TOWARDS REAL-TIME SIMULATION OF HYPERELASTIC MATERIALS

Tiantian Liu

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2018

Ladislav Kavan, Adjunct Assistant Professor
Computer and Information Science
Supervisor of Dissertation

Lyle Ungar, Professor
Computer and Information Science
Graduate Group Chairperson

Dissertation Committee

Norman I. Badler, Professor
Computer and Information Science
University of Pennsylvania

Camillo J. Taylor, Professor
Computer and Information Science
University of Pennsylvania

Chenfanfu Jiang, Assistant Professor
Computer and Information Science
University of Pennsylvania

Eftychios Sifakis, Assistant Professor
Computer Science
University of Wisconsin-Madison

TOWARDS REAL-TIME SIMULATION OF HYPERELASTIC MATERIALS

2018

Tiantian Liu

To Xi Zhang, my beloved wife,

because you always understand;

and to Ruochong E. Liu, our lovely daughter,

even it is too early for you to understand.

Acknowledgements

I would like to first thank Prof. Norman Badler and Prof. Stephen Lane for introducing computer graphics to my life when I was a master student at Penn. Without you, I could never find my passion in computer graphics research.

Many thanks to my friend Joe Kider and Qing Sun. Though we did not make it for our SIGGRAPH Asia submission, I really enjoyed the experience of my first research work on computer graphics with you.

My sincerer thanks to Prof. Adam Bargteil, Prof. James O'Brien and Prof. Eftychios Sifakis for all the insightful discussions. It would be impossible to finish the quality research projects without your wise vision.

I can not thank Mike Felker and Amy Calhoun more for their kindness. You make me feel someone is always taking care of me in person.

Last but not least, I would like to express my greatest gratitude to my advisor Prof. Ladislav Kavan who is both a best friend and a role model. We share not only the excitement of our research but the happiness of our lives as well. It is hard to image to work with anyone else than you as my mentor.

ABSTRACT

TOWARDS REAL-TIME SIMULATION OF HYPERELASTIC MATERIALS

Tiantian Liu

Ladislav Kavan

We propose a new method for physics-based simulation supporting many different types of hyperelastic materials from mass-spring systems to three-dimensional finite element models, pushing the performance of the simulation towards real-time. Fast simulation methods such as Position Based Dynamics exist, but support only limited selection of materials; even classical materials such as corotated linear elasticity and Neo-Hookean elasticity are not supported. Simulation of these types of materials currently relies on Newton's method, which is slow, even with only one iteration per timestep. In this work, we start from simple material models such as mass-spring systems or as-rigid-as-possible materials. We express the widely used implicit Euler time integration as an energy minimization problem and introduce auxiliary projection variables as extra unknowns. After our reformulation, the minimization problem becomes linear in the node positions, while all the non-linear terms are isolated in individual elements. We then extend this idea to efficiently simulate a more general spatial discretization using finite element method. We show that our reformulation can be interpreted as a quasi-Newton method. This insight enables very efficient simulation of a large class of hyperelastic materials. The quasi-Newton interpretation also allows us to leverage ideas from numerical optimization. In particular, we show that our solver can be further accelerated using L-BFGS updates (Limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm). Our final method is typically more than ten times faster than one iteration of Newton's method without compromising quality. In fact, our result is often more accurate than the result obtained with one iteration of Newton's method. Our method is also easier to implement, implying reduced software development costs.

Contents

List of Figures

Chapter 1

Introduction

Physics-based animation is an important tool in computer graphics even though creating visually compelling simulations often requires a lot of patience. Waiting for results is not an option in real-time simulations, which are necessary in applications such as computer games and training simulators, e.g., surgery simulators. Previous methods for real-time physics such as Position Based Dynamics [Müller et al., 2007] have been successfully used in many applications and commercial products, despite the fact that these methods support only a restricted set of material models.

The advantages of more general material models were nicely demonstrated in the work of Xu et al. [2015], who proposed a new class of spline-based materials particularly suitable for physics-based animation. Their user-friendly spline interface enables artists to easily modify material properties in order to achieve desired animation effects. However, their system relies on Newton's method, which is slow, even if the number of Newton's iterations per frame is limited to one. Our method enables fast simulation of general hyperelastic materials including hte spline-based materials, combining the benefits of artist-friendly material interfaces with the advantages of fast simulation, such as rapid iterations and/or higher resolutions.

In order to simulate the motion of deformable objects in time, we usually start from a backward Euler time integration which can be formulated as an optimization problem

where we minimize a multi-variate function g. Newton's method is commonly treated as the state-of-art numerical solution to minimize g. It minimizes g by performing descent along direction $-(\nabla^2 g)^{-1}\nabla g$, where $\nabla^2 g$ is the Hessian matrix, and ∇g is the gradient. One problem with Newton's method is that the Hessian $\nabla^2 g$ can be indefinite, in which case Newton's direction could erroneously *increase* g. This undesired behavior can be prevented by so-called "definiteness fixes" [Nocedal and Wright, 2006, Teran et al., 2005]. While definiteness fixes require some computational overheads, the slow speed of Newton's method is mainly caused by the fact that the Hessian changes at every iteration, i.e., we need to solve a *new* linear system for every Newton step.

The point of departure for our method is a quasi-Newton approach to minimize the multi-variate function g. In general, quasi-Newton methods [Nocedal and Wright, 2006] work by replacing the Hessian $\nabla^2 g$ with a linear operator $\mathbf{A}$, where $\mathbf{A}$ is positive definite and solving linear systems $\mathbf{A}\mathbf{x} = \mathbf{b}$ is fast. The descent directions are then computed as $-\mathbf{A}^{-1}\nabla g$ (where the inverse is of course not explicitly evaluated, in fact, $\mathbf{A}$ is often not even represented with a matrix). The trade-off is that if $\mathbf{A}$ is a poor approximation of the Hessian, the quasi-Newton method may converge slowly. Unfortunately, coming up with an effective approximation of the Hessian is not easy. We tried many previous quasi-Newton methods, but even after boosting their performance with L-BFGS [Nocedal and Wright, 2006], we were unable to obtain an effective method for real-time physics. We start by accelerating models with simple quadratic energy functions such as mass-spring system and some finite element modes using local-global optimization methods. We show that this kind of acceleration can be re-formulated as a quasi-Newton method with some remarkable properties, in particular, the resulting $\mathbf{A}_{\texttt{our}}$ matrix is *constant* and *positive definite*. This re-formulation enables us to generalize the method to simulate more general hyperelastic materials, such as the Neo-Hookean or spline-based materials.

The quasi-Newton formulation also allows us to further improve convergence of our solver. We propose using L-BFGS, which uses curvature information estimated from a

certain number of previous iterates to improve the accuracy of our Hessian approximation $\mathbf{A}_{\texttt{our}}$. Adding the L-BFGS Hessian updates introduces only a small computational overhead while accelerating the convergence of our method. However, the performance of L-BFGS highly depends on the quality of the initial Hessian approximation. With previous quasi-Newton methods, we observed rather disappointing convergence properties. The combination of our Hessian approximation $\mathbf{A}_{\texttt{our}}$ with L-BFGS is quite effective.

The L-BFGS convergence boosting is compatible with our first contribution, i.e., fast simulation of complex non-linear materials. Specifically, we can simulate any materials satisfying the Valanis-Landel assumption [Valanis and Landel, 1967] which includes many classical materials, such as St. Venant-Kirchhoff, Neo-Hookean, Mooney-Rivlin, and also the recently proposed spline-based materials [Xu et al., 2015]. In summary, our final method achieves fast convergence while being able to simulate a large variety of hyperelastic materials.

Chapter 2

Related Work

2.1 Material Models

The work of Terzopoulos et al. [1987] pioneered physically based animation, an indispensable tool in feature animation and visual effects nowadays. Models derived from continuum mechanics plays an important role in simulating deformable objects. The key idea of most models is to represent the resistance of any deformable object as a scalar energy function, whose spatial derivative is the opposite direction of the elastic force.

Mass-spring systems are conceptually the simplest way to simulate deformations. It is commonly used in real-time applications, where in contrast to scientific computing, a highly accurate model is not always the concern. As a result, mass-spring systems are widely used to simulate one and two-dimensional structures such as hair [Selle et al., 2008] and cloth [Choi and Ko, 2005], and can be sometimes extended to simulate elastic solids [Teschner et al., 2004].

Finite Element Methods (FEM) has been used in computer graphics too, mostly in applications where accuracy is strictly needed. A linear model is the simplest FEM model, but it suffers from treating rotation as deformation too, thus it can be only applied in small deformation simulations. Corotated linear elasticity models mitigate this problem by extracting the rotation from deformation using polar decomposition [Chao et al., 2010,

Müller and Gross, 2004]. Nonlinear models such as St. Venant-Kirchhoff, Neo-Hookean or Mooney-Rivlin material models are widely used in many works too. One can refer to the SIGGRAPH course note [Sifakis and Barbic, 2012] for a more general survey for material models derived from continuum mechanics. In order to resolve the unwanted numerical issues caused by degenerated or even inverted elements, Irving et al. [2004] introduced invertible finite element methods that can recover from inversions. Recently, Xu et al. [2015] showed that all corotated linear elasticity, St. Venant-Kirchhoff, and Neo-Hookean materials satisfy the Valanis-Landel assumption [Valanis and Landel, 1967] and can be therefore represented in one unified representation based on principal stretches. They further proposed new "spline based materials" which can be easily controlled by artists to achieve desired animation effects.

Instead of using elastic forces and potentials to simulate deformable objects, constraint-based approaches play an important role in simulating hard constraints such as inextensible cloth [Goldenthal et al., 2007] or posing specific limits to deformations using strain limiting [Narain et al., 2012, Thomaszewski et al., 2009]. Position Based Dynamics [Müller et al., 2007] also models the material as infinitely stiff constraints and solves the Karush-Kuhn-Tucker (KKT) system using nonlinear iterative solvers. For some materials whose potential energy has a special quadratic form, a compliant constraint based approach can be also used to simulate both elastic forces and hard constraints in one unified framework [Macklin et al., 2016, Tournier et al., 2015].

2.2 Time Integration Methods

Regardless of whether one employs a mass-spring system or another method based on continuum mechanics, some numerical time integration technique is necessary to simulate the system dynamics. The most straightforward integration methods are explicit, such as explicit Euler [Press, 2007]. For the purposes of physics-based animation, where performance concerns dictate large timesteps, explicit methods are often not sufficiently robust

and experience stability problems. Seminal works [Baraff and Witkin, 1998, Terzopoulos et al., 1987] introduced the implicit Euler method which offers robustness for large timesteps. Explicit Euler can not stably handle stiff simulation with large timesteps because it constantly injects energy into the simulation, while implicit Euler has its integration error as excessive numerical damping, rapidly removing high-frequency motions from the simulation. Symplectic integrators [Kharevych et al., 2006, Stern and Desbrun, 2006] are known for their long-term energy conservation behaviors. However, those symplectic integrators are not guaranteed to behave stably for all simulations. Implicit Euler integration continues to be one of the most popular choices in applications physics-based animation where robustness is more important and numerical damping is less concerned.

2.3 Numerical Solvers

Regardless of the particular flavor and formulation of implicit integration, Newton's method remains the computational workhorse for solving the system of non-linear equations. Semi-implicit integration can be seen as the result of applying one Newton iteration to implicit integration as well [Baraff and Witkin, 1998, Tournier et al., 2015]. Based on the observation that most elastic forces are conservative, Martin et al. [2011] anti-differentiate the equations of motion in implicit Euler, and convert the original non-linear root finding problem to a non-linear minimization problem which can be solved with Newton's method more robustly. But still, a robust implementation of Newton's method requires lots of precautions such as careful line search [Boyd and Vandenberghe, 2004] and definiteness fix for indefinite Hessians [Irving et al., 2004]. In spite of those special treatments, the performance of Newton's method is already bad because it requires a Hessian matrix evaluation and a linear system solve for *every* iteration.

Multi-grid methods represent another approach to accelerate physics-based simulations [Georgii and Westermann, 2006, McAdams et al., 2011, Müller, 2008, Tamstorf et al., 2015, Wang et al., 2010]. Multi-grid methods are attractive especially for highly detailed meshes

where sparse direct solvers become hindered by high memory requirements. However, constructing multi-resolution data structures and picking suitable parameters is not a trivial task. Another way to speed up FEM is by using subspace simulation where the nodal degrees of freedom are replaced with a low-dimensional linear subspace [An et al., 2008, Barbič and James, 2005, Li et al., 2014]. These methods can be very efficient; however, deformations that were not accounted for during the subspace construction may not be well represented. A variety of approaches have been designed to address this limitation while trying to preserve efficiency [Harmon and Zorin, 2013, Teng et al., 2014, 2015]. Simulating at coarser resolutions is also possible, while crafting special data-driven materials which avoid the loss of accuracy typically associated with lower resolutions [Chen et al., 2015].

Quasi-Newton methods is another way to accelerate Newton's method. A quasi-Newton method employs approximate Hessians, trading fast linear solve with suboptimal descent direction. It has been studied long time before real-time simulations were feasible. Several research papers have been done to accelerate Newton's method in FEM simulations by updating the Hessian approximation only once every frame [Bathe and Cimento, 1980, Fish et al., 1995]. However, even one Hessian update is usually so computationally expensive that can not fit into the computing time limit of real-time applications. Deuflhard [2011] minimizes the number of Hessian factorizations by carefully scheduled Hessian updates. But the update rate will heavily depend on the deformation. A good Hessian approximation suitable for real-time applications should be easy to refactorize or capable of prefactorization. One straightforward constant approximation which is good for prefactorization is the Hessian evaluated at the rest-pose (undeformed configuration). The rest-pose Hessian is positive semi-definite and its use at any configuration enables pre-factorization. Unfortunately, the actual Hessian of deformed configurations is often very different from the rest-pose Hessian and this approximation is therefore not satisfactory for larger deformations. To improve upon this, Müller et al. [2002] introduced per-vertex "stiffness warping" of the rest-pose Hessian, which is more accurate and can still leverage pre-factorized rest-pose Hessian. Unfortunately, the per-vertex stiffness warping approach can introduce

non-physical ghost forces which violate momentum conservation and can lead to instabilities [Müller and Gross, 2004]. This problem was addressed by *per-element* stiffness warping [Müller and Gross, 2004] which avoids the ghost forces but, unfortunately, the per-element-warped stiffness matrices need to be re-factorized, introducing computational overheads which are prohibitive in real-time simulation. For corotated elasticity, Hecht et al. [2012] proposed an improved method which can reuse previously computed Hessian factorization. Specifically, the sparse Cholesky factors are updated only *when* necessary and also only *where* necessary. This spatio-temporal staging of Cholesky updates improves run-time performance, however, the Cholesky updates are still costly and their scheduling can be problematic especially in real-time applications, which require approximately constant per-frame computing costs. Also, the frequency of Cholesky updates depends on the simulation: fast motion with large deformations will require more frequent updates and thus more computation, or risking ghost forces and potential instabilities. Neither is an option in real-time simulators. Quasi-Newton methods have gained their popularity in geometry processing recently. A Laplacian matrix of the mesh can be used to approximate Hessian matrices in geometry processing problems to reduce either the per-iteration computational cost [Kovalsky et al., 2016] or even the number of required iterations [Rabinovich et al., 2017].

Fast simulations of deformable objects using shape matching [Müller et al., 2005, Rivers and James, 2007] paved the way towards more general Position Based Dynamics (PBD) methods [Müller et al., 2007, Stam, 2009]. The past decade witnessed rapid development of Position Based methods, including improvements of the convergence [Kim et al., 2012, Müller, 2008], robust simulation of elastic models [Müller and Chentanez, 2011], generalization to fluids [Macklin and Müller, 2013] and continuum-based materials [Bender et al., 2014a, Müller et al., 2014], unified solvers including multiple phases of matter [Macklin et al., 2014], and most recently, methods to avoid element inversion [Müller et al., 2015]. We refer to a recent survey [Bender et al., 2014b] for a more detailed summary of Position Based methods. As a fast treatment to deliver visually plausible

results of deformable body simulations, Position Based Dynamics is widely used in game engines such as NVIDIA PhysX, Havok Cloth, and Bullet, but rarely used in applications where material properties need to be carefully handled. Macklin et al. [2016] revealed the relationship between PBD and constraint-based dynamics. Their eXtended Position Based Dynamics (XPBD) framework is able to converge towards the actual material properties given a certain type of elastic material, such as mass-spring systems.

Chapter 3

Background

In this chapter, we discuss the basis to simulate hyperelastic materials. We start from the definition of hyperelasticity to the needed spatial and temporal discretization for a simulation. We will also discuss basic time integration methods.

3.1 Hyperelasticity

Elastic objects tend to recover themselves to their undeformed shapes. In continuum mechanics, deformation is usually described as a map ϕ between the undeformed pose (or reference pose) $\mathbf{X}$ in Ω and a deformed pose $\mathbf{x}$ in Ω_t. The relationship between the "current configuration" $\mathbf{x}$ and the "initial configuration" $\mathbf{X}$ is captured by this deformation map $\phi : \mathbb{R}^{d_1} \rightarrow \mathbb{R}^{d_2}$: $\mathbf{x} = \phi(\mathbf{X})$, where d_1 and d_2 are the dimensions of the simulated domain. For example, if the object is performing a rigid body motion, we will have:

$$\mathbf{x} = \mathbf{R}(t)\mathbf{X} + \mathbf{b}(t) \tag{3.1}$$

where $\mathbf{R}$ is a rotation matrix and $\mathbf{b}$ is a translation vector, both $\mathbf{R}$ and $\mathbf{b}$ are dependent on time t. If an 2D object is being stretched in its first dimension with a constant rate v, we will again have:

$$\begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix} = \begin{bmatrix} vt & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X^{(1)} \\ X^{(2)} \end{bmatrix} \tag{3.2}$$

where the super script (1) and (2) denotes for the first and second coordinates.

An important physical quantity is the Jacobian matrix of the deformation map, called deformation gradient $\mathbf{F} \in \mathbb{R}^{d_2 \times d_1}$: $\mathbf{F}_{i,j} = \partial \phi_i(\mathbf{X}) / \partial \mathbf{X}_j$, $i = 1, ..., d_2$, $j = 1, ..., d_1$. In the rigid body motion case and the stretching case we talked before, the deformation gradient $\mathbf{F}$ will be $\mathbf{R}(t)$ and $[vt \quad 0; \quad 0 \quad 1]$ respectively. Intuitively speaking, the deformation gradient can be seen as a *linear* relationship between the reference shape and the current shape. For an arbitrary point and its infinitesimal neighbors, a deformation map ϕ can be approximated closely using this linear relationship:

$$\mathbf{x} = \phi(\mathbf{X}) \approx \mathbf{F}\mathbf{X} + \mathbf{b} \tag{3.3}$$

For elastic materials, the internal force that resists deformation is dependent on the deformation map ϕ and time t. Based on Eq. 3.3, this can be understood as that the force is dependent on the deformation gradient $\mathbf{F}$ for every point of an object and time t. To further simply our problem, we introduce another characteristic property: *hyperelasticity*. The elastic force of a hyperelastic material only depends on the current state but not the prior history to get to that state. For example, as shown in Figure 3.1, the final states of the elastic blobs in both configurations will have the same elastic force, regardless of how they get to their final states. This property is highly related to the fact that the elastic force is *conservative*, which is to say that the work done by the elastic force in moving a particle between two states is independent of the taken path.

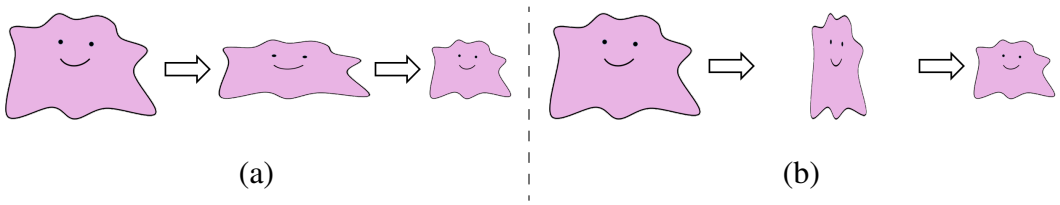


Figure 3.1: Demonstration of hyperelasticity. In configuration (a), an elastic blob is compressed in the vertical direction first and then compressed in the horizontal direction. In configuration (b), the same blob is compressed in horizontal direction first and vertical direction last. Both configurations end up with the same final state.

