized collision detection for a deforming surface, which would allow the viscoelastic model presented to be fully functional also within the haptics loop itself. As touched on in Section 7.3.4, further work could be done on performing automated acquisition of damping tensors. Truly investigating this aspect requires access to data sets for several models. It may also be interesting to examine other replacements for the linear elastic equations of a DGFM. Other models may be useful for cer-

tain types of materials. Adding mass terms, for example, would allow for oscillating motion. In the name of low computational cost, only first order derivative approximations were used. However, it may be worth investigating whether the cost of higher order approximations is offset by the ability to take larger time steps due to improved accuracy of higher order solution methods. In this same vein, methods for adapting time step size to improve efficiency may be useful in that respect as well. Finally, it may be possible to take advantage of the GPU (graphics processing unit) in modern video hardware to partially shift simulation cost off the main processor. With regard to haptics, this is probably not as likely to be successful, but for graphical display purposes such work could improve the speed of the simulation.

Bibliography

- [1] Yuan-Cheng Fung. *Biomechanics: Mechanical Properties of Living Tissues*. Springer-Verlag, New York, second edition, 1993. 1
- [2] Jochen Lang, Dinesh K. Pai, and Robert J. Woodham. Acquisition of elastic models for interactive simulation. *The International Journal of Robotics Research*, 21(8):713–733, August 2002. 1, 2.4, 5.3, 7.2.2, 7.3.4
- [3] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *Proceedings of ACM SIGGRAPH*, Anaheim, July 1987. ACM Press. 2
- [4] Alex Pentland and John Williams. Good vibrations: Modal dynamics for graphics and animation. In *Proceedings of ACM SIGGRAPH*, pages 215–222, Boston, July 1989. ACM Press. 2, 2.2.3
- [5] Sarah F. Gibson and Brian Mirtich. A survey of deformable modeling in computer graphics. Technical Report TR-97-19, Mitsubishi Electric Research Laboratory, November 1997. 2
- [6] Stephen M. Platt and Norman I. Badler. Animating facial expressions. In *Proceedings of ACM SIGGRAPH*, pages 245–252. ACM Press, 1981. 2.1
- [7] William T. Reeves. Particle systems—a technique for modeling a class of fuzzy objects. In *Proceedings of SIGGRAPH*, pages 359–375. ACM Press, 1983. 2.1
- [8] Michel Carignan, Ying Yang, Nadia Magnenat Thalmann, and Daniel Thalmann. Dressing animated synthetic actors with complex deformable clothes. In *Proceedings of ACM SIGGRAPH*, pages 99–104. ACM Press, 1992. 2.1

- [9] David E. Breen, Donald H. House, and Michael J. Wozny. Predicting the drape of woven cloth using interacting particles. In *Proceedings of ACM SIGGRAPH*, pages 365–372. ACM Press, 1994. 2.1
- [10] Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Proceedings of Graphics Interface*, pages 147–154, 1995. 2.1
- [11] B. Eberhardt, A. Weber, and W. Strasser. A fast, flexible particle-system model for cloth draping. *IEEE Computer Graphics and Applications*, 16(5):52–59, September 1996. 2.1
- [12] David Baraff, Andrew Witkin, and Michael Kass. Untangling cloth. *ACM Transactions on Graphics (TOG)*, 22(3):862–870, 2003. 2.1
- [13] J. Groß, M. Hauth, O. Eitzmuß, and G. F. Bueß. Modelling viscoelasticity in soft tissues. In *Proceedings of the Int. Workshop on Deformable Modelling and Soft Tissue Simulation*, November 2001. 2.1, 3.3.1
- [14] J. Groß and G. Bueß. VR models for surgical training with realistic biophysical properties. In H. U. Lemke, M. W. Vannier, K. Inamura, A. G. Farman, and K. Doi, editors, *Proceedings of CARS*, pages 1008–1013. Springer Verlag, 2001. 2.1, 3.3.1
- [15] Allan van Gelder. Approximate simulation of elastic membranes by triangulated spring meshes. *Journal of Graphics Tools*, 3(2):21–41, 1998. 2.1
- [16] David Baraff and Andrew Witkin. Physically based modeling. In *SIGGRAPH Course Notes*. ACM Siggraph, 2003. 2.1, 3
- [17] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proceedings of SIGGRAPH*, pages 43–54. ACM Press, 1998. 2.1, 3.3.1
- [18] Michael Hauth and Olaf Eitzmuss. A high performance solver for the animation of deformable objects using advanced numerical methods. In *Proceedings of Eurographics*. Eurographics Association and Blackwell Publishers, 2001. 2.1
- [19] Gilbert Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, Wellesley, MA, 1986. 2.2.1, 4.1
- [20] Hans R. Schwarz. *Methode der finiten Elemente : eine Einführung unter besonderer Berücksichtigung der Rechenpraxis*, volume 47 of *Leitfäden*