Based on the assumption of hyperelasticity, we can treat the elastic force as the negative gradient direction of its corresponding elastic energy. Since the resistance of deformation can be very different for each part of an object, the total elastic energy of it can thus be defined as an integral over the elastic energy density function over its entire domain Ω:

$$E(\mathbf{x}) = E(\phi; \mathbf{X}) = \int_{\Omega} \Psi(\phi; \mathbf{X}) d\mathbf{X} \tag{3.4}$$

where $\Psi(\phi; \mathbf{X})$ stands for the elastic energy density close to reference shape point $\mathbf{X}$ under the deformation map ϕ. Based on the approximation (Eq. 3.3) we had for local deformation maps, we can further simply the energy representation as:

$$E(\mathbf{x}) = E(\mathbf{F}; \mathbf{X}) = \int_{\Omega} \Psi(\mathbf{F}(\mathbf{X})) d\mathbf{X} \tag{3.5}$$

3.2 Strain Measures and Material Models

One might think once the relationship between the elastic energy and the deformation gradient is explicitly presented, we can start simulating deformable solids easily. This is true. However, defining an explicit relationship between them is not a straightforward work. For example, inspired by a simple system such as a mass-spring system, if we want a zero elastic energy corresponding to a minimally deformed state, we can define the energy density function to something like:

$$\Psi(\mathbf{F}) = \frac{1}{2}||\mathbf{F} - \mathbf{I}||^2 \tag{3.6}$$

where $\mathbf{I}$ stands for an identity matrix with the same dimensionality of $\mathbf{F}$. This is an intuitive definition because the energy will be zeroed out once the object goes back to its reference configuration. But there are two problems with this definition. First, this energy is not rotational invariant, it penalizes rigid body transformation which should not happen in practice. Second, depending on the dimensionality of the domain of the reference shape and the current shape, the deformation gradient matrix $\mathbf{F}$ might not even be a square matrix, in which case the Identity matrix $\mathbf{I}$ is hard to define. The reason why using the deformation

gradient $\mathbf{F}$ as the measurement for deformation is that it can not be directly linked to a quantitative indicator that can describe how severe the deformation is. Ideally, we want a rotational-invariant descriptor to tell us the severity of a given deformation, this is descriptor is usually called *strain*. In mechanical science, the relationship between the energy density Ψ and the strain (often denoted as $\boldsymbol{\epsilon}$) can be described as:

$$\Psi(\boldsymbol{\epsilon}) = \mu||\boldsymbol{\epsilon}||_F^2 + \frac{\lambda}{2}\,\mathrm{tr}(\boldsymbol{\epsilon})^2 \tag{3.7}$$

where μ is the *Lamé's second parameter* which represents the resistance for shearing, it is also sometimes referred as the *shear modulus* G; λ is the *Lamé's first parameter* which stands for the resistance for change of volume. Other elastic moduli can also be represented as the combination of these two parameters. For instance, Young's Modulus can be represented as $E_{Young} = \mu(3\lambda + 2\mu)/(\lambda + \mu)$, the subscript "Young" is used to distinguish Young's Modulus from the energy representation $E(x)$ we used before. Intuitively speaking, the Frobenius norm of the strain $\boldsymbol{\epsilon}$ should represent how much the material is sheared or stretched, and the trace of $\boldsymbol{\epsilon}$ is a nice linear approximation of the change of volume.

Depending on different measurements of the strain, different types of materials can be defined. For example, *linear elasticity* can be defined with a *small strain tensor* $\boldsymbol{\epsilon}_s$:

$$\boldsymbol{\epsilon}_s = \frac{1}{2}(\mathbf{F} + \mathbf{F}^\mathsf{T}) - \mathbf{I} \tag{3.8}$$

This tensor is not rotational invariant either, but it is not biased towards one rotation direction. It works well only when the deformation is small enough around its reference configuration.

If we want to measure the strain in a rotational invariant way, we can use the *Green strain tensor* $\boldsymbol{\epsilon}_G$:

$$\boldsymbol{\epsilon}_G = \frac{1}{2}(\mathbf{F}^\mathsf{T}\mathbf{F} - \mathbf{I}) \tag{3.9}$$

Substituting this Green strain tensor into Eq. 3.7 will give us the St. Venant-Kirchhoff model. To explain the reason why this model is rotational invariant, we can decompose

the deformation gradient $\mathbf{F}$ into a rotation matrix $\mathbf{R}$ and a symmetric matrix $\mathbf{S}$ using polar decomposition $\mathbf{F} = \mathbf{RS}$. Therefore the Green strain tensor will ignore the rotation component of the deformation gradient automatically because it will be $\boldsymbol{\epsilon}_G = \frac{1}{2}(\mathbf{S}^2 - \mathbf{I})$ after we substitute the polar decomposition into Eq. 3.9. We can intuitively explain the polar decomposition in a 2D FEM simulation: the rotation matrix $\mathbf{R}$ has only one degree of freedom for rotation, while the symmetric matrix $\mathbf{S}$ has the rest of all three degrees of freedom for two stretching modes on the principal axes and one shearing mode. This strain tensor automatically removes the degree(s) of freedom from rotation without explicitly performing the polar decomposition. The only problem is that this tensor is invariant not only to rotation, but to any orthogonal matrix including reflection. If we invert the entire object like flipping an umbrella, the Green strain tensor will still be zero. Numerically speaking, this is caused by multiple minima of the energy density function of the St. Venant Kirchhoff material. In practice, this will cause inversion of a simulated object easily.

Another possibility is to use *corotated linear elasticity* that corresponds to the following strain tensor:

$$\boldsymbol{\epsilon}_c = \mathbf{S} - \mathbf{I} \tag{3.10}$$

where the $\mathbf{S}$ is exactly the symmetric component of the deformation gradient we get from polar decomposition, which encodes all the shearing and stretching information. The idea of corotated linear elasticity is trying to mimic the behavior of linear elasticity as much as possible with an extra safeguard of removing the rotational component of the deformation gradient, enabling the rotational invariant property of the strain measurement. For more constitutive material models used in computer graphics applications, one can refer to Chapter 3 of the SIGGRAPH course note [Sifakis and Barbic, 2012, Part 1].

3.3 Spatial Discretization

In computer graphics applications, we are not able to simulate an entire elastic body as a continuous object, we instead discretize it first and simulate a discrete version of it.

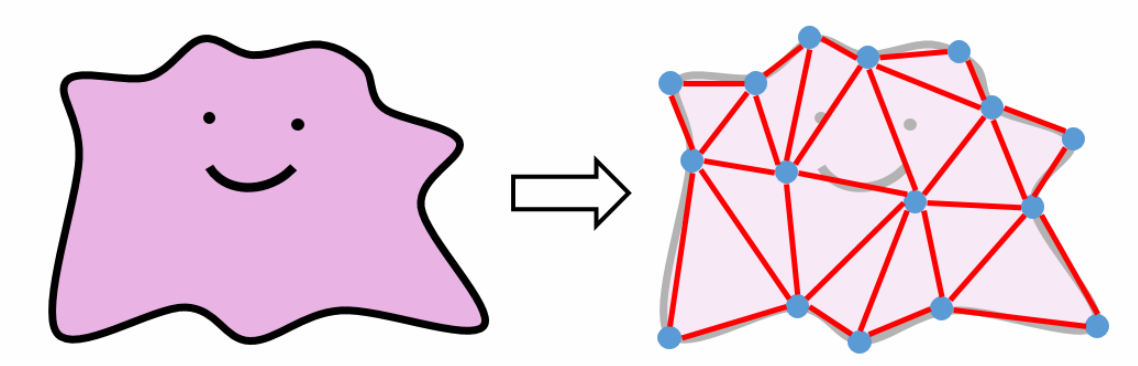


Figure 3.2: Discretization of a deformable blob.

As shown in Figure 3.2, given a continuous planar object, we first tessellate it into triangles. We then distribute the total mass of the elastic object to the individual vertices (shown as blue dots) of the triangles. And we use the position and velocity of those vertices to represent the motion of the entire mesh. For a volumetric deformable object, we similarly tessellate it into tetrahedra, and use the vertices of each tetrahedron to represent the whole mesh too. We usually call a discretized triangle or tetrahedron an *element* of the mesh, since they are the basic building blocks of a mesh now.

Once the tessellation is done, we use a *linear finite element method* to simplify our problem. The key assumption of linear FEM is to rewrite the deformation map as a piecewise linear function over the elements, based on the definition of deformation gradient, for each element $\mathscr{E}_i$, we can define a *linear* deformation map of it as:

$$\phi(\mathbf{X}) = \mathbf{F}_i\mathbf{X} + \mathbf{b}_i \tag{3.11}$$

where $\mathbf{F}_i$ is a *constant* deformation gradient and $\mathbf{b}_i$ is a constant translation vector. Given this assumption, we can further rewrite the energy of the system as:

$$E(\mathbf{x}) = \int_{\Omega} \Psi(\mathbf{F}(\mathbf{X},\mathbf{x}))d\mathbf{X} = \Sigma_{\mathscr{E}_i}\int_{\Omega_{\mathscr{E}_i}} \Psi(\mathbf{F}_i(\mathbf{x}))d\mathbf{X} = \Sigma_{\mathscr{E}_i} w_i \Psi(\mathbf{F}_i(\mathbf{x})) \tag{3.12}$$

where $w_i = \int_{\Omega_{\mathscr{E}_i}} d\mathbf{X}$ represents for the area or volume of the element $\mathscr{E}_i$ in its reference shape. In that case, as long as we can compute the deformation gradient for all the elements, the total elastic energy of the system can be uniquely computed too.

3D Tetrahedral and 2D triangle elements. To simulate a volumetric object in 3D, linear tetrahedral elements are most commonly used. Similarly, linear triangular elements are often used to simulate planar objects in 2D. Let us consider one tetrahedral element whose reference vertices are $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$, $\mathbf{X}_4$ and current vertices are $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, $\mathbf{x}_4$, as shown in Figure 3.3.

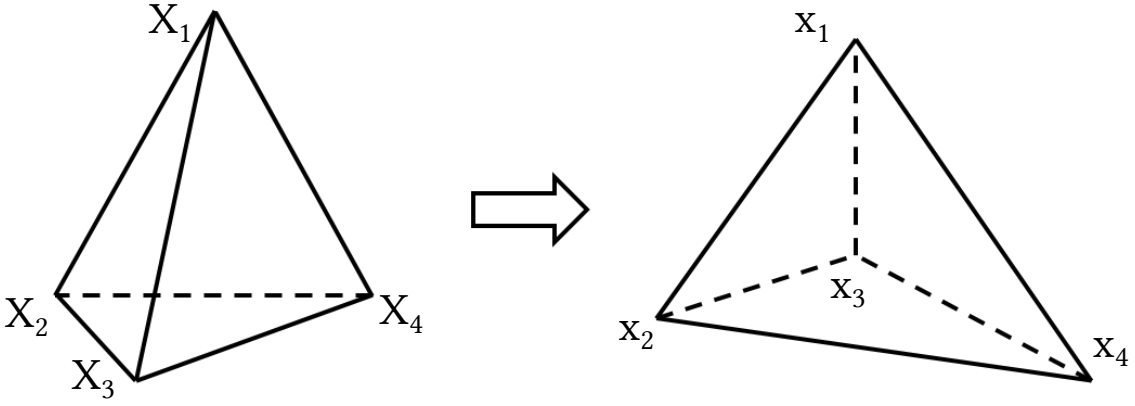


Figure 3.3: One linear tetrahedron being deformed from its reference shape (left) to the current shape (right)

According to the linear assumption of the deformation map for this linear element, we have $\mathbf{x}_i = \mathbf{F}\mathbf{X}_i + b$, $i = 1, ..., 4$. After subtracting the last equation and some rearranging, we will get:

$$\begin{bmatrix} \mathbf{x}_1 - \mathbf{x}_4 & \mathbf{x}_2 - \mathbf{x}_4 & \mathbf{x}_3 - \mathbf{x}_4 \end{bmatrix} = \mathbf{F} \begin{bmatrix} \mathbf{X}_1 - \mathbf{X}_4 & \mathbf{X}_2 - \mathbf{X}_4 & \mathbf{X}_3 - \mathbf{X}_4 \end{bmatrix} \quad (3.13)$$

$$\mathbf{D}_s = \mathbf{F}\mathbf{D}_m \quad (3.14)$$

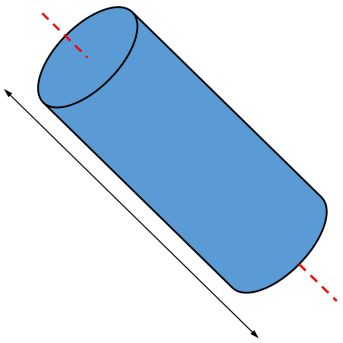
where $\mathbf{D}_s$ is a 3×3 matrix standing for the current *shape matrix* and $\mathbf{D}_m$ stands for the reference shape matrix. If we assume the reference shape is not degenerated with zero-volume, the determinant of $\mathbf{D}_m$ is non-zero. We can further define the deformation gradient of a linear tetrahedral element as:

$$\mathbf{F} = \mathbf{D}_s \mathbf{D}_m^{-1} \quad (3.15)$$

The deformation gradient for a triangular element moving in a 2D space can be defined exactly the same way where $\mathbf{D}_s$ and $\mathbf{D}_m$ are both 2×2 matrices then.

3D triangle elements. Sometimes we want to simulate a volume-less thin sheet moving in a 3D space, such as a piece of cloth or an aluminum coil. Those surfaces are usually represented as 2D triangle meshes. In that case the deformation map ϕ is a map from $\mathbb{R}^2$ to $\mathbb{R}^3$, and therefore the deformation gradient is going to be a 3×2 matrix represented the same as $\mathbf{D}_s\mathbf{D}_m^{-1}$ where $\mathbf{D}_s$ in this case is a 3×2 matrix and $\mathbf{D}_m$ is a 2×2 matrix.

3D spring elements. There is another way to discretize elastic components in a simulation. Instead of using tetrahedra or triangles or codimensional planes, one can always discretize an elastic body into a mass-spring system. As shown in Figure 3.2, we can discretize the entire object into vertices represented as blue dots and springs represented as red lines, ignoring the triangles in between. Apparently, in a reference shape, all springs are in their rest lengths, achieving a zero-elastic-energy state. Note that it is not always true to speak the other way around, because when all springs are in their rest lengths, the mesh itself might not recover to its reference shape. This mass-spring representation is useful in cloth simulation because one can argue that a piece of cloth is made of strings in warp and weft directions, but not made of triangular planes. In order to describe deformation and define a proper energy to the mass-spring system, we can also understand a spring as a bar that can only be deformed in one dimension as shown in the right figure. In this case, the deformation gradient $\mathbf{F}$ of the spring can be as well defined as $\mathbf{F} = \mathbf{D}_s\mathbf{D}_m^{-1}$ where now $\mathbf{D}_s$ is a 3×1 vector $D_s = \mathbf{x}_1 - \mathbf{x}_2$ and $\mathbf{D}_m = \mathbf{X}_1 - \mathbf{X}_2$ is a scalar, leaving $\mathbf{F}$ to be a 3×1 vector too. Inspired by corotated linear elasticity, we can define the strain tensor of this bar as:



$$\boldsymbol{\epsilon}_{bar} = ||\mathbf{F}|| - 1 \tag{3.16}$$

where the norm of $\mathbf{F}$ stands for the only singular value of the 3×1 vector $\mathbf{F}$, indicating the ratio of stretching or compression of the current state, and $\boldsymbol{\epsilon}$ therefore represents how severe the bar is deformed in the dimension where it is allowed to deform. We can further substitute this strain tensor into Eq. 3.7, with the assumption that the bar has only one

degree of freedom and can not resist any volume change, yielding zero λ value. We can write the final energy density of the bar as $\Psi(\mathbf{x}) = \mu(||\mathbf{F}(\mathbf{x})|| - 1)^2$, where the relationship between $\mathbf{F}$ and $\mathbf{x}$ is already known as $\mathbf{F} = (\mathbf{x}_1 - \mathbf{x}_2)/l_0$ with $l_0 = \mathbf{D}_m$ representing the rest length. Assuming the cross-section area of the bar is A_0, we can write the total elastic energy of the bar as:

$$E_{bar}(\mathbf{x}) = A_0 l_0 \mu(||\frac{\mathbf{x}_1 - \mathbf{x}_2}{l_0}|| - 1)^2 = \frac{A_0 \mu}{l_0}(||\mathbf{x}_1 - \mathbf{x}_2|| - l_0)^2 \tag{3.17}$$

Giving the relationship between Hooke's spring stiffness and the Lamé's parameters: $k = A_0 E_{Young}/l_0 = 2A_0\mu/l_0$, we can rewrite the bar energy using the representation of Hooke's Law:

$$E_{bar}(\mathbf{x}) = \frac{A_0 \mu}{l_0}(||\mathbf{x}_1 - \mathbf{x}_2|| - l_0)^2 = \frac{1}{2}k(||\mathbf{x}_1 - \mathbf{x}_2|| - l_0)^2 \tag{3.18}$$

This observation indicates that the energy representation of a mass-spring system following Hooke's Law is also a valid finite element discretization, allowing us to use this simple discretization to simulate complicated deformable objects.

3.4 Temporal Discretization

Simulation is hard because the main job it works on is to predict the future. To be more specific, given the information of the reference shape and existing history of a sequence

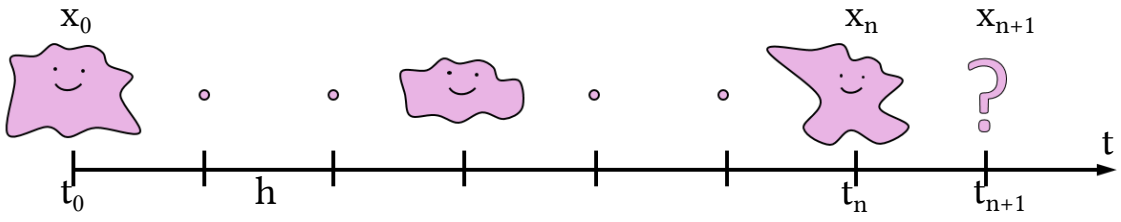


Figure 3.4: A sequence of deformed states along the time axis. The discrete knots on the axis are frames; the length between two adjacent frames on the time axis is the timestep size h.

of deformed shapes along the temporal axis, we want to predict the state of a deformable object in the next timestep. As shown in Figure 3.4, when getting the current positions $\mathbf{x}_n$ and possibly the current velocities $\mathbf{v}_n$ of the vertices of a deformable object, we need to predict the states after a certain timestep h. To make the notation clear, from now on, we will use the subscript of the states as frame indices. For instance, $\mathbf{x}_0$ stands for the positions at the initial state, $\mathbf{v}_n$ stands for the velocities at the current state t_n, and $\mathbf{x}_{n+1}$ stands for the positions at an unknown state after next timestep is taken.

In a continuous time integration scheme, the equations of motion can be represented as follows:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \int_0^h \mathbf{v}(t)dt \tag{3.19}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \int_0^h \mathbf{M}^{-1}(\mathbf{f}_{int}(\mathbf{x}(t) + \mathbf{f}_{ext})dt \tag{3.20}$$

where $\mathbf{M}$ is the mass matrix – typically diagonally lumped, $\mathbf{f}_{int}$ is a state dependent internal force caused by elasticity, $\mathbf{f}_{ext}$ is a state independent force such as gravity, and t is the time variable that goes from 0 to timestep size h, $\mathbf{x}(0)$ means the position at the current state $\mathbf{x}_n$ and $\mathbf{x}(h)$ means the position at the predicted state $\mathbf{x}_{n+1}$. Newton's second law of motion is used in Eq. 3.20, indicating $\mathbf{M}^{-1}(\mathbf{f}_{int}(\mathbf{x}(t) + \mathbf{f}_{ext})$ is the acceleration of an object given its current position.

Evaluating the integrals in Eq. 3.19 and Eq. 3.20 exactly is almost impossible for most reasonably interesting materials. Even in a toy 1D case as shown in Figure 3.5, evaluating the integral exactly means computing the exact purple area of figure (a), which lacks an analytical solution for most 1D functions. What we usually do is to estimate those integrals using numerical approximations. Depending on different approximations of the integrals in Eq. 3.19 and Eq. 3.20, there are generally two schemes for the time integration: explicit time integration and implicit time integration.

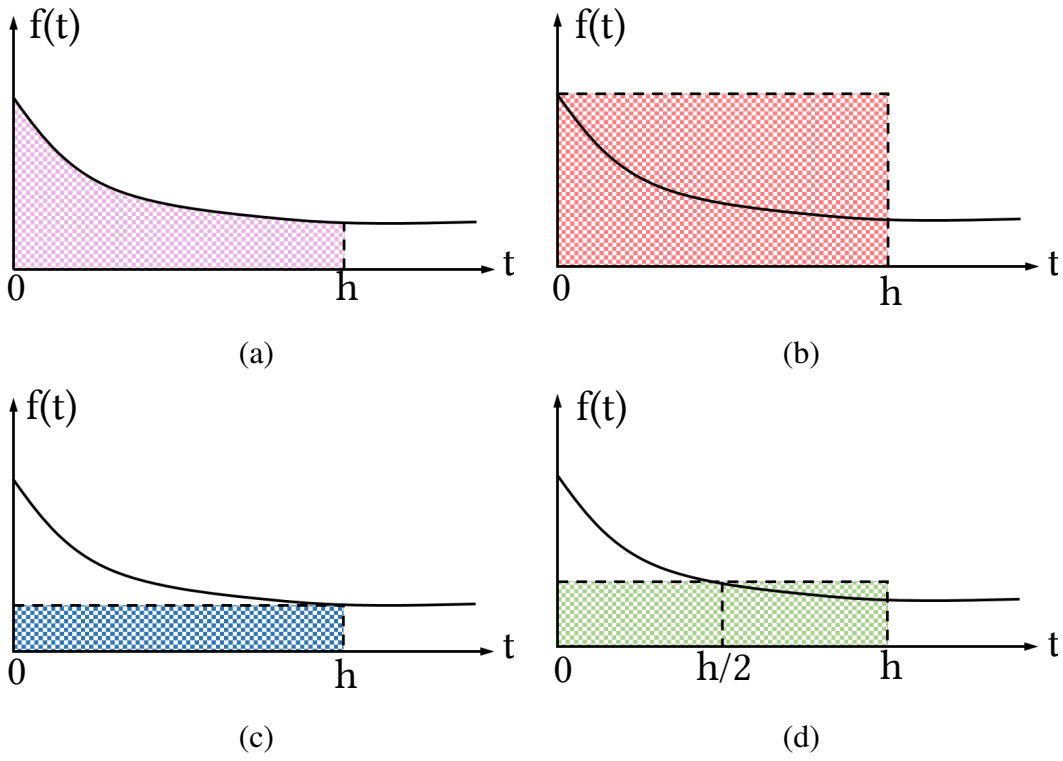


Figure 3.5: Different time integration schemes for a toy 1D case. The area colored in purple in (a) is the exact integral of $\int_0^h f(t)dt$; the red area in (b) is an approximated integral using forward Euler; the blue area in (c) is an approximated integral using backward Euler; the green area in (d) is an approximated integral using implicit midpoint.

3.4.1 Explicit Time Integration Methods

Explicit time integration methods approximate the integrals in Eq. 3.19 and Eq. 3.20 using explicit formula. One straightforward idea is to use position and velocity values at the current state as a constant approximation, assuming they do not change too much during one timestep. It can be illustrated as figure (b) in Figure 3.5. This scheme is usually referred as *forward Euler* or *explicit Euler* which has the following update rules:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{v}_n \tag{3.21}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{M}^{-1}(\mathbf{f}_{int}(\mathbf{x}_n) + \mathbf{f}_{ext}) \tag{3.22}$$

where we can see that the all the unknown variables ($\mathbf{x}_{n+1}$ and $\mathbf{v}_{n+1}$) appear only on the left-hand-side of the equations. This is the reason why it is an explicit time integration method.

Forward Euler has its built-in problem where the Hamiltonian of the system drifts up over time, injecting extra energy into the simulation and therefore making it unstable. One simple way to stabilize the long-term behavior of the energy is to flip the order of position update and velocity update in Eq. 3.21 and Eq. 3.22, which will yield us:

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{M}^{-1}(\mathbf{f}_{int}(\mathbf{x}_n) + \mathbf{f}_{ext}) \tag{3.23}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{v}_{n+1} \tag{3.24}$$

This is still an explicit time integration scheme if executed sequentially, and is commonly referred as *symplectic Euler* [Kharevych et al., 2006, Stern and Desbrun, 2006]. The symplecticiy is a geometrically motivated property that can ensure a steady long-term behavior of the system Hamiltonian *if* the timestep is small enough to ensure its stability. The reason why both of these explicit schemes rely on small time steps is that they step to the future blindly using the current configuration, assuming the forces are not changed during one step. Those blind steps might *overshoot* and gain unwanted energy into the system. One way to mitigate this problem is to use higher order explicit time integrators such as the second order or the fourth order Runge-Kutta methods. We are not going into more details about Runge-Kutta methods here, one can look it up at Chapter 17 in [Press, 2007]. The overall idea of these explicit methods is that they rely on limited timestep sizes which might not be suitable for real-time application where a typical timestep is set to be 10^{-3} to 10^{-2} second, in which case we might want to resort to implicit time integration methods.

3.4.2 Implicit Time Integration Methods

As oppose to forward Euler as an explicit method, one stable approximation of Eq. 3.19 and Eq. 3.20 is to estimate them on the other end of the timestep, i.e., treating them as

constant quantities evaluated at the end of each timestep:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{v}_{n+1} \tag{3.25}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{M}^{-1}(\mathbf{f}_{int}(\mathbf{x}_{n+1}) + \mathbf{f}_{ext}) \tag{3.26}$$

This integration scheme is called *backward Euler* or *implicit Euler* because the unknowns are now on both sides of the equations. It has the opposite Hamiltonian behavior with forward Euler where the long-term behavior of the total energy of a system simulated by backward Euler is going down. It means backward Euler introduces artificial damping into the simulations, stabilizing those simulations even in large timesteps. In the 1D toy example shown in Figure 3.5, backward Euler approximates the integral using the area of the blue box in figure (c).

If a long-term energy conservation behavior is needed, one might want a symplectic version of an implicit time integration method. *Implicit midpoint* can be used for those applications with the following update rules:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\left(\frac{\mathbf{v}_n + \mathbf{v}_{n+1}}{2}\right) \tag{3.27}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{M}^{-1}\left(\mathbf{f}_{int}\left(\frac{\mathbf{x}_n + \mathbf{x}_{n+1}}{2}\right) + \mathbf{f}_{ext}\right) \tag{3.28}$$

In the 1D toy case, implicit midpoint estimates the integral using the green box as shown in figure (d) in Figure 3.5. Note that going implicit is not a safeguard for unconditional stability. Although implicit midpoint method behaves more stable compared to symplectic Euler, it is not unconditionally stable even solved perfectly. That is why implicit Euler is used more often in computer graphics applications given the stability concerns.

If we substitute Eq. 3.26 into Eq. 3.25, we will get a single nonlinear root finding problem with one single unknown variable $\mathbf{x}_{n+1}$:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{v}_n + h^2\mathbf{M}^{-1}\mathbf{f}_{ext} + h^2\mathbf{M}^{-1}\mathbf{f}_{int}(\mathbf{x}_{n+1}) \tag{3.29}$$

Given the assumption that we are simulating hyperelastic materials whose force is conservative, we can represent the internal force as $\mathbf{f}_{int}(\mathbf{x}) = -\nabla_{\mathbf{x}}E(\mathbf{x})$. We can then

plug this into Eq. 3.29 and anti-differentiate the entire equation on $\mathbf{x}_{n+1}$. Once it is done, computing $\mathbf{x}_{n+1}$ is equivalent to minimize the following problem:

$$g(\mathbf{x}) = \underbrace{\frac{1}{2h^2}||\mathbf{x}-\mathbf{y}||^2_{\mathbf{M}}}_{\text{inertia}} + \underbrace{E(\mathbf{x})}_{\text{elasticity}} \tag{3.30}$$

where $\mathbf{y} = \mathbf{x}_n + h\mathbf{v}_n + h^2\mathbf{M}^{-1}\mathbf{f}_{ext}$ is an aggregated vector encoding the history and external forces and $E(\mathbf{x})$ is the elastic energy of the system. By definition, the state of $arg\,min_{\mathbf{x}} g(\mathbf{x})$ is automatically a root for Eq. 3.29 which is the solution of the backward Euler integration. Intuitively, the first term in Eq. 3.30 can be interpreted as "inertial potential", attracting $\mathbf{x}$ towards $\mathbf{y}$, where $\mathbf{y}$ corresponds to a state predicted by Newton's first law – motion without the presence of any internal forces. The second term penalize $\mathbf{x}$ with large elastic deformations. Minimizing this $g(\mathbf{x})$ function corresponds to finding a compromise between the inertia and elasticity. Similarly we can also express the solution for implicit midpoint as the minimum of:

$$\tilde{g}(\mathbf{x}) = \frac{1}{2h^2}||\mathbf{x}-\tilde{\mathbf{y}}||^2_{\mathbf{M}} + E\left(\frac{\mathbf{x}+\mathbf{x}_n}{2}\right) \tag{3.31}$$

where $\tilde{\mathbf{y}} = \mathbf{x}_n + h\mathbf{v}_n + \frac{h^2}{2}\mathbf{M}^{-1}\mathbf{f}_{ext}$ is a modified inertia term, and the energy is evaluated at a interpolated place between the current state and the unknown.

Those formulae are sometimes called "variational implicit time integrators" [Martin et al., 2011], and can be solved with state-of-art numerical solvers such as Newton-Raphson method. We will temporarily leave the formulae here, and defer the numerical solutions to the next chapter.

Chapter 4

State-of-art Numerical Methods

4.1 General Descent Method

Let us begin with the variational form of Implicit Euler integration described using Eq. 3.30, other implicit integrators can be solved in similar ways too. The first term in Eq. 3.30 is the inertia term, which is a pure quadratic term over $\mathbf{x}$ and has a global minimum point at $\mathbf{x} = \mathbf{y}$. This term simply means that the next state position $\mathbf{x}$ will be attracted to the inertia variable $\mathbf{y}$ if there is no internal force at all. The second elastic energy term is non-linear and usually even non-convex, making the minimization of Eq. 3.30 difficult. For an arbitrary non-linear function $g(\mathbf{x})$, since we can not find an analytically solution to minimize it, we shall resort to numerical methods. As long as a deterministic algorithm can produce a sequence $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(k)}, \mathbf{x}^{(k+1)}, \ldots$ that procedurally reduces the energy function: $g(\mathbf{x}^{(k+1)}) < g(\mathbf{x}^{(k)})$, we can run that algorithm until convergence. This kind of algorithm is usually called a descent method that can be briefly described in Alg. 1, where $\delta\mathbf{x}^{(k)}$ is a descent direction determined by a certain algorithm, α is a positive step size to make sure the descent progress is sufficient and the step criterion is usually measured by either the norm of the residual $||\nabla g(\mathbf{x}^{(k)})||$ or the decrement $-\nabla g(\mathbf{x}^{(k)})^{\mathsf{T}}\delta\mathbf{x}^{(k)}$ is small enough. Note that for any descent direction $\delta\mathbf{x}^{(k)}$ the decrement $-\nabla g(\mathbf{x}^{(k)})^{\mathsf{T}}\delta\mathbf{x}^{(k)}$ must be positive, indicating that the angle between the descent direction $\delta\mathbf{x}^{(k)}$ and the infinitesimal

steepest descent direction $-\nabla g(\mathbf{x}^{(k)})$ is acute. Of course setting the descent direction to be always the steepest descent direction $\delta\mathbf{x}^{(k)} = -\nabla g(\mathbf{x}^{(k)})$ is a trivial way to satisfy this. This is usually called *steepest descent method* or *gradient descent method* which is the simplest descent method to implement. But it is also notorious for its slow convergence behavior: a typical non-linear problem usually requires more than thousands of gradient descent iterations to converge.

Algorithm 1: General Descent Method.

$\mathbf{x}^{(1)} :=$ Initial guess;

for $k = 1, \ldots,$ numIterations *&& Stop criterion is not satisfied* **do**

  Find a descent direction: $\delta\mathbf{x}^{(k)}$;

  Line search, find a proper step size: $\alpha > 0$;

  Update: $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha\delta\mathbf{x}^{(k)}$;

end

Overshooting is another problem in general descent methods, that is why we need an extra line search procedure to safeguard the descent process. For simple cases in low dimension spaces, an exact line search can be applied. The key idea of an exact line search is to find a value α to minimize $g(\mathbf{x})$ by setting: $\alpha = \text{argmin}_{\alpha>0}\, g(\mathbf{x} + \alpha\delta\mathbf{x})$. It is normally not applicable for complicated scenarios because of the non-linearity of $g(\mathbf{x})$ itself. Backtracking line search is another alternative to approximately estimate the step size. A backtracking line search usually starts with a initial guess of step size equals to 1, and constantly reduce it by a fixed faction of β, until the Armijo condition $g(\mathbf{x}+\alpha) < g(\mathbf{x})+\gamma\alpha\nabla g(\mathbf{x})^{\mathsf{T}}\delta\mathbf{x}$ is satisfied. Where $\beta \in (0, 1)$ decides how crude the search is, the lower the more crude, and $\gamma \in (0, 0.5)$ is a non-zero constant to ensure sufficient descent progress for every descent step. As suggested by Boyd and Vandenberghe [2004], we usually set $\beta = 0.5$ and $\gamma = 0.03$ in most of our implementations of descent methods.

4.2 Newton's Method

Newton's method is one of the most popular descent methods because of its nice convergence speed. Instead of using the steepest descent direction, Newton's method also takes the curvature of a function at the given point and estimate the descent direction as follows:

$$\delta\mathbf{x} = -\nabla^2 g(\mathbf{x})^{-1}\nabla g(\mathbf{x}) \tag{4.1}$$

where $\nabla^2 g(\mathbf{x})$ is the Hessian matrix. Newton's descent direction can be seen as filtered gradient direction by the inverse of the Hessian matrix. Alg. 2 presents an pseudo code implementation of Newton's method with backtracking line search.

Algorithm 2: Newton Solver with Backtracking Line Search

$\mathbf{x}^{(1)} := \mathbf{y}$;
$g(\mathbf{x}^{(1)}) := \texttt{evalObjective}(\mathbf{x}^{(1)})$
for $k = 1, \ldots,$ numIterations **do**
  $\nabla g(\mathbf{x}^{(k)}) := \texttt{evalGradient}(\mathbf{x}^{(k)})$
  $\nabla^2 g(\mathbf{x}^{(k)}) := \texttt{evalHessian}(\mathbf{x}^{(k)})$
  $\delta\mathbf{x}^{(k)} := -\nabla^2 g(\mathbf{x}^{(k)})^{-1}\nabla g(\mathbf{x}^{(k)})$
  $\alpha := 1/\beta$
  repeat
    $\alpha := \beta\alpha$
    $\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \alpha\delta\mathbf{x}^{(k)}$
    $g(\mathbf{x}^{(k+1)}) := \texttt{evalObjective}(\mathbf{x}^{(k+1)})$
  until $g(\mathbf{x}^{(k+1)}) \leq g(\mathbf{x}^{(k)}) + \gamma\alpha\,(\nabla g(\mathbf{x}^{(k)}))^{\mathsf{T}}\delta\mathbf{x}^{(k)}$;
end

4.2.1 Basic Building Blocks for Newton's Method

If we apply Alg. 2 to solve Eq. 3.30, for every Newton iteration we need to evaluate the gradient $\nabla g(\mathbf{x}) = \frac{\mathbf{M}}{h^2}(\mathbf{x} - \mathbf{y}) + \nabla E(\mathbf{x})$ and the Hessian $\nabla^2 g(\mathbf{x}) = \frac{\mathbf{M}}{h^2} + \nabla^2 E(\mathbf{x})$ once,

and the objective function $g(\mathbf{x}) = \frac{1}{2h^2}||\mathbf{x} - \mathbf{y}||^2_{\mathbf{M}} + E(\mathbf{x})$ at least once. We can see that for an arbitrary material model, as long as we can compute the objective/gradient/Hessian of the elastic potential, the inertia term can be trivially added to the system. These objective/gradient/Hessian evaluations become the basic building blocks for Newton's method.

Mass-spring system in 3D. A simple spring in 3D can be described using its two endpoints and rest-length as shown in Figure 4.1:

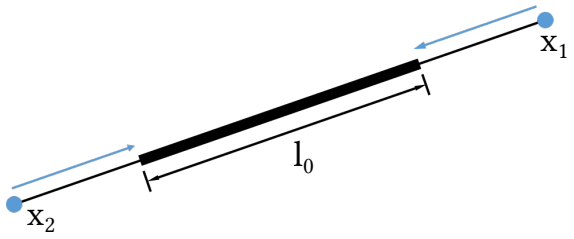


Figure 4.1: One simple spring with its rest length l_0 and two endpoints $\mathbf{x}_1$ and $\mathbf{x}_2$.

According to Hooke's law, the spring potential is defined as:

$$E(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{2}k\left(||\mathbf{x}_1 - \mathbf{x}_2|| - l_0\right)^2 \tag{4.2}$$

where $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^3$ are spring endpoints, $l_0 \geq 0$ is the rest length, and $k \geq 0$ is the spring stiffness.

The spatial gradient of the spring potential can be therefore derived easily as:

$$\frac{\partial E}{\partial \mathbf{x}_1} = k\left(||\mathbf{x}_1 - \mathbf{x}_2|| - l_0\right)\frac{\mathbf{x}_1 - \mathbf{x}_2}{||\mathbf{x}_1 - \mathbf{x}_2||} \tag{4.3}$$

$$\frac{\partial E}{\partial \mathbf{x}_2} = -\frac{\partial E}{\partial \mathbf{x}_1} \tag{4.4}$$

where intuitively, the term $(||\mathbf{x}_1 - \mathbf{x}_2|| - l_0)$ is the magnitude of the spring force and $\frac{\mathbf{x}_1 - \mathbf{x}_2}{||\mathbf{x}_1 - \mathbf{x}_2||}$ indicates the opposite direction of the force. The total elastic force created by a spring sum up to zero: $\sum_{i=1}^{2} \partial E / \partial \mathbf{x}_i = 0$, showing that the inner force of a spring will not change the momentum of itself.

We can further derive the Hessian of the spring as:

$$\frac{\partial^2 E}{\partial \mathbf{x}_1 \partial \mathbf{x}_1} = k\left(\mathbf{I} - \frac{l_0}{||\mathbf{x}_1 - \mathbf{x}_2||}\left(\mathbf{I} - \frac{(\mathbf{x}_1 - \mathbf{x}_2)(\mathbf{x}_1 - \mathbf{x}_2)^\mathsf{T}}{||\mathbf{x}_1 - \mathbf{x}_2||^2}\right)\right) \tag{4.5}$$

$$\frac{\partial^2 E}{\partial \mathbf{x}_1 \partial \mathbf{x}_2} = -\frac{\partial^2 E}{\partial \mathbf{x}_1 \partial \mathbf{x}_1} \tag{4.6}$$

$$\frac{\partial^2 E}{\partial \mathbf{x}_2 \partial \mathbf{x}_1} = -\frac{\partial^2 E}{\partial \mathbf{x}_1 \partial \mathbf{x}_1} \tag{4.7}$$

$$\frac{\partial^2 E}{\partial \mathbf{x}_2 \partial \mathbf{x}_2} = \frac{\partial^2 E}{\partial \mathbf{x}_1 \partial \mathbf{x}_1} \tag{4.8}$$

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

FEM in 3D. Although higher order spatial discretization is also used in graphics applications [Bargteil and Cohen, 2014], we still assume linear finite element for simplicity [Sifakis and Barbic, 2012]. The state of a linear tetrahedral element can be described using its four corner points as shown in Figure 3.3. The potential energy of one element is defined as:

$$E(\mathbf{F}(\mathbf{x})) = w\Psi(\mathbf{F}(\mathbf{x})) \tag{4.9}$$

where $\mathbf{F}(\mathbf{x}) = \mathbf{D}_s(\mathbf{x})\mathbf{D}_m^\mathsf{T}$ is the deformation gradient in Eq. 3.15 and w is the rest-pose volume of the element: $w = det(\mathbf{D}_m)$.