- der angewandten Mathematik und Mechanik*) ; (Teubner Studienbücher Mathematik. Teubner, 3. neubearbeitete auflage edition, 1993. 2.2.1
- [21] James F. O'Brien and Jessica K. Hodgins. Graphical modeling and animation of brittle fracture. In *Proceedings of ACM SIGGRAPH*, pages 137–146. ACM Press/Addison-Wesley Publishing Co., 1999. 2.2.1
- [22] James F. O'Brien, Adam W. Bargteil, and Jessica K. Hodgins. Graphical modeling and animation of ductile fracture. In *Proceedings of ACM SIGGRAPH*, pages 291–294. ACM Press, 2002. 2.2.1
- [23] 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, 1996. 2.2.1
- [24] Yan Zhuang. *Real-time Simulation of Physically Realistic Global Deformations*. PhD thesis, University of California, Berkeley, 2000. 2.2.1
- [25] Yan Zhuang and John Canny. Haptic interaction with global deformations. In *Proceedings of the IEEE International Conference on Robotics and Automations (ICRA00)*, pages 24–28, San Francisco, Apr 2000. 2.2.1
- [26] Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H. Barr. Dynamic real-time deformations using space & time adaptive sampling. In *Proceedings of SIGGRAPH*, pages 31–36. ACM Press, 2001. 2.2.1
- [27] Carlos Alberto Brebbia, Jose C. F. Telles, and Luis C. Wrobel. *Boundary Element Techniques: Theory and Applications in Engineering*. Springer, New York, second edition, 1984. 2.2.2, 5.2, 5.2.1
- [28] Carlos Alberto Brebbia and S. Walker. *Boundary Element Techniques in Engineering*. Newnes-Butterworths, London, 1980. 2.2.2
- [29] Doug L. James and Dinesh K. Pai. ArtDefo - accurate real time deformable objects. In *Proceedings of ACM SIGGRAPH*, pages 65–72, Los Angeles, August 1999. ACM Press. 2.2.2, 5.2, 5.2.1
- [30] Doug L. James and Dinesh K. Pai. A unified treatment of elastostatic and rigid contact simulation for real time haptics. *Haptics-e*, 2(1), September 2001. 2.2.2, 5, 5.2
- [31] N. Zhang. Dynamic condensation of mass and stiffness matrices. *Journal of Sound and Vibration*, 188(4):601–615, 1995. 2.2.3

- [32] Kris K. Hauser, Chen Shen, and James F. O'Brien. Interactive deformation using modal analysis with constraints. In *Proceedings of Graphics Interface*, pages 247–256. A K Peters, June 2003. 2.2.3
- [33] James F. O'Brien, Perry R. Cook, and Georg Essl. Synthesizing sounds from physically based motion. In *Proceedings of ACM SIGGRAPH*, pages 529–536. ACM Press, 2001. 2.2.3
- [34] Doug L. James and Dinesh K. Pai. DyRT: Dynamic response textures for real time deformation simulation with graphics hardware. In *Proceedings of ACM SIGGRAPH*, pages 582–585, San Antonio, July 2002. ACM Press. 2.2.3
- [35] Gagatay Basdogan. Real-time simulation of dynamically deformable finite element models using modal analysis and spectral lanczos decomposition methods. In *Proceedings of the Medicine Meet Virtual Reality (MMVR) Conference*, Irvine, January 2001. 2.2.3
- [36] Craig B. Zilles and J. K. Salisbury. A constraint-based god-object method for haptic display. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 1995. 2.3
- [37] Diego C. Ruspini, Krasimir Kolarov, and Oussama Khatib. The haptic display of complex graphical environments. In *Proceedings of SIGGRAPH*, pages 345–352. ACM Press/Addison-Wesley Publishing Co., 1997. 2.3
- [38] Federico Barbagli, Kenneth Salisbury, and Domenico Prattichizzo. Dynamic local models for stable multi-contact haptic interaction with deformable objects. In *Proceedings of the Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 109–116, Los Angeles, March 2003. 2.3
- [39] G. Szekély, C. Brechbuhler, J. Dual, R. Enzler, R. Hutter J. Hug, N. Ironmonger, M. Kauer, V. Meier, P. Niederer, A. Rhomberg, P Schmid, G. Schweitzer, M. Thaler, V. Vuskovic, G. Troster, U. Haller, and M. Bajka. Virtual reality-based simulation of endoscopic surgery. *Presence*, 9(3):313–333, 2000. 2.3
- [40] D. d'Aulignac, M.C. Cavusoglu, and C. Laugier. A haptic interface for a virtual exam of the human thigh. In *Proceedings of the International Conference on Robotics and Automation*, pages 2452–2457, San Francisco, April 2000. 2.3

- [41] G. Picinbono, H. Delingette, and N. Ayache. Non-linear and anisotropic elastic soft tissue models for medical simulation. In *Proceedings of International Conference on Robotics and Automation*, pages 1371–1376, Seoul, May 2001. 2.3
- [42] D. d’Aulignac, M.C. Cavusoglu, and C. Laugier. Modeling the human thigh for a realistic echographic simulator with force feedback. In *Proceedings of the International Conference on Medical Image Computer-Assisted Intervention*, pages 1191–1198, Cambridge, United Kingdom, September 1999. 2.4
- [43] Iman Brouwer, Jeffrey Ustin, Loren Bentley, Alana Sherman, Neel Dhruv, and Frank Tendick. Measuring in vivo animal soft tissue properties for haptic modeling in surgical simulation. *Medicine Meets Virtual Reality*, pages 69–74, 2001. 2.4
- [44] V. Vuskovic, M. Kauer, G. Székely, and M. Reidy. Realistic force feedback for virtual reality based diagnostic surgery simulators. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1592–1598, San Francisco, 2000. 2.4
- [45] H.D. Hoeg, A.B. Slatkin, J.W. Burdick, and W.S. Grundfest. Biomechanical modeling of the small intestine as required for the design and operation of a robotic endoscope. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1599–1606, San Francisco, 2000. 2.4
- [46] U. Kuhnappel, H.K. Cakmak, and H. Maa. Endoscopic surgery training using virtual reality and deformable tissue simulation. *Computers & Graphics*, 24:671–682, 2000. 2.4
- [47] Dinesh K. Pai, Kees van den Doel, Doug L. James, Jochen Lang, John E. Lloyd, Joshua L. Richmond, and Som H. Yau. Scanning physical interaction behavior of 3d objects. In *Proceedings of SIGGRAPH*, pages 87–96. ACM Press, 2001. 2.4, 5.4
- [48] Hans R. Schwarz. *Numerische Mathematik*. Teubner, 4. überarb. und erweiterte auflage edition, 1997. 4.2
- [49] Uri M. Ascher and Linda R. Petzold. *Computer methods for ordinary differential equations and differential-algebraic equations*. SIAM, Philadelphia, 1998. 4.2

- [50] Eric Weistein's World of Physics. <http://scienceworld.wolfram.com/physics/>. 5.2.1
- [51] Charles T. Loop. Smooth subdivision surfaces based on triangles, 1987. Master's thesis, University of Utah, Department of Mathematics. 5.4
- [52] *The American Heritage Dictionary of the English Language*. Houghton Mifflin, fourth edition, 2000. 6
- [53] SensAble Technologies, Inc. <http://www.sensable.com>. 6, 8.2, 8.3
- [54] E. Colgate, P. Grafing, M. Stanley, and G. Schenkel. Implementation of stiff virtual walls in force-reflecting interfaces. In *Proceedings of VRAIS*, pages 202–208, Seattle, September 1993. 6.1
- [55] Active Measurement (ACME) Facility, University of British Columbia. <http://www.cs.ubc.ca/nest/lci/acme>. 8.1
- [56] Python 2.2.3. <http://www.python.org>. 8.3
- [57] PyOpenGL 2.0. <http://pyopengl.sourceforge.net>. 8.3
- [58] David Ascher, Paul F. Dubouis, Konrad Hinsen, Jim Hugunin, and Travis Oliphant. Numerical python users' guide. Technical Report UCRL-MA-128569, Lawrence Livermore National Laboratory, 2001. 8.3
- [59] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. 8.3
- [60] R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001. 8.3