If we denote $\mathbf{x}_i^j$ as the j-th dimension of the i-th vertex of a element, then for $i = 1, 2, 3$, we have:

$$\frac{\partial \mathbf{D}_s}{\partial \mathbf{x}_i^j} = \delta_j \delta_i^\mathsf{T} \tag{4.10}$$

where δ_i is a 3×1 indicator vector: $\delta_i(i) = 1$ and $\delta_i(j) = 0$ for $\forall j \neq i$. $\delta_j \delta_i^\mathsf{T}$ simply means a matrix whose value is 1 at its j-th row and i-the column and is 0 everywhere else. Based on this observation, we can derive the gradient of elastic potential for a 3D linear element as:

$$\frac{\partial E(\mathbf{F}(\mathbf{x}))}{\partial \mathbf{x}_i^j} = w\frac{\partial \Psi(\mathbf{F})}{\partial \mathbf{F}} : \frac{\partial \mathbf{F}}{\partial \mathbf{x}_i^j} = w\mathbf{P} : \delta_j \delta_i^\mathsf{T} \mathbf{D}_m^{-1} = w\,\mathrm{tr}(\mathbf{P}\mathbf{D}_m^{-\mathsf{T}} \delta_i \delta_j^\mathsf{T}) = w\delta_j^\mathsf{T} \mathbf{P}\mathbf{D}_m^{-\mathsf{T}} \delta_i \tag{4.11}$$

where ":" denotes for the contraction operator: $\mathbf{A} : \mathbf{B} = \sum_i \sum_j \mathbf{A}_{ij}\mathbf{B}_{ij}$, similarly to the dot product between vectors, and $\mathbf{P} = \partial \Psi(\mathbf{F})/\partial \mathbf{F}$ is the first Piola-Kirchhoff stress tensor. This

is basically saying that for $i = 1, 2, 3$, $\partial E(\mathbf{F}(\mathbf{x}))/\partial \mathbf{x}_i^j$ is the j-th row and i-th column of matrix $w\mathbf{P}\mathbf{D}_m^{-T}$. Similarly, we can compute $\partial E(\mathbf{F}(\mathbf{x}))/\partial \mathbf{x}_4^j = w\delta_j^\mathsf{T}\mathbf{P}\mathbf{D}_s^{-\mathsf{T}}(-\delta_1 - \delta_2 - \delta_3)$, which also satisfy the following property: $\sum_{i=1}^4 \partial E(\mathbf{F}(\mathbf{x}))/\partial \mathbf{x}_i = 0$.

In order to compute the Hessian matrix of the this kind of elastic potential we can apply the same trick again, for $i_1 = 1, 2, 3$ and $i_2 = 1, 2, 3$, we have:

$$\frac{\partial^2 E(\mathbf{F}(\mathbf{x}))}{\partial \mathbf{x}_{i_1}^{j_1} \partial \mathbf{x}_{i_2}^{j_2}} = w\delta_{j_1}^\mathsf{T} \left[\left[\delta_{j_2}^\mathsf{T} \left[\frac{\partial \mathbf{P}}{\partial \mathbf{F}} \mathbf{D}_m^{-\mathsf{T}} \right] \delta_{i_2} \right] \mathbf{D}_m^{-\mathsf{T}} \right] \delta_{i_1} \tag{4.12}$$

This is an almost-tensor-free way to explain the hessian matrix, with complicated subscripts. Here i_1 and i_2 are indices of the vertices of the elements, j_1 and j_2 corresponds to the x-, y- and z- coordinates, $\partial \mathbf{P}/\partial \mathbf{F}$ is a $3 \times 3 \times 3 \times 3$ tensor. Speaking in English, as long as the tensor $\partial \mathbf{P}/\partial \mathbf{F}$ is computed, we can first multiply it by $\mathbf{D}_m^{-\mathsf{T}}$, then exact the j_2-th row and i_2-th column of this tensor which becomes a 3×3 matrix, multiply the matrix by $\mathbf{D}_m^{-\mathsf{T}}$ again, and then we can find the scalar $\partial^2 E(\mathbf{F}(\mathbf{x}))/\partial \mathbf{x}_{i_1}^{j_1} \partial \mathbf{x}_{i_2}^{j_2}$ at the j_1-th row and i_1-th column of the result matrix. The implementation of this Hessian matrix construction only requires some reshuffling operations once the tensor $\partial \mathbf{P}/\partial \mathbf{F}$ is computed.

Different energy density functions might be used for different simulations. The corresponding first Piola-Kirchhoff stress tensor $\mathbf{P}$ and its derivative $\partial \mathbf{P}/\partial \mathbf{F}$ will be changed from material models too. Please refer to [Sifakis and Barbic, 2012] for specific FEM material models.

4.2.2 Limitations

Although the Hessian evaluation and construction look complicated, the computational cost of it is usually not extremely high if implemented carefully. A bigger problem is the solve of the linear system in Eq. 4.1 to get the descent direction for every Newton iteration. Apparently, we should not invert the Hessian matrix explicitly because that will produce a huge dense matrix, a direct solver using Cholesky factorization or an iterative solve such as conjugate gradient method is usually applied to solve this linear system. But neither of them can achieve real-time performance even for middle-scaled meshes with thousands of

degrees of freedom.

Another even worse problem is that Newton's method was originally designed for convex optimization problems, and it might not behave well in non-convex cases. Unfortunately, most of the elastic potential energies are non-convex.

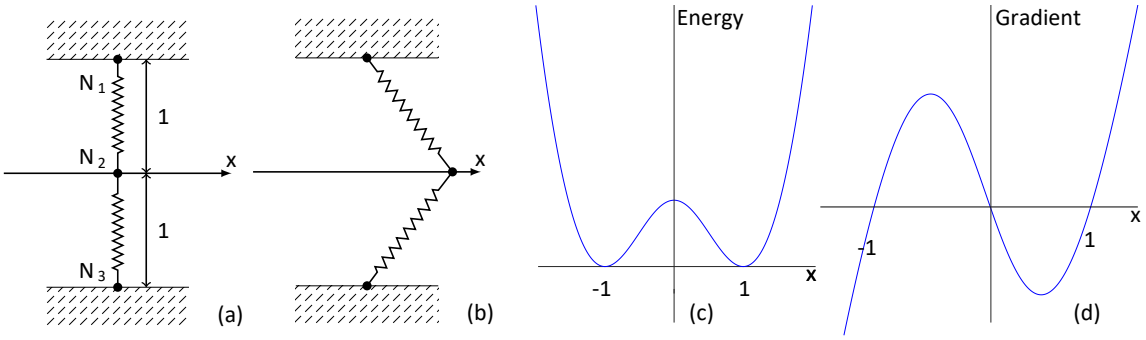


Figure 4.2: A simple mass-spring model with only one degree of freedom: the central vertex (N_2) can slide on the horizontal direction with a parameter x, as shown in (a, b). The rest length of both the springs are $\sqrt{2}$. Figure (c) plots the total energy of the two springs, which has two minima at $x = -1$ and $x = 1$ and one local maximum at $x = 0$. Figure (d) shows the gradient of the spring energy.

Figure 4.2 shows that even a mass-spring system in an extremely simple setting, the energy of the entire system is still non-convex. If given an initial guess near $x = 0$, Newton's method will even return us an ascent direction pointing to $x = 0$. This is because the Newton direction is not attracted by the minima of the system, but by any stationary point. One way to enforce a descent direction from Newton's method is to modify the Hessian matrix to a positive definite one. Once the Hessian matrix is positive definite, it is ensured that the angle between the Newton direction and the steepest descent direction will be acute: $\left(-\left[\nabla^2 g\right]^{-1}\nabla g\right)^{\mathsf{T}}(-\nabla g) = ||\nabla g||^2_{[\nabla^2 g]^{-1}} > 0$.

In order to achieve the positive definiteness of the Hessian matrix, we can apply Tikhonov regularization to the system Hessian matrix. It is usually a trial-and-error approach applied with direct factorizations. One can try to add more and more identity

matrices to the system Hessian upon failed Cholesky factorization cases until the factorization is succeeded. Another way to factorize the Hessian matrix is based on the observation that the system Hessian matrix is the summation of the Hessian matrices of all elements. Therefore as long as all elements have positive definite Hessian matrices, the overall system Hessian will be positive definite too [Irving et al., 2004]. The per-element Hessian regularization method works on each element individually, it applies eigen decomposition to the Hessian matrices of all elements and clamps all negative eigen values to non-negative.

The drawback of Tikhonov regularization is that it requires a proper heuristics to determine how many identity matrices are needed for achieving a positive definite Hessian, it might take a huge amount of time if the input Hessian matrix is far away from positive definite. On the other hand, the per-element regularization works on much smaller matrices and has an explicit upper bound for its computational overhead. However, it will waste a lot of cycles even on elements with already positive definite Hessian blocks because the eigen decomposition will be applied on those "good elements" before we know they are good. Both of the regularization methods will hurt the convergence property of Newton's method because the curvature information obtained from the Hessian matrix will be blended with extra regularization terms.

4.3 Position Based Dynamics

4.3.1 Problem Statement

Position based dynamics (PBD) [Müller et al., 2007] was introduced to computer graphics as an alternative to simulate deformable objects to an extent of visually plausibility in an efficient way. However, people usually consider Position Based Dynamics as an ad-hoc solution because it is not rigorously derived from continuum mechanics. Recently, Macklin et al. [2016] revealed the relationship between PBD and a constrained dynamics simulation. Instead of simulating a variational implicit Euler integration in Eq. 3.30. Position based

dynamics is trying to simulate a constrained problem as following:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2h^2}||\mathbf{x}-\mathbf{y}||^2_{\mathbf{M}} \\ \text{subject to} \quad & \mathbf{c}(\mathbf{x}) = \mathbf{0} \end{aligned} \tag{4.13}$$

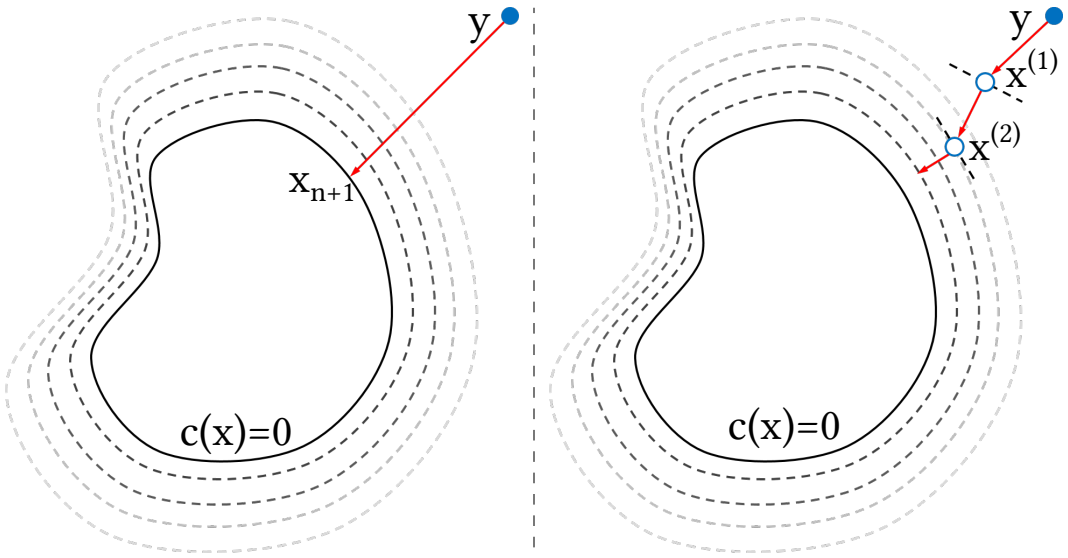
where the objective function is simply the inertia term in Eq. 3.30, but instead of representing all the elastic potential as a penalty-like term, PBD treats all the constraints as infinitely hard constraints. For instance, in a mass-spring system simulation, similarly to [Goldenthal et al., 2007], PBD also treats all springs as inextensible springs, and defines the constraint for a spring in Figure 4.1 as: $c(\mathbf{x}_1, \mathbf{x}_2) = ||\mathbf{x}_1 - \mathbf{x}_2|| - l_0 = 0$.

4.3.2 Fast Numerics

We can interpret the constrained problem described in Eq. 4.13 as finding the closest projection on a non-linear manifold $\mathbf{c}(\mathbf{x}) = \mathbf{0}$ from a given point $\mathbf{y}$. It is hard to solve solely because of the non-linearity of the constraints. It will be a much simpler problem if all constraints are linear. Goldenthal et al. [2007] proposed a method to approximate this non-linear projection in an iterative way called *Step and Project (SAP)*. Let us consider a sequence of points: $\mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}$, ... , $\mathbf{x}^{(k)}$, ... where $\mathbf{x}^{(1)} = \mathbf{y}$ is the initial guess. The rule to update each point in this sequence can be defined as following:

$$\begin{aligned} \mathbf{x}^{(k+1)} = \underset{x}{\text{argmin}} \quad & \frac{1}{2h^2}||\mathbf{x}-\mathbf{x}^{(k)}||^2_{\mathbf{M}} \\ \text{subject to} \quad & \mathbf{c}(\mathbf{x}^{(k)}) + \nabla_{\mathbf{x}}\mathbf{c}(\mathbf{x}^{(k)})(\mathbf{x}-\mathbf{x}^{(k)}) = \mathbf{0} \end{aligned} \tag{4.14}$$

The differences between Eq. 4.14 and Eq. 4.13 are: 1. instead of minimizing the distance between $\mathbf{x}_{n+1}$ and $\mathbf{y}$, SAP tries to deviate the position in the next iteration as little as possible from the *previous iteration*; 2. for every iteration of SAP, the constraints treated as linear constraints. A geometric interpretation of SAP can be found in Figure 4.3, where the left figure shows a fulled solved problem described in Eq. 4.13 and the right figure shows the steps of SAP. From this geometric interpretation, we can easily see why this algorithm is called Step and Project: for every iteration, it projects the positions of the



Fully solved Eq. 4.13 | Step and Project

Figure 4.3: Geometric interpretation of Step and Project (SAP).

vertices into a zero-value manifold for the linearized constraints, and then steps towards that projection.

We can use Newton-Raphson to solve Eq. 4.14. Since this is a linearly-constrained quadratic problem, Newton Raphson should be able to converge in one iteration with an arbitrary initial guess. Therefore we can set the initial guess as $\mathbf{x} = \mathbf{x}^{(k)}$ and $\boldsymbol{\lambda} = \mathbf{0}$, which will give us the update rule for $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \delta\mathbf{x}$ as:

$$\begin{bmatrix} \frac{\mathbf{M}}{h^2} & \nabla_{\mathbf{x}}\mathbf{c}(\mathbf{x}^{(k)})^{\mathsf{T}} \\ \nabla_{\mathbf{x}}\mathbf{c}(\mathbf{x}^{(k)}) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x} \\ \delta\boldsymbol{\lambda} \end{bmatrix} = - \begin{bmatrix} \mathbf{0} \\ \mathbf{c}(\mathbf{x}^{(k)}) \end{bmatrix} \tag{4.15}$$

We can do an even better job to solve this KKT system by observing the simple structure of the top-left corner of the KKT matrix. By applying a *Schur Complement* to the bottom-right zero-matrix in the KKT matrix, we will have:

$$(\mathbf{0} - \nabla_{\mathbf{x}}\mathbf{c}(\mathbf{x}^{(k)})(\frac{\mathbf{M}}{h^2})^{-1}\nabla_{\mathbf{x}}\mathbf{c}(\mathbf{x}^{(k)})^{\mathsf{T}})\delta\boldsymbol{\lambda} = -\mathbf{c}(\mathbf{x}^{(k)}) \tag{4.16}$$

which can be rearranged into:

$$\delta\boldsymbol{\lambda} = \frac{1}{h^2}(\nabla_{\mathbf{x}}\mathbf{c}(\mathbf{x}^{(k)})\mathbf{M}^{-1}\nabla_{\mathbf{x}}\mathbf{c}(\mathbf{x}^{(k)})^{\mathsf{T}})^{-1}\mathbf{c}(\mathbf{x}^{(k)}) \tag{4.17}$$

Since we often represent the mass matrix $\mathbf{M}$ as a diagonally lumped matrix, inverting it is not a problem. For the matrix $\nabla_{\mathbf{x}}\mathbf{c}(\mathbf{x}^{(k)})\mathbf{M}^{-1}\nabla_{\mathbf{x}}\mathbf{c}(\mathbf{x}^{(k)})^{\mathsf{T}}$, although we use a inversion notation there in the right hand side of the equation, in practice, it is not necessary to invert it explicitly. We can instead rely on direct linear solvers based on Cholesky factorization or iterative solvers such as conjugate gradient method to solve it. By definition, this system matrix is a positive semi-definite matrix and might be singular in edge cases. In those cases, we also need to regularize it in order to solve the linear system.

After $\delta\lambda$ is solved, we can solve for $\delta\mathbf{x}$ using the first row of Eq. 4.15:

$$\delta\mathbf{x} = -h^2\mathbf{M}^{-1}\nabla_{\mathbf{x}}\mathbf{c}(\mathbf{x}^{(k)})^{\mathsf{T}}\delta\boldsymbol{\lambda} \tag{4.18}$$

PBD is trying to solve exactly Eq. 4.17 and Eq. 4.18. However, instead of solving all the constraints as a whole, it projects each constraint one at a time. It is called "Non-linear Gauss-Seidel" because it resembles Gauss-Seidel iterations to solve linear systems. For each constraint, the PBD update rules of $\delta\lambda$ and $\delta\mathbf{x}$ are as following:

$$\delta\boldsymbol{\lambda} = \frac{\mathbf{c}(\mathbf{x}^{(k)}}{h^2\nabla_{\mathbf{x}}\mathbf{c}(\mathbf{x}^{(k)})\mathbf{M}^{-1}\nabla_{\mathbf{x}}\mathbf{c}(\mathbf{x}^{(k)})^{\mathsf{T}}} \tag{4.19}$$

$$\delta\mathbf{x} = -h^2\mathbf{M}^{-1}\nabla_{\mathbf{x}}\mathbf{c}(\mathbf{x}^{(k)})^{\mathsf{T}}\delta\boldsymbol{\lambda} \tag{4.20}$$

For a mass-spring system with inextensible spring constraints as $c(\mathbf{x}_1, \mathbf{x}_2) = ||\mathbf{x}_1 - \mathbf{x}_2|| - l_0 = 0$, we can substitute the constraint function and constraint gradient into Eq. 4.19 and Eq. 4.20 to achieve:

$$\delta\mathbf{x}_1 = -\frac{m_1^{-1}}{m_1^{-1} + m_2^{-1}}(||\mathbf{x}_1 - \mathbf{x}_2|| - l_0)\frac{\mathbf{x}_1 - \mathbf{x}_2}{||\mathbf{x}_1 - \mathbf{x}_2||} \tag{4.21}$$

$$\delta\mathbf{x}_2 = +\frac{m_2^{-1}}{m_1^{-1} + m_2^{-1}}(||\mathbf{x}_1 - \mathbf{x}_2|| - l_0)\frac{\mathbf{x}_1 - \mathbf{x}_2}{||\mathbf{x}_1 - \mathbf{x}_2||} \tag{4.22}$$

where the $\delta\mathbf{x}_1$ and $\delta\mathbf{x}_2$ are the projections for the two endpoints of a spring, m_1 and m_2 are the weights of the two endpoints respectively.

4.3.3 Limitations

Neither SAP nor PBD is converging to the exact solution of Eq. 4.13, their results will drift because they discard the information of $\mathbf{y}$ right after the very first iteration. However,

if enough number of iterations is executed, both of them will give a reasonable state that almost satisfy all the non-linear constraints. The major problem of PBD is the poor convergence speed. In practice, no one has even tried to run PBD until converge due to the little progress every iteration can make – a converged solution using PBD typically needs hundreds or thousands of iterations to achieve. Usually, applications based on PBD will terminate in a very small number of iterations because of the real-time requirement. One side effect of not having enough PBD iterations in a simulation is that the material will look stretchier than it should be. This is due to the initial guess before the projection steps. As described in the paper, the state vector $\mathbf{x}$ is initialized as $\mathbf{y}$ which represents the inertia term following Newtons first law of motion. If the work done by the nonlinear Gauss-Seidel solve or Jacobi solve is not big enough, the final solution will be attracted more by this $\mathbf{y}$ term, behaving overstretched.

Chapter 5

Fast Simulation of Mass-Spring Systems

Mass-spring systems provide a simple yet practical method for modeling a wide variety of objects, including cloth, hair, and deformable solids. However, as with other methods for modeling elasticity, obtaining realistic material behaviors typically requires constitutive parameters that result in numerically stiff systems. Explicit time integration methods are fast but when applied to these stiff systems they have stability problems and are prone to failure. Traditional methods for implicit integration remain stable but require solving large systems of equations [Baraff and Witkin, 1998, Press, 2007]. The high cost of solving these systems of equations limits their utility for real-time applications (e.g., games) and slows production work flows in off-line settings (e.g., film and visual effects).

In this chapter, we describe a fast implicit solver for standard mass-spring systems with spring forces governed by Hooke's law. We consider the optimization formulation of implicit Euler integration [Martin et al., 2011], where time-stepping is cast as a minimization problem. Our method works well with large timesteps—most of our examples assume a fixed timestep corresponding to the framerate, i.e., $h = 1/30s$. In contrast to the traditional approach of employing Newton's method, we reformulate this minimization problem by introducing auxiliary variables (spring directions). This allows us to apply a block coordinate descent method which alternates between finding optimal spring directions (local step) and finding node positions (global step). In the global step, we solve a linear

system. The matrix of our linear system is independent of the current state, which allows us to benefit from a pre-computed sparse Cholesky factorization.

5.1 Reformulating Spring Potential

As we have seen in Figure 4.2 before, the spring potential defined by Hooke's law forms a non-convex optimization problem that even Newton's method has troubles to solve it. The main idea of our method is to reformulate the energy potential E in a way that will allow us to employ a block coordinate descent method. The key to our reformulation is the following fact showing that the spring potential in Eq. 4.2 is a solution to a specially designed constrained minimization problem:

$$\frac{1}{2}k(||\mathbf{x}_1 - \mathbf{x}_2|| - l_0)^2 = \min_{||\mathbf{p}||=l_0} \frac{1}{2}k||(\mathbf{x}_1 - \mathbf{x}_2) - \mathbf{p}||^2 \tag{5.1}$$

Proof. Given the constraint $||\mathbf{p}|| = l_0$, we can rewrite the left hand side of Eq. 5.1 into:

$$\frac{1}{2}k(||\mathbf{x}_1 - \mathbf{x}_2|| - l_0)^2 = \frac{1}{2}k(||\mathbf{x}_1 - \mathbf{x}_2|| - ||\mathbf{p}||)^2$$

By applying the reverse triangle inequality we have:

$$(||\mathbf{x}_1 - \mathbf{x}_2|| - ||\mathbf{p}||)^2 \leq ||(\mathbf{x}_1 - \mathbf{x}_2) - \mathbf{p}||^2$$

Next, if we substitute $\mathbf{p} = l_0 \frac{\mathbf{x}_1 - \mathbf{x}_2}{||\mathbf{x}_1 - \mathbf{x}_2||}$ into the right hand side of Eq. 5.1, we obtain:

$$\begin{aligned} \frac{1}{2}k\left|\left|(\mathbf{x}_1 - \mathbf{x}_2) - l_0\frac{\mathbf{x}_1 - \mathbf{x}_2}{||\mathbf{x}_1 - \mathbf{x}_2||}\right|\right|^2 &= \frac{1}{2}k\left|\left|\frac{\mathbf{x}_1 - \mathbf{x}_2}{||\mathbf{x}_1 - \mathbf{x}_2||}(||\mathbf{x}_1 - \mathbf{x}_2|| - l_0)\right|\right|^2 \\ &= \frac{1}{2}k(||\mathbf{x}_1 - \mathbf{x}_2|| - l_0)^2 \end{aligned}$$

Therefore, when $\mathbf{p} = l_0 \frac{\mathbf{x}_1 - \mathbf{x}_2}{||\mathbf{x}_1 - \mathbf{x}_2||}$, the right hand side of Eq. 5.1 produces its minimum value that equals to the left. □

We name the auxiliary variable $\mathbf{p}$ to represent projection, because the closed-form solution of $\mathbf{p}$ to minimize the constrained optimization problem can be seen as a *projection*:

for each spring, $\mathbf{p}$ is the projection of the difference between two endpoints of a spring $\mathbf{x}_1 - \mathbf{x}_2$ onto a sphere whose radius is the rest length of the spring l_0.

If we sum up the contribution of all springs together, we get the potential energy of the entire mass-spring system as:

$$E(\mathbf{x}) = \frac{1}{2}\sum_{j=1}^{m}\left(k_j \min_{\mathbf{p}_j = l_{j0}} ||\mathbf{x}_{j1} - \mathbf{x}_{j2} - \mathbf{p}_j||^2\right) = \frac{1}{2}\sum_{j=1}^{m}\left(k_j \min_{\mathbf{p}_j = l_{j0}} ||\mathbf{x}_{j1}^\mathsf{T} - \mathbf{x}_{j2}^\mathsf{T} - \mathbf{p}_j^\mathsf{T}||^2\right) \tag{5.2}$$

where m is the number of springs in the mass-spring system, for the j-th spring, k_j is its stiffness constant, l_{j0} is its rest length, $\mathbf{x}_{j1}$ and $\mathbf{x}_{j2}$ are positions of its two endpoints, $\mathbf{p}_j$ is the auxiliary projection variable for this j-th spring. The transpositions of all vectors are intentionally added to fit the representation of our position vector $\mathbf{x}$. We represent the position $\mathbf{x}$ in a three-dimensional space as a $n \times 3$ matrix where n is the number of vertices. $\mathbf{x} = [\mathbf{x}_1^\mathsf{T}; \mathbf{x}_2^\mathsf{T}; \ldots; \mathbf{x}_n^\mathsf{T}]$, similarly we can introduce our projection matrix $\mathbf{p} \in \mathbb{R}^{m\times 3}$ for the entire system: $\mathbf{p} = [\mathbf{p}_1^\mathsf{T}; \mathbf{p}_2^\mathsf{T}; \ldots; \mathbf{p}_m^\mathsf{T}]$. Based on this notation, we can see that:

$$\mathbf{x}_{j1}^\mathsf{T} - \mathbf{x}_{j2}^\mathsf{T} = \mathbf{G}_j^\mathsf{T}\mathbf{x} \tag{5.3}$$

$$\mathbf{p}_j = \mathbf{S}_j^\mathsf{T}\mathbf{p} \tag{5.4}$$

where $\mathbf{G}_j \in \mathbb{R}^{n\times 1}$ is the incidence vector of the j-th spring, i.e. $\mathbf{G}_j(j1) = 1$, $\mathbf{G}_j(j2) = -1$ and zero otherwise, assuming $j1$ and $j2$ are the indices for the endpoints of the j-th spring. Similarly, $\mathbf{S}_j \in \mathbb{R}^{m\times 1}$ is a indicator vector for the j-th spring, $\mathbf{S}_j = \delta_j$. Therefore, we can rewrite Eq. 5.2 as:

$$E(\mathbf{x}) = \min_{\mathbf{p}\in\mathbb{M}} \frac{1}{2}\,\mathrm{tr}\left(\mathbf{x}^\mathsf{T}\left(\sum_{j=1}^{m} k_j\mathbf{G}_j\mathbf{G}_j^\mathsf{T}\right)\mathbf{x}\right) - \mathrm{tr}\left(\mathbf{x}^\mathsf{T}\left(\sum_{j=1}^{m} k_j\mathbf{G}_j\mathbf{S}_j^\mathsf{T}\right)\mathbf{p}\right) + C \tag{5.5}$$

where $\mathbb{M} = \{(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m) : ||\mathbf{p}_j|| = l_{j0}\}$ is the set of rest-length spring directions, C is a constant irrelevant to the optimization. For brevity, we define $\mathbf{L} = \sum_{j=1}^{m} k_j\mathbf{G}_j\mathbf{G}_j^\mathsf{T}$ and $\mathbf{J} = \sum_{j=1}^{m} k_j\mathbf{G}_j\mathbf{S}_j^\mathsf{T}$, thus we can further simplify Eq. 5.5 into:

$$E(\mathbf{x}) = \min_{\mathbf{p}\in\mathbb{M}} \frac{1}{2}\,\mathrm{tr}\left(\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x}\right) - \mathrm{tr}\left(\mathbf{x}^\mathsf{T}\mathbf{J}\mathbf{p}\right) + C \tag{5.6}$$

Substituting Eq. 5.6 into our variational implicit Euler integrator Eq. 3.30 will yield us:

$$g(\mathbf{x}) = \min_{\mathbf{p}\in\mathbb{M}} \left(\frac{1}{2h^2} \text{tr}\left((\mathbf{x}-\mathbf{y})^\mathsf{T}\mathbf{M}(\mathbf{x}-\mathbf{y})\right) + \frac{1}{2}\text{tr}\left(\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x}\right) - \text{tr}\left(\mathbf{x}^\mathsf{T}\mathbf{J}\mathbf{p}\right) + C \right) \tag{5.7}$$

Therefore, once the following constrained minimization problem is solved, the entire implicit Euler integration is solved too:

$$\min_{\mathbf{x}\in\mathbb{R}^{n\times 3},\mathbf{p}\in\mathbb{M}} \left(\frac{1}{2h^2} \text{tr}\left((\mathbf{x}-\mathbf{y})^\mathsf{T}\mathbf{M}(\mathbf{x}-\mathbf{y})\right) + \frac{1}{2}\text{tr}\left(\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x}\right) - \text{tr}\left(\mathbf{x}^\mathsf{T}\mathbf{J}\mathbf{p}\right) + C \right) \tag{5.8}$$

5.2 Numerical Solution

It might sound like a bad idea to reformulate our original unconstrained minimization problem Eq. 3.30 into a constrained optimization problem Eq. 5.8 with an extra auxiliary variable $\mathbf{p}$ which almost doubles the degrees of the freedom of the system. However, the new problem Eq. 5.8 can be really easily solved by alternating optimization (also known as local-global optimization [Sorkine and Alexa, 2007]). Starting from an initial guess for $\mathbf{x}$ (we use $\mathbf{x}^{(1)} = \mathbf{y}$ as a trivial solution to minimize the inertia term), we first fix $\mathbf{x}$ and compute the projection vector $\mathbf{p}$. This is usually called a "local step" because, in order to get the stacked vector $\mathbf{p}$, we only need to go find all the spring directions. For every spring, it is a local operation because the positions of the endpoints are all fixed. Second, we fix $\mathbf{p}$ and compute the optimal $\mathbf{x}$. This is called the "global step" because it is solving a large convex quadratic optimization problem that involved all the vertices together. The solution for the "global step" is trivial when $\mathbf{p}$ is fixed, we can even compute a closed-form solution $\mathbf{x}^*$ as:

$$\mathbf{x}^* = \left(\frac{M}{h^2} + \mathbf{L}\right)^{-1} \left(\frac{M}{h^2}\mathbf{y} + \mathbf{J}\mathbf{p}\right) \tag{5.9}$$

By definition $\mathbf{L}$ is a positive-semidefinite matrix (it is very similar to the graphical Laplacian matrix of the system), thus the system matrix $\left(\frac{M}{h^2} + \mathbf{L}\right)$ is guaranteed to be positive definite assuming all vertices have positive mass. This matrix is only dependent on the connectivity and stiffness of the springs, the mass of all vertices and the time step which are all not likely to change during the simulation. Therefore, we can pre-compute its

sparse Cholesky factorization (guaranteed to exist), which makes the linear system solve very fast. Also, because under our reformulation, the position variable $\mathbf{x}$ can be represented as a $n \times 3$ matrix instead of a $3n \times 1$ vector, the dimension of system matrix $\left(\frac{M}{h^2} + \mathbf{L}\right)$ is much smaller than the Hessian matrix, acceleration the linear system solve even more.

We run this local-global optimization repeatedly until the maximum number of iterations is executed or the constrained optimization problem is converged. Note that this method for mass-spring system simulation is not an ad-hoc solution, it will converge to the exact same solution of the implicit Euler method with standard Hookean springs.

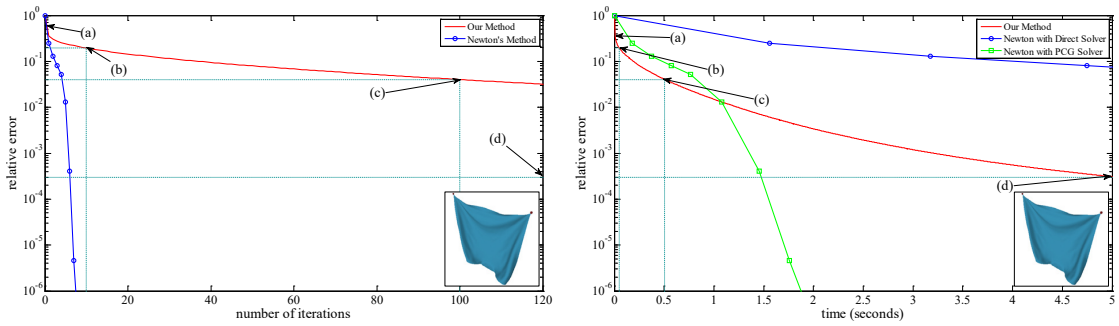
5.3 Results

We study the convergence speed on one typical frame of our cloth-swinging animation, as shown in Figure 5.1. The relative error reported in Figure 5.1 is defined as:

$$\frac{g(\mathbf{x}^{(i)}) - g(\mathbf{x}^*)}{g(\mathbf{x}^{(1)}) - g(\mathbf{x}^*)} \tag{5.10}$$

where $\mathbf{x}^{(1)}$ is the initial guess, $\mathbf{x}^{(i)}$ is the current iterate, and $\mathbf{x}^*$ is the final solution. Our method exhibits a linear convergence rate, whereas Newton's method quickly enters its quadratic convergence phase [Boyd and Vandenberghe, 2004]. However, Figure 5.1 (left) ignores the fact that one iteration of Newton's method is computationally substantially more expensive than one iteration of our method. In Figure 5.1 (right), we therefore plot the relative error with respect to time. We see that Preconditioned Conjugate Gradients runs much faster in this case than a sparse direct solver. For both methods, as well as with our technique, we use the Eigen library, running on a single core of Intel i7-3720QM CPU at 2.60GHz.

While block coordinate descent cannot compete with the quadratically convergent stage of Newton's method, we notice that our approach outperforms Newton's method in its first (damped) phase. In other words, Newton's method becomes more effective only when the current iterate $\mathbf{x}_i$ is already close to the solution $\mathbf{x}^*$. If an exact solution is desired, our technique can be useful for quickly calculating a good starting point for Newton's method.



	Iteration Count	Time	Relative Error
(a)	1	5.4ms	3.61×10^{-1}
(b)	10	50.6ms	1.96×10^{-1}
(c)	100	501ms	4.02×10^{-2}
(d)	1000	5.05s	2.98×10^{-4}

Figure 5.1: Comparison of relative error vs. iteration count (left) does not reflect the cost of each iteration. Right we plot the relative error vs. computation time. In both graphs we focus on one time step of our curtain-swinging animation at the depicted frame.

The main practical benefit of our method stems from the fact that exact solution is rarely required in physics-based animation. Indeed, previous methods [Baraff and Witkin, 1998] limit the number of iterations of Newton's method to one. To experimentally evaluate the effect of approximate solutions, we tested our method on a simple animation sequence simulated with our method using 1, 10, 100, and 1000 iterations of the local/global solver. One iteration produces a stable and plausible simulation, but the wrinkles look a bit inflexible. Ten iterations seem to offer the best trade-off between speed and quality. In our example frame (Figure 5.1), ten iterations of our method achieve better relative error than one iteration of Newton's method (0.196, vs. 0.2496 for Newton) as well as faster run-time (50.6ms, vs. 181ms for one iteration of Newton with PCG). With a hundred or a thousand iterations it is difficult to tell the difference from an exact solution.

Quick approximate simulation can be achieved also using Position Based Dynamics

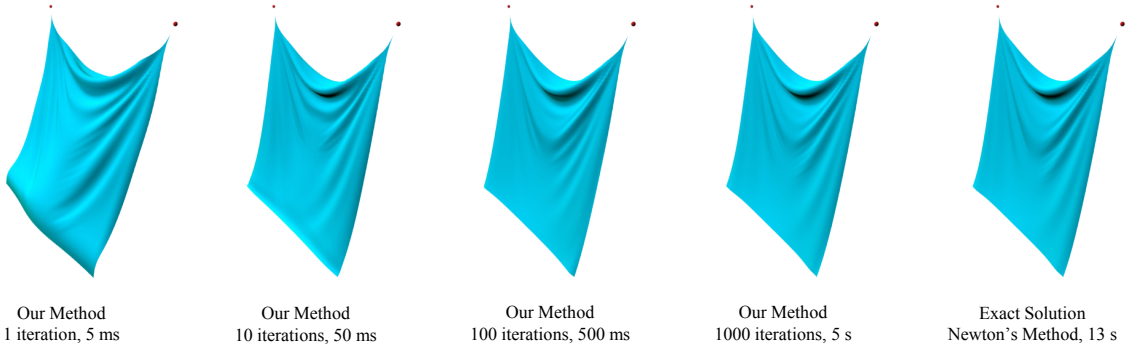


Figure 5.2: One example frame from our cloth animation simulated using our method with 1, 10, 100, and 1000 iterations of our local/global solver. Exact solution computed using Newton's method is shown for comparison.

(PBD) [Müller et al., 2007]. One problem with PBD is that its stiffness parameters are not compatible with the standard Hookean model. We tried to carefully tune the PBD parameters to get behavior as close as possible to our settings. Unfortunately, even though the PBD solver adjusts its parameters according to the number of iterations, increasing the number of iterations still increases the effective stiffness of the system. Our method does not suffer from this problem and converges to the exact implicit Euler solution.

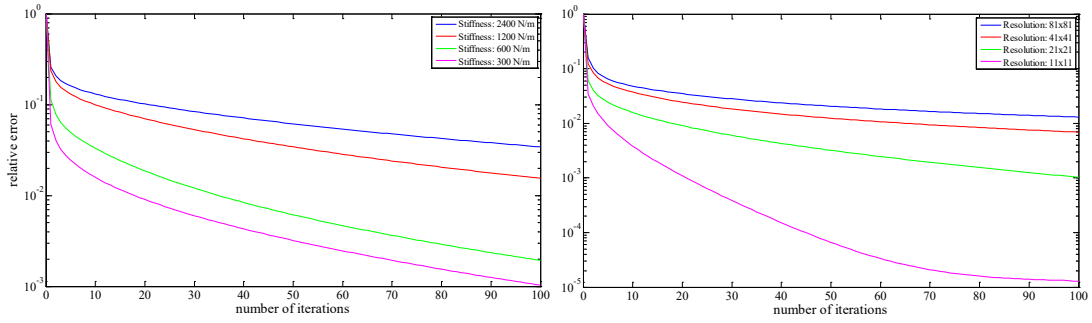


Figure 5.3: Convergence of our method for varying spring stiffness coefficients (left) and varying spatial resolution (right).

We designed the following experiments to analyze the convergence of our method. In

the first experiment, we test how spring stiffness affects the convergence, using our curtain model with 441 vertices. Figure 5.3 reports the relative error from Eq. 5.10 averaged over 50 simulation frames. As expected, higher stiffness leads to slower convergence. In the next experiment, we study the effect of varying spatial resolution using a curtain model with fixed dimension (1m $\times$ 1m) and mass (1kg). To achieve resolution independent material behavior, the spring stiffness is proportional to the resolution, i.e., when we double the resolution, we divide the spring lengths by two and multiply their stiffness by two. In all experiments we observe that while our method proceeds very quickly early on, subsequent iterations are less effective in reducing the error. Therefore, if exact results are desired, we do not advise iterating our method until convergence but instead recommend switching to Newton's method.

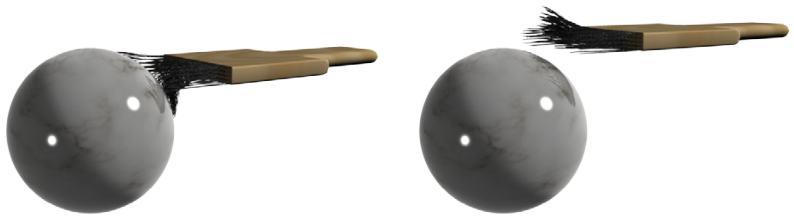


Figure 5.4: Bristles of a brush colliding with a rigid object and each other.

Collision handling is an important aspect of physics-based animation. We designed two experiments to test the behavior of our method in scenarios rich in contact and collisions. The first test involves a brush model, where individual strands collide not only with a static rigid object, but also with each other (self-collisions), see Figure 5.4. Our second example is a dancing clothed character (Figure 5.5), leveraging the publicly available models and code from ARCSim [Narain et al., 2012]. In the clothing example, 20 iterations of our method take a total time of 12.2ms. ARCSim's collision detection and response, executed once per frame, takes 476ms and therefore presents the bottleneck. We conclude that our method is best suited for real-time applications where approximate collision handling is sufficient. Note that adaptive remeshing would have invalidated our pre-factored system

matrices, so we disable the adaptive remeshing functions of ARCSim.

Figure 5.5: Character clothing with continuous collision detection, including both cloth-body and cloth-cloth collisions.

Chapter 6

Projective Dynamics

Finite element discretizations of continuum mechanics formulations allow us to simulate more accurate and sophisticated phenomena with complex non-linear materials. In this chapter, we show that for energy potentials with a specific structure we can generalize the fast numerical solution from mass-spring discretization to finite element method (FEM) as well.

6.1 Reformulating FEM Potential

The biggest takeaways from our spring energy reformulation are: 1. we represent the state of a spring using the *difference* between its two endpoints instead of their absolute positions, giving us a nice property that the translational component of the motion is automatically removed from the simulation; 2. we rewrite the original non-convex spring potential into a pure quadratic form, representing the squared distance between the spring state $\mathbf{x}_1 - \mathbf{x}_2$ and its corresponding projection, leaving all the non-convex components in the projection vector $\mathbf{p}$ which can be computed in parallel in the local step.

Notice that $\mathbf{x}_1 - \mathbf{x}_2$ is nothing but a deformation descriptor for a spring: it encodes the information of the rotation and current length of a spring regardless of the translation of its endpoints. Under the linear element assumptions (Eq. 3.11), the deformation gradient

$\mathbf{F}$ of an element plays a similar role too. The deformation gradient represents the current rotation and deformation of an element, ignoring the global translational changes of the vertices.

Like the spring potential described using Hooke's law, we treat our special FEM potential as a convex quadratic distance measure in order to further accelerate it:

$$\Psi(\mathbf{F}(\mathbf{x})) = \min_{\mathbf{p}\in\mathbb{M}} \frac{1}{2} k ||\mathbf{F} - \mathbf{p}||_F^2 \tag{6.1}$$

where k is the stiffness parameter, $\mathbf{F}$ is the deformation gradient of an element, and the subscript $_F$ denotes for Frobenius norm. $\mathbf{p}$ is a matrix that belongs to a constrained manifold $\mathbb{M}$ which depends on the visual effects we want the material to behave. For instance, $\mathbb{M}$ can be $SO(3)$ to enforce an as-rigid-as-possible [Chao et al., 2010] behavior of an element. Similarly, with our mass-spring system reformulation, the analytical solution of $\mathbf{p}$ can be seen as the projection of the deformation gradient $\mathbf{F}$.

Like Eq. 5.2, we can sum up the contribution of all elements and define the total elastic energy as:

$$E(\mathbf{x}) = \frac{1}{2}\sum_{j=1}^{m} \min_{\mathbf{p}_j\in\mathbb{M}_j} V_j k_j ||\mathbf{F}_j - \mathbf{p}_j||_F^2 = \frac{1}{2}\sum_{j=1}^{m} \min_{\mathbf{p}_j\in\mathbb{M}_j} V_j k_j ||\mathbf{F}_j^\mathsf{T} - \mathbf{p}_j^\mathsf{T}||_F^2 \tag{6.2}$$

where V_j is the rest-pose volume of the j-th element and k_j is the material stiffness. We intentionally put the transposition in the energy representation, in order to further simplify it. We know that $\mathbf{F}_j(\mathbf{x})^\mathsf{T} = \mathbf{D}_{mj}^{-\mathsf{T}}\mathbf{D}_{sj}(\mathbf{x})^\mathsf{T}$, where $\mathbf{D}_{sj}^\mathsf{T}$ can be represented as a linear combination of the position vector $\mathbf{x}$ as $\mathbf{D}_{sj}^\mathsf{T} = \mathbf{A}_j\mathbf{x}$, and $\mathbf{A}_j \in \mathbb{R}^{3\times n}$ is special selection matrix:

$$\mathbf{A}_j = \begin{bmatrix} \dots & 1 & \dots & 0 & \dots & 0 & \dots & -1 & \dots \\ \dots & 0 & \dots & 1 & \dots & 0 & \dots & -1 & \dots \\ \dots & 0 & \dots & 0 & \dots & 1 & \dots & -1 & \dots \end{bmatrix} \tag{6.3}$$

where the $[1;0;0]$, $[0;1;0]$ and $[0;0;1]$ columns are corresponding to the indices of the first, second and third vertex of the j-th element, and the $[-1;-1;-1]$ column is corresponding

to the index of the last vertex of the j-th element. The rest of the A_j matrix is filled with zeros. Therefore we have $\mathbf{A}_j\mathbf{x} = [\mathbf{x}_{j1}^\mathsf{T} - \mathbf{x}_{j4}^\mathsf{T}; \mathbf{x}_{j2}^\mathsf{T} - \mathbf{x}_{j4}^\mathsf{T}; \mathbf{x}_{j3}^\mathsf{T} - \mathbf{x}_{j4}^\mathsf{T}] = \mathbf{D}_{sj}^\mathsf{T}$. Similarly to the mass-spring system representation, if we define a stacked matrix $\mathbf{p} = [\mathbf{p}_1^\mathsf{T}; \mathbf{p}_2^\mathsf{T}; \ldots; \mathbf{p}_m^\mathsf{T}] \in \mathbb{R}^{3m\times 3}$, a spectial indicator matrix $\mathbf{S}_j = \delta_j \otimes \mathbf{I}_{33} \in \mathbb{R}^{3m\times 3}$ (where $\mathbf{I}_{33}$ is a 3×3 identity matrix and $\otimes$ denotes for Kronecker product) and a differential operator $\mathbf{G}_j = \left(\mathbf{D}_{mj}^{-\mathsf{T}}\mathbf{A}_j\right)^\mathsf{T} \in \mathbb{R}^{n\times 3}$, we can rewrite the elastic energy in Eq. 6.2 as:

$$E(\mathbf{x}) = \frac{1}{2}\sum_{j=1}^{m} \min_{\mathbf{p}_j\in\mathbb{M}_j} V_j k_j ||\mathbf{G}_j^\mathsf{T}\mathbf{x} - \mathbf{S}_j^\mathsf{T}\mathbf{p}||_F^2 \tag{6.4}$$

Once we define two constant matrices $\mathbf{L}$ and $\mathbf{J}$ as: $\mathbf{L} = \sum_{j=1}^{m} V_j k_j \mathbf{G}_j\mathbf{G}_j^\mathsf{T} \in \mathbb{R}^{n\times n}$ and $\mathbf{J} = \sum_{j=1}^{m} V_j k_j \mathbf{G}_j\mathbf{S}_j^\mathsf{T} \in \mathbb{R}^{n\times 3m}$, we can further simplify Eq. 6.4 as:

$$E(\mathbf{x}) = \min_{\mathbf{p}\in\mathbb{M}} \frac{1}{2}\,\mathrm{tr}\left(\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x}\right) - \mathrm{tr}\left(\mathbf{x}^\mathsf{T}\mathbf{J}\mathbf{p}\right) + C \tag{6.5}$$

Thanks to the reuse of the operators $\mathbf{G}$, $\mathbf{S}$ and the projection matrix $\mathbf{p}$, Eq. 6.5 looks identical with Eq. 5.6. Despite the overloading of $\mathbf{G}$, $\mathbf{S}$ and $\mathbf{p}$, the major difference between Eq. 6.5 and Eq. 5.6 is the constrained manifold $\mathbb{M}$. The definition of $\mathbb{M}$ varies in different scenarios. It can be $\mathbb{M} = \{(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m) : \mathbf{p}_j \in SO(3)\}$ to enforce the as-rigid-as-possible behavior of the elastic body; or it can be $\mathbb{M} = \{(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m) : \mathbf{p}_j \in SL(3)\}$ to preserve the volume of each element as much as possible, where $SL(3)$ is a special linear group of 3×3 matrices with determinant 1.

We can substitute Eq. 6.5 into Eq. 3.30 to write down the final constrained optimization to simulate finite element models in 3D:

$$\min_{\mathbf{x}\in\mathbb{R}^{n\times 3}, \mathbf{p}\in\mathbb{M}} \left(\frac{1}{2h^2}\,\mathrm{tr}\left((\mathbf{x}-\mathbf{y})^\mathsf{T}\mathbf{M}(\mathbf{x}-\mathbf{y})\right) + \frac{1}{2}\,\mathrm{tr}\left(\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x}\right) - \mathrm{tr}\left(\mathbf{x}^\mathsf{T}\mathbf{J}\mathbf{p}\right) + C\right) \tag{6.6}$$

6.2 Numerical Solution

Like the proposed fast-mass spring simulation method, we can apply the local-global optimization method to optimize Eq. 6.6. The global step is exactly the same with the

mass-spring system:

$$\mathbf{x}^* = \left(\frac{M}{h^2} + \mathbf{L}\right)^{-1} \left(\frac{M}{h^2}\mathbf{y} + \mathbf{J}\mathbf{p}\right) \tag{6.7}$$

where the matrix $\frac{M}{h^2} + \mathbf{L}$ is only dependent on the mesh topology, the rest pose and stiffness of all elements, the time step of the simulation and the mass for all points. We pre-compute this system matrix and its Cholesky factorization, so that during each global step, only a forward-backward substitution is needed to solve the linear system.

The local step which is the step to find the projections really depends on specific material models. Unlike the mass-spring system, the local step for FEM simulations usually does not have a closed-form solution because the non-convexity in most of the FEM problems is more complicated.

For example, if we want to enforce the change of any deformation to be less than a certain threshold, we can run a signed singular value decomposition (SSVD) [Sorkine, 2009] on the deformation gradient: $\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$ and set the projection to be $\mathbf{p} = \mathbf{U}\tilde{\mathbf{\Sigma}}\mathbf{V}^\mathsf{T}$, where $\tilde{\mathbf{\Sigma}}$ is clamped from $\mathbf{\Sigma}$ by some user threshold $\sigma_{min} \leq \tilde{\mathbf{\Sigma}}_{ii} \leq \sigma_{max}$. If we set $\sigma_{min} = \sigma_{max} = 1$ it will enforce the projection to be rotation mode only: $\mathbf{p} \in SO(3)$. Figure 6.1 shows the results of a cloth simulation with different σ_{min} and σ_{max} choices.

Another interesting projection is to project the deformation gradient onto $SL(3)$. We can start from the SSVD of $\mathbf{F}$ as well, and formulate a non-linear local problem as:

$$\begin{aligned} \min_{\mathbf{d}} \quad & ||\mathbf{d}||^2 \\ \text{s.t.} \quad & c(\mathbf{d}) = 0 \end{aligned}$$

where $c(\mathbf{d}) = \prod_{i=1}^{3}(\mathbf{\Sigma}_{ii} + \mathbf{d}_i) - 1$ and the unknown variable $\mathbf{d}$ is a $3 \times 1 vector$. This little optimization problem can be solved by any iterative quadratic programming method. Since it is in the local step, we can run it for all the elements in parallel. After the optimization is solved, we can use $\mathbf{p} = \mathbf{U}(\mathbf{\Sigma} + diag(\mathbf{d}))\mathbf{V}^\mathsf{T}$ as the projection of $\mathbf{F}$ on $SL(3)$. The effects of the volume constraints can be seen if Figure 6.2.

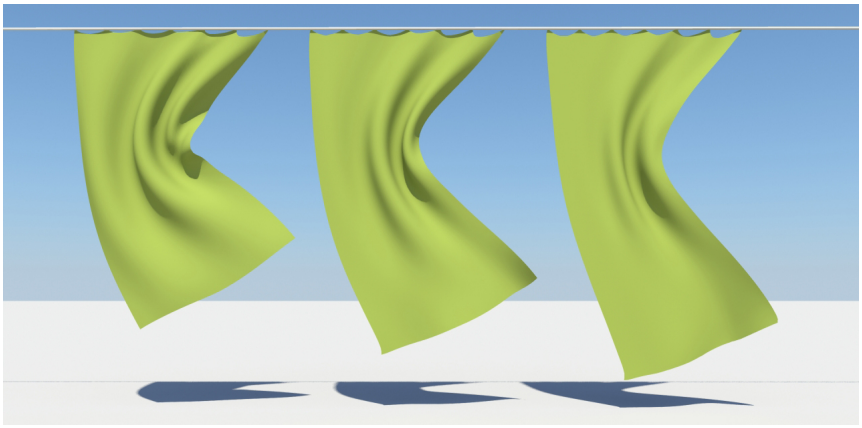


Figure 6.1: Starting from the same mesh, strain limiting allows simulating material that can undergo small to moderate amount of stretching. From left to right, we use strain limits [σ_{min}, σ_{max}] of [90%, 110%], [80%, 120%] and [70%, 130%]. Notice how the cloth stretches and how the folds get absorbed when the limit increases.

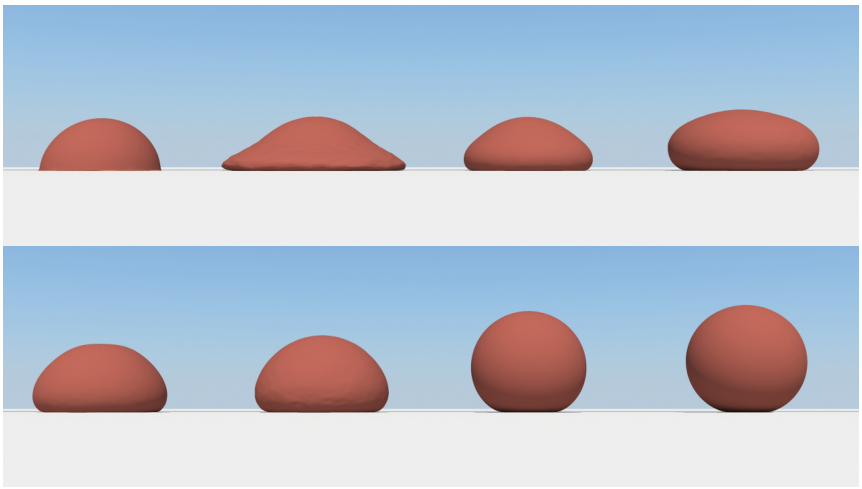


Figure 6.2: Varying weight combinations of volume preservation and strain constraints allow the simulation of different types of materials for volumetric objects. The top row shows the effect of increasing the weight of the strain constraints from left to right; the bottom row shows the effect of increasing the weight of the volume preservation constraints from left to right.

Chapter 7

Quasi-Newton Methods for Real-time Simulation of Hyperelastic Materials

Currently, our method works only on a special type of elastic models Eq. 6.1 – a squared distance function between an element's deformation descriptor and its projection. However, most of the elastic material models used by continuum mechanics (such as St. Venant-Kirchhoff and Neo-Hookean models, or even as simple as a co-rotated linear elasticity model) can not be represented in this way. In order to support more general hyperelastic materials, we start from a reformulation of our existing methods.

7.1 Quasi-Newton Interpretation

Let us now describe how our existing methods can be interpreted as a quasi-Newton method. Note that Eq. 5.8 and Eq. 6.6 will converge to the exact solution of an implicit Euler integration using their corresponding material model, the auxiliary projection variable $\mathbf{p}$ is introduced only for the local-global optimization scheme. Instead of being used as an auxiliary variable, $\mathbf{p}$ can be also seen as an implicit function on $\mathbf{x}$ that can minimize Eq. 6.6.

In that case, we can rewrite the objective of our variational implicit Euler as following:

$$g(\mathbf{x}) = \frac{1}{2h^2}\,\mathrm{tr}\left((\mathbf{x}-\mathbf{y})^\mathsf{T}\mathbf{M}(\mathbf{x}-\mathbf{y})\right) + \frac{1}{2}\,\mathrm{tr}\left(\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x}\right) - \mathrm{tr}\left(\mathbf{x}^\mathsf{T}\mathbf{J}\mathbf{p}(\mathbf{x})\right) + \frac{1}{2}\,\mathrm{tr}\left(\mathbf{p}(\mathbf{x})^\mathsf{T}\mathbf{S}\mathbf{p}(\mathbf{x})\right) \tag{7.1}$$

where $\mathbf{p}(\mathbf{x})$ is an implicit projection function that maps from $\mathbf{x} \in \mathbb{R}^{n\times 3}$ to $\mathbb{R}^{3m\times 3}$ for 3D finite element models or $\mathbb{R}^{m\times 3}$ for mass-spring systems. Note that the previous constant term C is not negligible anymore, and appears as $\frac{1}{2}\,\mathrm{tr}\left(\mathbf{p}(\mathbf{x})^\mathsf{T}\mathbf{S}\mathbf{p}(\mathbf{x})\right)$ in this optimization, where $S = \sum_{j=1}^{m} V_j k_j \mathbf{S}_j \mathbf{S}_j^\mathsf{T}$, $\mathbf{S}_j$ is defined in Eq. 5.4 and Eq. 6.4. By definition, $\mathbf{p}(\mathbf{x})$ can be seen as a stacked function from all elements: $\mathbf{p}(\mathbf{x}) = [\mathbf{p}_1(\mathbf{x})^\mathsf{T}; \mathbf{p}_2(\mathbf{x})^\mathsf{T}; \ldots; \mathbf{p}_m(\mathbf{x})^\mathsf{T}]$ and for the j-th element, the implicit projection function $\mathbf{p}_j(\mathbf{x})$ can be seen as:

$$\mathbf{p}_j(\mathbf{x}) = \underset{\mathbf{p}_j \in \mathbb{M}_j}{\mathrm{argmin}}\ ||\mathbf{G}_j\mathbf{x} - \mathbf{p}_j||^2 \tag{7.2}$$

Since now our objective goes back to an unconstrained minimization problem with a single variable $\mathbf{x}$, we can first compute the gradient of $g(\mathbf{x})$:

$$\nabla g(\mathbf{x}) = \frac{1}{h^2}\mathbf{M}(\mathbf{x}-\mathbf{y}) + \mathbf{L}\mathbf{x} - \mathbf{J}\mathbf{p}(\mathbf{x}) + \frac{\partial \mathbf{p}(\mathbf{x})}{\partial \mathbf{x}} : (\mathbf{S}\mathbf{p}(\mathbf{x}) - \mathbf{J}^\mathsf{T}\mathbf{x}) \tag{7.3}$$

where the last term $\frac{\partial \mathbf{p}(\mathbf{x})}{\partial \mathbf{x}} : (\mathbf{S}\mathbf{p}(\mathbf{x}) - \mathbf{J}^\mathsf{T}\mathbf{x})$ comes from the derivative of $\mathbf{p}(\mathbf{x})$ by applying the chain rule. $\frac{\partial \mathbf{p}(\mathbf{x})}{\partial \mathbf{x}}$ is a $3m \times 3 \times n \times 3$ tensor in FEM cases and an $m \times 3 \times n \times 3$ tensor in mass-spring cases. The overall gradient $\nabla g(\mathbf{x})$ is an $n \times 3$ matrix, where the trace operator nicely goes away due to its linear property.

Although $\frac{\partial \mathbf{p}(\mathbf{x})}{\partial \mathbf{x}}$ is an ugly tensor and is hard to compute for arbitrary projections. However, as long as $\mathbf{p}(\mathbf{x})$ is a projection function, the term $\frac{\partial \mathbf{p}(\mathbf{x})}{\partial \mathbf{x}} : (\mathbf{S}\mathbf{p}(\mathbf{x}) - \mathbf{J}^\mathsf{T}\mathbf{x})$ is always zero.

Proof. Because the total potential energy is the summation of the individual potential

energy contributions from all elements. we cam rewrite $\frac{\partial \mathbf{p}(\mathbf{x})}{\partial \mathbf{x}}$ as:

$$\begin{aligned}
\frac{\partial \mathbf{p}(\mathbf{x})}{\partial \mathbf{x}} : (\mathbf{S}\mathbf{p}(\mathbf{x}) - \mathbf{J}^\mathsf{T}\mathbf{x}) &= \frac{\partial \mathbf{p}(\mathbf{x})}{\partial \mathbf{x}} : \sum_{j=1}^{m} V_j k_j (\mathbf{S}_j \mathbf{S}_j^\mathsf{T} \mathbf{p}(\mathbf{x}) - \mathbf{S}_j \mathbf{G}_j^\mathsf{T} \mathbf{x}) && (7.4)\\
&= \sum_{j=1}^{m} V_j k_j \mathbf{S}_j^\mathsf{T} \frac{\partial \mathbf{p}(\mathbf{x})}{\partial \mathbf{x}} : (\mathbf{S}_j^\mathsf{T} \mathbf{p}(\mathbf{x}) - \mathbf{G}_j^\mathsf{T} \mathbf{x}) && (7.5)\\
&= \sum_{j=1}^{m} V_j k_j \frac{\partial \mathbf{p}_j(\mathbf{x})}{\partial \mathbf{x}} : (\mathbf{S}_j^\mathsf{T} \mathbf{p}(\mathbf{x}) - \mathbf{G}_j^\mathsf{T} \mathbf{x}) && (7.6)
\end{aligned}$$

Note that $\mathbf{S}_j$ is a selection matrix for the j-th constraint, $\frac{\partial \mathbf{p}_j(\mathbf{x})}{\partial \mathbf{x}} = \mathbf{S}_j^\mathsf{T} \frac{\partial \mathbf{p}(\mathbf{x})}{\partial \mathbf{x}}$ simply extracts the components of $\frac{\partial \mathbf{p}(\mathbf{x})}{\partial \mathbf{x}}$ contributed by the j-th element, forming a smaller $3 \times 3 \times n \times 3$ tensor. And by definition $\mathbf{G}_j\mathbf{x}$ is the deformation descriptor for the j-th element: it is $\mathbf{x}_{j1}^\mathsf{T} - \mathbf{x}_{j2}^\mathsf{T}$ for a mass-spring system and is $\mathbf{F}_j^\mathsf{T}$ for an FEM case; $\mathbf{S}_j\mathbf{p}$ is the transpose of the projection of the j-th element: $\mathbf{S}_j\mathbf{p} = \mathbf{p}_j^\mathsf{T}$. Therefore $(\mathbf{S}_j^\mathsf{T}\mathbf{p}(\mathbf{x}) - \mathbf{G}_j^\mathsf{T}\mathbf{x})$ simply means the difference between an element's deformation descriptor and its projection. Given that $\frac{\partial \mathbf{p}_j(\mathbf{x})}{\partial \mathbf{x}}$ is a tensor that can only slide on the tangent space of $\mathbb{M}_j$ by definition of $\mathbf{p}_j(\mathbf{x})$ described in Eq. 7.2, the contraction between $\frac{\partial \mathbf{p}_j(\mathbf{x})}{\partial \mathbf{x}}$ and $(\mathbf{S}_j^\mathsf{T}\mathbf{p}(\mathbf{x}) - \mathbf{G}_j^\mathsf{T}\mathbf{x})$ will vanish (like the dot-product between two orthogonal vectors):

$$\frac{\partial \mathbf{p}_j(\mathbf{x})}{\partial \mathbf{x}} : (\mathbf{S}_j^\mathsf{T}\mathbf{p}(\mathbf{x}) - \mathbf{G}_j^\mathsf{T}\mathbf{x}) = \mathbf{0}, \text{for } \forall j \in [1, m] \tag{7.7}$$

Substituting Eq. 7.7 into Eq. 7.6 will give us:

$$\frac{\partial \mathbf{p}(\mathbf{x})}{\partial \mathbf{x}} : (\mathbf{S}\mathbf{p}(\mathbf{x}) - \mathbf{J}^\mathsf{T}\mathbf{x}) = \mathbf{0} \tag{7.8}$$

which ends the proof. □

Substituting Eq. 7.8 into Eq. 7.3 will get us a much simpler representation of the gradient of our objective function:

$$\nabla g(\mathbf{x}) = \frac{1}{h^2}\mathbf{M}(\mathbf{x} - \mathbf{y}) + \mathbf{L}\mathbf{x} - \mathbf{J}\mathbf{p}(\mathbf{x}) \tag{7.9}$$

This gradient is clear enough as if the projection function $\mathbf{p}(\mathbf{x})$ is independent of $\mathbf{x}$, similarly to what we did in the local-global optimization , where the projection matrix is

treated as constant in the global step. Newton's method will proceed by computing the second order derivative of $g(\mathbf{x})$, i.e. the Hessian matrix $\nabla^2 g(\mathbf{x}) = \frac{\mathbf{M}}{h^2} + \nabla^2 E(\mathbf{x})$, and use it to compute a descent direction; $\delta \mathbf{x}_{Newton} = -\left(\nabla^2 g(\mathbf{x})\right)^{-1} \nabla g(\mathbf{x})$. Some extra safeguard approaches (described in section Section 4.2.2) might be needed too, to ensure $\delta \mathbf{x}_{Newton}$ is always a descent direction.

Instead of applying the inverse of Hessian matrix to get the descent direction, what will happen if we apply the inverse of $\frac{\mathbf{M}}{h^2} + \mathbf{L}$ to the negative gradient? Simple algebra reveals:

$$-\left(\frac{\mathbf{M}}{h^2} + \mathbf{L}\right)^{-1} \nabla g(\mathbf{x}) = \left(\frac{M}{h^2} + \mathbf{L}\right)^{-1} \left(\frac{M}{h^2}\mathbf{y} + \mathbf{J}\mathbf{p}(\mathbf{x})\right) - \mathbf{x} \tag{7.10}$$

where the first term on the right hand side $\left(\frac{M}{h^2} + \mathbf{L}\right)^{-1} \left(\frac{M}{h^2}\mathbf{y} + \mathbf{J}\mathbf{p}(\mathbf{x})\right)$ is exactly the result $\mathbf{x}^*$ of one iteration of the local-global optimization, see Eq. 5.9 and Eq. 6.7. Therefore, we can rearrange Eq. 7.10 to get:

$$\mathbf{x}^* = \mathbf{x} - \left(\frac{\mathbf{M}}{h^2} + \mathbf{L}\right)^{-1} \nabla g(\mathbf{x}) \tag{7.11}$$

where we can define $\delta \mathbf{x} = -\left(\frac{\mathbf{M}}{h^2} + \mathbf{L}\right)^{-1} \nabla g(\mathbf{x})$ as a *descent direction*. It is guaranteed to be a descent direction without any definiteness treatments because $\frac{\mathbf{M}}{h^2} + L$ is by definition a positive definite matrix. Once we understand one local-global iteration as an update based on Eq. 7.11, we can treat our methods in simulating mass-spring systems and FEM models as a special *quasi-Newton* method. It is special because a typical quasi-Newton method uses line search techniques [Nocedal and Wright, 2006] to find a parameter α such that $\mathbf{x} + \alpha\delta\mathbf{x}$ reduces the objective sufficiently. However, with the special type of energy Eq. 6.1, the optimal value of α is always 1 when run our quasi-Newton method.

This quasi-Newton interpretation suggests that a similar optimization strategy might be effective for more general elastic potential energies. We will first focus on isotropic elastic materials, deferring the discussion of anisotropic materials. The assumption of isotropy (material-space rotation invariance) together with world-space rotation invariance means that we can express elastic energy density function Ψ as a function of singular values of the deformation gradient [Irving et al., 2004, Sifakis and Barbic, 2012]. In the volumetric

case, we have three singular values $\sigma_1, \sigma_2, \sigma_3 \in \mathbb{R}$, also known as "principal stretches". The function $\Psi(\sigma_1, \sigma_2, \sigma_3)$ must be invariant to any permutation of the principal stretches, e.g., $\Psi(\sigma_1, \sigma_2, \sigma_3) = \Psi(\sigma_2, \sigma_1, \sigma_3)$ etc. Because directly working with such functions Ψ could be cumbersome, we instead use the Valanis-Landel hypothesis [Valanis and Landel, 1967], which assumes that:

$$\begin{aligned}\Psi(\sigma_1, \sigma_2, \sigma_3) = a(\sigma_1) + a(\sigma_2) + a(\sigma_3) + \\ b(\sigma_1\sigma_2) + b(\sigma_2\sigma_3) + b(\sigma_1\sigma_3) + c(\sigma_1\sigma_2\sigma_3),\end{aligned} \tag{7.12}$$

where $a, b, c : \mathbb{R} \to \mathbb{R}$. Many material models can be written in the Valanis-Landel form, including linear corotated material [Sifakis and Barbic, 2012], St. Venant-Kirchhoff, Neo-Hookean, and Mooney-Rivlin. The recently proposed spline-based materials [Xu et al., 2015] are also based on the Valanis-Landel assumption. How can we simulate these types of materials efficiently? Invoking the quasi-Newton interpretation discussed above, our method will minimize the objective g by performing descent along direction $\delta\mathbf{x} := -(\mathbf{M}/h^2 + \mathbf{L})^{-1}\nabla g(\mathbf{x})$. The mass matrix $\mathbf{M}$ and step size h are defined as before, and computing $\nabla g(\mathbf{x})$ is straightforward. But how to define a matrix $\mathbf{L}$ for a given material model? This matrix can still have the form $\mathbf{L} := \sum w_j \mathbf{G}_j \mathbf{G}_j^\mathsf{T}$, but the question is how to choose the weights w_j. Previously, we assumed the weights are given as $w_j = V_j k_j$, where $V_j > 0$ is rest-pose volume of j-th element, and $k_j > 0$ is a stiffness parameter provided by the user. In our case, the user instead specifies a material model according to Eq. 7.12 from which we have to infer the appropriate k_j value. In the following we drop the subscript j for ease of notation.

For linear materials (Hooke's law), stiffness is given as the second derivative of elastic energy. Therefore, it would be tempting to set k equal to the second derivative of Ψ at the rest pose (corresponding to $\sigma_1 = \sigma_2 = \sigma_3 = 1$), which evaluates to $a''(1) + 2b''(1) + c''(1)$, regardless of whether we differentiate with respect to σ_1, σ_2, or σ_3. Even though this method would produce suitable k for some materials (such as corotated elasticity), it does not work e.g. for a polynomial material defined as $a(x) = \mu(x-1)^4, b(x) = 0, c(x) = 0$. With this material, the second derivatives at $x = 1$ evaluate to zero regardless of the value of μ, which

would lead to zero stiffness which is obviously not a good approximation. The problem is the second derivative takes into account only infinitesimally small neighborhood of $x = 1$, i.e., the rest pose. However, we need a *single* value of k which will work well in the entire range of deformations expected in our simulations. To capture this requirement, we define an interval $[x_{\text{start}}, x_{\text{end}}]$ where we expect our principal stretches to be. We consider the following stress function:

$$f(\sigma_1) = \left.\frac{\partial \Psi}{\partial \sigma_1}\right|_{\sigma_2=1,\sigma_3=1} = a'(\sigma_1) + 2b'(\sigma_1) + c'(\sigma_1), \tag{7.13}$$

and define our k as the slope of the best linear approximation of Eq. 7.13 at $[x_{\text{start}}, x_{\text{end}}]$. Formally:

$$k := \operatorname*{argmin}_k \int_{x_{\text{start}}}^{x_{\text{end}}} (k(x-1) - f(x))^2 dx. \tag{7.14}$$

Note that due to the symmetry of the Valanis-Landel assumption, we would obtain exactly the same result if we differentiated with respect to σ_2 or σ_3 (instead of σ_1 as above). We study different choices of $[x_{\text{start}}, x_{\text{end}}]$ intervals and in summary, the results are not very sensitive on the particular choice of x_{start} and x_{end}. The key fact is that regardless of the specific setting of x_{start} and x_{end}, spatial variations of μ are correctly taken into account, i.e., softer and stiffer parts of the simulated object will have different μ coefficients. Even though all elements have the same $[x_{\text{start}}, x_{\text{end}}]$ interval, the resulting matrices $\mathbf{L}$ and $\mathbf{J}$ properly reflect the spatially varying stiffness.

In our previous methods using Eq. 6.1, the line search parameter $\alpha = 1$ is always guaranteed to decrease the objective $g(\mathbf{x})$. Unfortunately, this is no longer true in our generalized quasi-Newton setting, where it is easy to find examples where $g(\mathbf{x} + \delta\mathbf{x}) > g(\mathbf{x})$, i.e., taking a step of size one actually *increases* the objective. This can lead to erroneous energy accumulation, potentially resulting in catastrophic failure of the simulation ("explosions"). Fortunately, thanks to the fact that $\mathbf{M}/h^2 + \mathbf{L}$ is positive definite, $\delta\mathbf{x}$ is guaranteed to be a descent direction. Therefore, there exists $\alpha > 0$ such that $g(\mathbf{x} + \alpha\delta\mathbf{x}) \leq g(\mathbf{x})$ (unless we are already at a critical point $\nabla g(\mathbf{x}) = \mathbf{0}$, at which point the optimization is finished). In fact, we can ask for even more, i.e., we can always find $\alpha > 0$ such that

$g(\mathbf{x} + \alpha\delta\mathbf{x}) \leq g(\mathbf{x}) + \gamma\alpha \operatorname{tr}((\nabla g(\mathbf{x}))^{\mathsf{T}}\delta\mathbf{x})$ for some constant $\gamma \in (0, 0.5)$. This is known as the Armijo condition which expresses the requirement of "sufficient decrease" [Nocedal and Wright, 2006], preventing the line search algorithm from reducing the objective only by a negligible amount. Another requirement for robust line search is to avoid too small steps α even though they might satisfy the Armijo condition.

We tested two possible strategies: 1) backtracking line search algorithm that satisfies only the Armijo condition, and 2) line search algorithm that satisfies both the Armijo condition and the "curvature condition" (collectively known as "Wolfe conditions"). In summary, we found that both methods lead to comparable error reduction, but the backtracking line search is faster. In our final algorithm, we, therefore, use the backtracking line search. Specifically, we set the initial α to 1 and multiply it by $\beta \in (0, 1)$ after every failed attempt. For most of our test cases, we experienced that $\beta = 0.5$, which means each failed line search attempt will reduce the step size by half, works well in practice.

Alg. 3 summarizes the process of computing one frame of our simulation. Similarly to Alg. 2, the outer loop performs quasi-Newton iterations and the inner loop implements the line search.

Indeed there is no "free lunch" to generalize our previous methods to support arbitrary hyperelastic materials. With energies using quadratic distance functions (Eq. 6.1), we do not need the line search, because $\alpha = 1$ always works. Indeed, if we drop the line search from Alg. 3, the algorithm becomes equivalent to a generalized local/global process, which is unstable for general material simulation purposes. Rejected line search attempts, i.e., additional iterations of the line search, represent the main computational overhead of our quasi-Newton method. Fortunately, we found that in practical simulations the number of extra line search iterations is relatively small.

Algorithm 3: Quasi-Newton Solver with Backtracking Line Search.

$\mathbf{x}^{(1)} := \mathbf{y}$;

$g(\mathbf{x}^{(1)}) := \texttt{evalObjective}(\mathbf{x}^{(1)})$

for $k = 1, \ldots,$ numIterations **do**

  $\nabla g(\mathbf{x}^{(k)}) := \texttt{evalGradient}(\mathbf{x}^{(k)})$

  $\delta\mathbf{x} := -(\mathbf{M}/h^2 + \mathbf{L})^{-1}\nabla g(\mathbf{x}^{(k)})$

  $\alpha := 1/\beta$

  repeat

    $\alpha := \beta\alpha$

    $\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \alpha\delta\mathbf{x}$

    $g(\mathbf{x}^{(k+1)}) := \texttt{evalObjective}(\mathbf{x}^{(k+1)})$

  until $g(\mathbf{x}^{(k+1)}) \leq g(\mathbf{x}^{(k)}) + \gamma\alpha \operatorname{tr}((\nabla g(\mathbf{x}^{(k)}))^{\mathsf{T}}\delta\mathbf{x})$;

end

7.2 Boosting Convergence

The quasi-Newton interpretation allows us to take advantage of further mathematical optimization techniques. In this section, we discuss how to accelerate the convergence of our method using L-BFGS (Limited-memory BFGS). The BFGS algorithm (Broyden-Fletcher-Goldfarb-Shanno) is one of the most popular general purpose quasi-Newton methods; its key idea is to approximate the Hessian using curvature information calculated from previous iterates, i.e., $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k-1)}$. The L-BFGS modification means that we will use only the most recent w iterates, i.e., $\mathbf{x}^{(k-w)}, \ldots, \mathbf{x}^{(k-1)}$; the rationale being that too distant iterates are less relevant in estimating the Hessian at $\mathbf{x}^{(k)}$.

In Alg. 3, the matrix $\mathbf{M}/h^2 + \mathbf{L}$ in line 4 can be interpreted as our initial approximation of the Hessian. This matrix is constant which on one hand enables its pre-factorization, but on the other hand, $\mathbf{M}/h^2 + \mathbf{L}$ may be far from the Hessian $\nabla^2 g(\mathbf{x}_k)$, which is the reason for slower convergence compared to Newton's method. L-BFGS allows us to develop a more accurate, state-dependent Hessian approximation, leading to faster convergence without too

much computational overhead (in our experiments the overhead is typically less than 1% of the simulation time). The key to fast iterations of L-BFGS is the fact that the progressively updated approximate Hessian $\mathbf{A}^{(k)}$ is not stored explicitly, which would require us to solve a new linear system $\mathbf{A}^{(k)}\delta\mathbf{x} = -\nabla g(\mathbf{x}^{(k)})$ each iteration, implying high computational costs. Instead, L-BFGS implicitly represents the *inverse* of $\mathbf{A}^{(k)}$, i.e., linear operator $\mathbf{B}^{(k)}$ such that the desired descent direction can be computed simply as $\delta\mathbf{x} = -\mathbf{B}^{(k)}\nabla g(\mathbf{x}^{(k)})$. The linear operator $\mathbf{B}^{(k)}$ is not represented using a matrix (which would have been dense), but instead, as a sequence of dot products, known as the L-BFGS two-loop recursion, see Alg. 4. For a more detailed discussion of BFGS and its variants, we refer to Chapters 6 and 7 of [Nocedal and Wright, 2006].

Algorithm 4: Descent direction computation with L-BFGS

$\mathbf{q} := \nabla g(\mathbf{x}^{(k)})$

for $i = k-1, \ldots, k-w$ **do**

  $\mathbf{s}^{(i)} := \mathbf{x}^{(i+1)} - \mathbf{x}^{(i)}$; $\mathbf{t}^{(i)} := \nabla g(\mathbf{x}^{(i+1)}) - \nabla g(\mathbf{x}^{(i)})$; $\rho^{(i)} := \mathrm{tr}((\mathbf{t}^{(i)})^\mathsf{T}\mathbf{s}^{(i)})$

  $\zeta^{(i)} := \mathrm{tr}((\mathbf{s}^{(i)})^\mathsf{T}\mathbf{q})/\rho^{(i)}$

  $\mathbf{q} := \mathbf{q} - \zeta^{(i)}\mathbf{t}^{(i)}$

end

$\mathbf{r} := (\mathbf{A}^{(0)})^{-1}\mathbf{q}$ // $\mathbf{A}^{(0)}$ is initial Hessian approximation

for $i = k-w, \ldots, k-1$ **do**

  $\eta := \mathrm{tr}((\mathbf{t}^{(i)})^\mathsf{T}\mathbf{r})/\rho^{(i)}$

  $\mathbf{r} := \mathbf{r} + \mathbf{s}^{(i)}(\zeta^{(i)} - \eta)$

end

$\delta\mathbf{x} := -\mathbf{r}$ // resulting descent direction

Alg. 4 requires us to provide an initial Hessian approximation $\mathbf{A}^{(0)}$, ideally such that the linear system $\mathbf{A}^{(0)}\mathbf{r} = \mathbf{q}$ can be solved efficiently. In our method, we use $\mathbf{M}/h^2 + \mathbf{L}$ as the initial Hessian approximation. At first, it may seem the initialization of the Hessian is perhaps not too important and the L-BFGS iterations quickly approach the exact Hessian. However, this intuition is not true. We experiment with different possible initializations of

the Hessian and show that our particular choice of $\mathbf{M}/h^2 + \mathbf{L}$ outperforms alternatives such as Hessian of the rest-pose and many others. In short, the reason is that the L-BFGS updates use only a few gradient samples, which provide only a limited amount of information about the exact Hessian. The appeal of the L-BFGS strategy is that it is very fast – the compute cost of the two for-loops in Alg. 4 is negligible compared to the cost of solving the linear system $\mathbf{A}^{(0)}\mathbf{r} = \mathbf{q}$ with our choice of $\mathbf{A}^{(0)} = \mathbf{M}/h^2 + \mathbf{L}$. This is true even for high values of w. In other words, the linear solve using $\mathbf{M}/h^2 + \mathbf{L}$ is still doing the "heavy lifting", while the L-BFGS updates provide additional convergence boost at the cost of minimal computational overhead.

Upgrading our method with L-BFGS is simple: we only need to replace the descent direction evaluation ($\delta\mathbf{x} := -(\mathbf{M}/h^2 + \mathbf{L})^{-1}\nabla g(\mathbf{x}^{(k)})$) in Alg. 3 with a call of Alg. 4. Note that for $w = 0$, Alg. 4 returns exactly the same descent direction as before. What is the optimal w, i.e., the size of the history window? Too small w will not allow us to unlock the full potential of L-BFGS. The main problem with too high w is not the higher computational cost of the two loops in Alg. 4, but the fact that too distant iterates (such as $\mathbf{x}^{(k-100)}$) may contain information irrelevant for the Hessian at $\mathbf{x}^{(k)}$ and the result can be even worse than with a shorter window. We found that $w = 5$ is typically a good value in our experiments.

Finally, we note that even though the Wolfe conditions are the recommended line search strategy for L-BFGS, we did not observe any significant convergence benefit compared to our backtracking strategy. However, evaluating the Wolfe conditions increases the computational cost per iteration and therefore, we continue to rely on the backtracking strategy as described in Alg. 3.

7.3 Results

Our method supports standard elastic materials, such as corotated linear elasticity, St. Venant-Kirchhoff and the Neo-Hookean model and also the spline-based material proposed

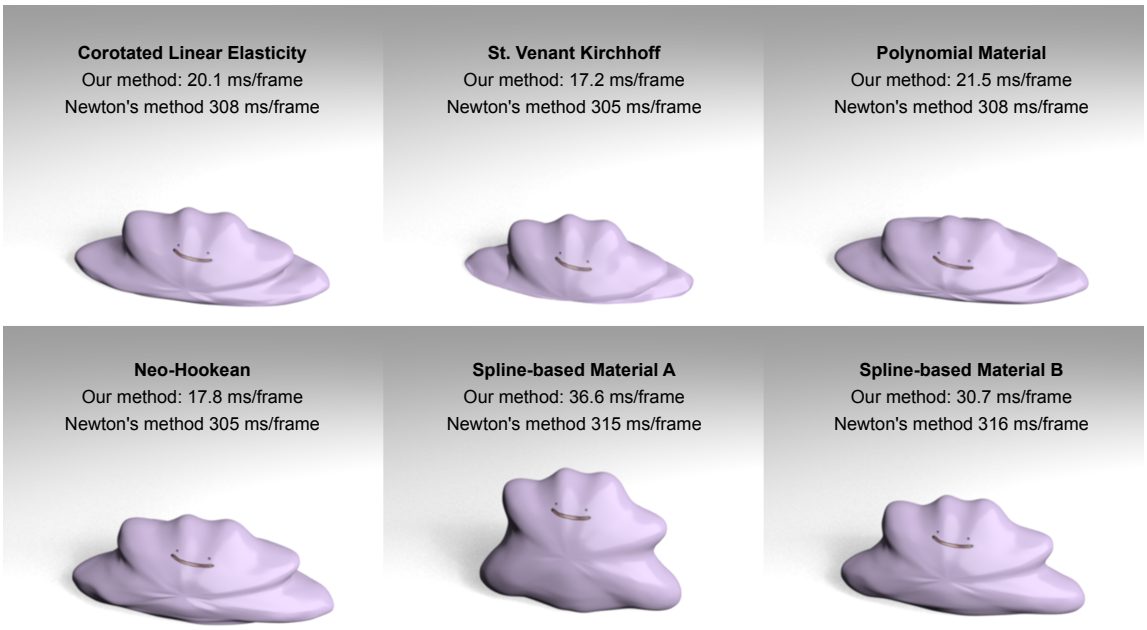


Figure 7.1: Our method enables fast simulation of many different types of hyperelastic materials. Compared to the commonly-applied Newton's method, our method is about 10 times faster, while achieving even higher accuracy and being simpler to implement. The Polynomial and Spline-based materials are models recently introduced by Xu et al. [2015]. Spline-based material A is a modified Neo-Hookean material with stronger resistance to compression; spline-based material B is a modified Neo-Hookean material with stronger resistance to tension.

by Xu et al. [2015], see Figure 7.1.

We demonstrate that our extensions to more general materials and the L-BFGS solver upgrade do not compromise simulation robustness. In Figure 7.2, we show an elastic hippo which recovers from an extreme (randomized) deformation with many inverted elements. Specifically, the hippo model uses L-BFGS with $m = 5$ and corotated linear elasticity with $\mu = 20$ and $\lambda = 100$.

In order to compare the convergence of different methods, we use the relative error,

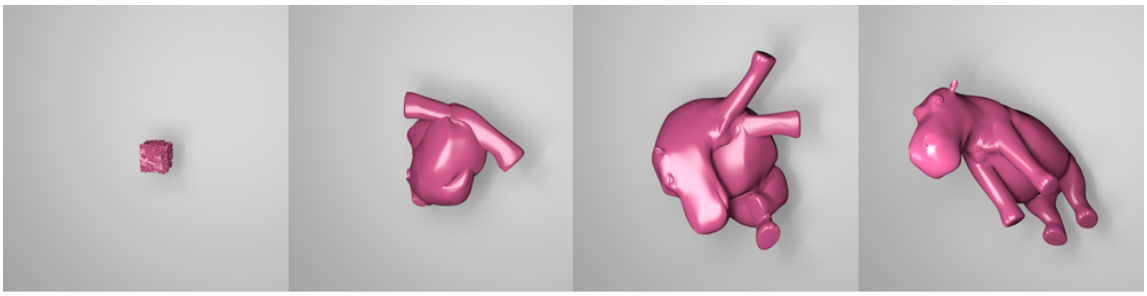


Figure 7.2: Our method is robust despite extreme initial conditions: a randomly initialized hippo returns back to its rest pose. This example does not contain any explicit inversion handling constraints, only the standard penalization of inverted elements due to corotated elasticity.

defined as:

$$\frac{g(\mathbf{x}^{(k)}) - g(\mathbf{x}^*)}{g(\mathbf{x}^{(1)}) - g(\mathbf{x}^*)}, \tag{7.15}$$

where $\mathbf{x}^{(1)}$ is the initial guess (we use $\mathbf{x}^{(1)} := \mathbf{y}$ for all methods), $\mathbf{x}^{(k)}$ is the k-th iterate, and $\mathbf{x}^*$ is the exact solution computed using Newton's method (iterated until convergence).

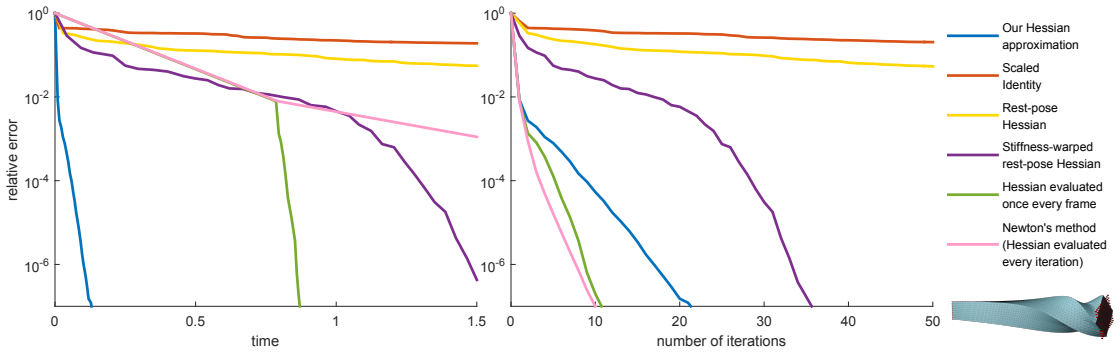


Figure 7.3: Convergence comparison of L-BFGS methods (all using $w = 5$) initialized with different Hessian approximations, along with Newton's method (baseline). The model is "Twisting bar" with Neo-Hookean elasticity.

We first analyze the different choices of the initial estimates of the Hessian. Our method can be interpreted as providing a particularly good initial estimate of the Hessian for L-BFGS. Therefore, it is important to compare to other possible Hessian initializations. In

a general setting, Nocedal and Wright [2006] recommend to bootstrap L-BFGS using a scaled identity matrix:

$$\mathbf{A}^{(0)} := \frac{\mathrm{tr}((\mathbf{s}^{(k-1)})^{\mathsf{T}}\mathbf{y}^{(k-1)})}{\mathrm{tr}((\mathbf{y}^{(k-1)})^{\mathsf{T}}\mathbf{y}^{(k-1)})}\mathbf{I}. \tag{7.16}$$

We experimented with this approach, but we found that our choice $\mathbf{A}^{(0)} := \mathbf{M}/h^2 + \mathbf{L}$ leads to much faster convergence, trumping the computational overhead associated with solving the pre-factorized system $\mathbf{A}^{(0)}\mathbf{r} = \mathbf{q}$ (see Figure 7.3, blue graph).

Another possibility would be to set $\mathbf{A}^{(0)}$ equal to the rest-pose Hessian, (formally, $\mathbf{A}^{(0)} := \mathbf{M}/h^2 + \nabla^2 E(\mathbf{X})$, where $\mathbf{X}$ is the rest pose configuration). It is also a constant matrix which can be pre-factorized. As shown in Figure 7.3 (yellow graph), this is a slightly better approximation than scaled identity, but still not very effective. This is because the actual Hessian depends on world-space rotations of the model, deviating significantly from the rest-pose Hessian.

$\mathbf{A}^{(0)}$	Rest-pose Hessian		Our Hessian approximation	
configuration	1	2	1	2
condition number of ${\mathbf{A}^{(0)}}^{-1}\nabla^2 g(\mathbf{x})$	2.45	45.5	9.94	9.94

Table 7.1: Condition number of ${\mathbf{A}^{(0)}}^{-1}\nabla^2 g(\mathbf{x})$ in Configurations 1 and 2 as in Figure 7.4, where $\nabla^2 g(\mathbf{x})$ is the exact Hessian matrix evaluated at the beginning of the frame and $\mathbf{A}^{(0)}$ is an approximate Hessian, computed 1) at the rest pose and 2) using our method.

Figure 7.4 shows an example illustrating the drawbacks of the rest-pose Hessian. Configuration 1 shows an elastic cube released from a slightly stretched state. In this configuration, setting $\mathbf{A}_0$ to the rest-pose Hessian results in faster error reduction than our method (red graph), because the initial configuration is close to the rest pose and therefore the exact Hessian is close to the rest-pose Hessian. Unfortunately, when we rotate the initial configuration by 90 degrees (Configuration 2), the rest-pose Hessian becomes ineffective, as it is far away from the exact Hessian (yellow graph). Our Hessian approximation is invariant to rigid body transformations and therefore leads to the same error reduction in both Configurations 1 and 2 (blue graph). To further analyze this effect,

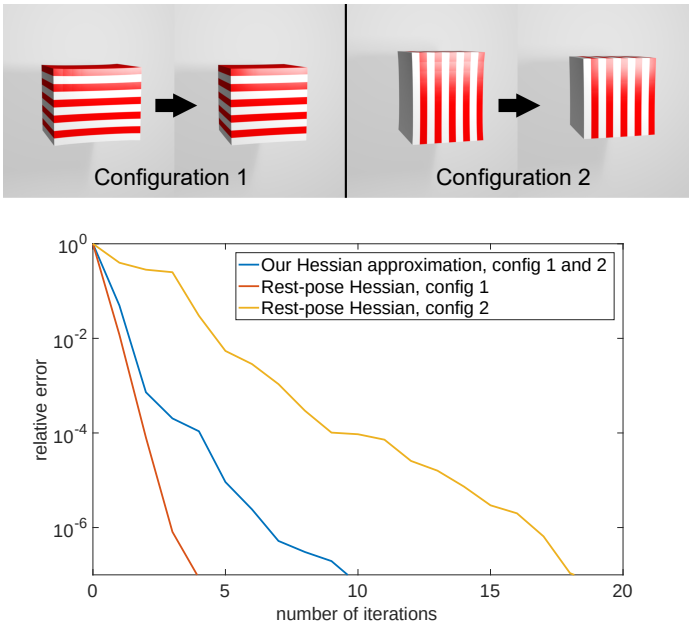


Figure 7.4: Convergence comparison of L-BFGS methods with different configurations. Configuration 1 is the simulation of an elastic cube with Neo-Hookean elasticy, released from a horizontally stretched pose, close to the rest pose. Configuration 2 is the same configuration rotated by 90 degrees. Using the rest-pose Hessian as initial Hessian approximation does not work well in Configuration 2. Our method is rotation invariant and therefore performs equally well in both configurations.

we also computed the condition number of $\mathbf{A}^{(0)^{-1}}\nabla^2 g(\mathbf{x})$ where $\mathbf{A}^{(0)}$ is an approximate Hessian. The conditions numbers reported in Table 7.3 confirm this observation.

Another interpretation of our Hessian approximation can be derived from the energy density function of the Neo-Hookean material [Sifakis and Barbic, 2012]:

$$\Psi(\mathbf{F}) = \frac{\mu}{2}(\text{tr}(\mathbf{F}^\mathsf{T}\mathbf{F}) - 3) - \mu \log(\det(\mathbf{F})) + \frac{\lambda}{2}\log^2(\det(\mathbf{F})) \tag{7.17}$$

where $\mathbf{F}$ is the deformation gradient and μ and λ are Lamé coefficients. In this case, our Hessian approximation corresponds to the first term, i.e., $\text{tr}(\mathbf{F}^\mathsf{T}\mathbf{F})$, which is indeed rotation invariant. The rest-pose Hessian is not rotation invariant and thus produces worse approximation of the exact Hessian as shown in Figure 7.4.

This issue of the rest-pose Hessian was observed [Müller et al., 2002], who proposed per-vertex *stiffness warping* as a possible remedy. Per-vertex stiffness warping still allows us to leverage pre-factorization of the rest-pose Hessian and results in better convergence than pure rest-pose Hessian, see Figure 7.3 (purple graph). However, per-vertex stiffness warping may introduce ghosts forces, because stiffness warping uses different rotation matrices for each vertex, which means that internal forces in one element no longer have to sum to zero. The ghost forces disappear in a fully converged solution, however, this would require a prohibitively high number of iterations.

Yet another possibility is to completely re-evaluate the Hessian at the beginning of each frame. This requires re-factorization, however, the remaining iterations can reuse the factorization, relying only on L-BFGS updates. When measuring convergence with respect to the number of iterations, this approach is very effective, as shown in Figure 7.3 (right, green graph). However, the cost of the initial Hessian factorization is significant, as can be seen in Figure 7.3 (left, green graph). Our method uses the same Hessian factorization for all frames, avoiding the per-frame factorization costs, while featuring excellent convergence properties, see Figure 7.3 (blue graph).

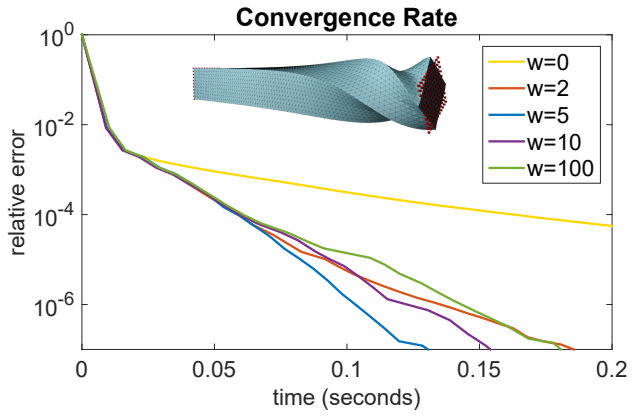


Figure 7.5: Comparison of L-BFGS convergence rate with different history window sizes (w).

In order to find the best history window size (w), we experimented with different values of w, see Figure 7.5. Too large m takes into account too distant iterates which can

lead to a worse approximation of the Hessian. In Figure 7.5, we see the optimal value is $w = 5$, which is also our recommended default setting. However, it is comforting that the algorithm is not particularly sensitive to the setting of w – even large values such as $w = 100$ produce only slightly worse convergence. We also observed that in scenarios with frequent collisions, the history becomes less useful. In these cases, reducing the window size according to the number of newly instantiated collision constraints may be beneficial.

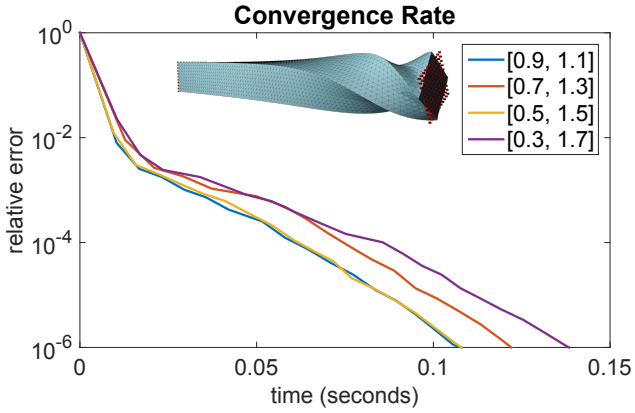


Figure 7.6: The convergence rate for different stiffness parameters chosen from different $[x_{\text{start}}, x_{\text{end}}]$ intervals.

As discussed in the previous section, we use Eq. 7.14 to define our stiffness parameter k as the slope of the best linear approximation of Eq. 7.13 at $\in [x_{\text{start}}, x_{\text{end}}]$. What is the best $[x_{\text{start}}, x_{\text{end}}]$ interval to use? In the limit, with $[x_{\text{start}}, x_{\text{end}}] \to [1, 1]$, our k would converge to the second derivative. However, a finite interval $[x_{\text{start}}, x_{\text{end}}]$ guarantees that our k is meaningful even for materials such as the polynomial material $a(x) = \mu(x-1)^4, b(x) = 0, c(x) = 0$; in this case, we obtain a k which depends linearly on μ. We argue the convergence of our algorithm is not very sensitive to a particular choice of the $[x_{\text{start}}, x_{\text{end}}]$ interval. In Figure 7.6, we show convergence graphs of a twisting bar with Neo-Hookean material using different intervals to compute the stiffness parameter k. Although Neo-Hookean material is highly non-linear, the convergence rates for different interval choices are quite similar. Therefore, we decided not to investigate more sophisticated strategies and we set $x_{\text{start}} = 0.5, x_{\text{end}} = 1.5$ in all of our simulations.

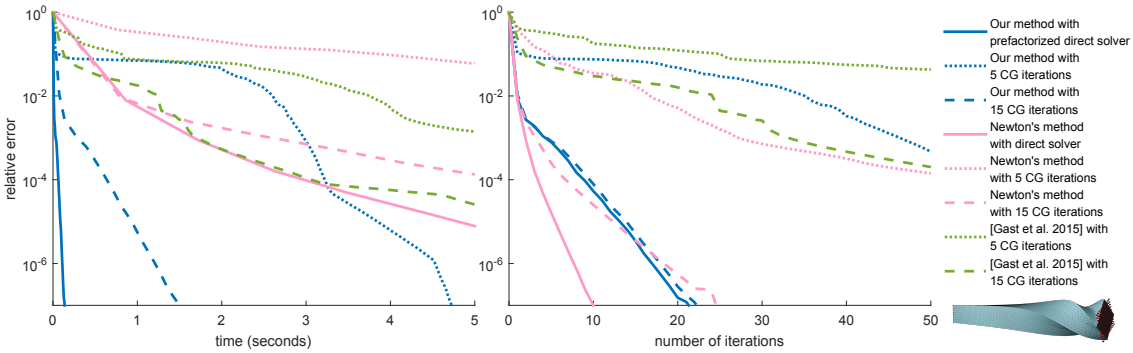


Figure 7.7: Convergence comparison of various methods using sparse direct solvers and conjugate gradients. The model is "Twisting bar" with Neo-Hookean elasticity.

Sparse iterative solvers do not require expensive factorizations and are therefore attractive in interactive applications. A particularly popular iterative solver is the Conjugate Gradient method (CG) [Hestenes and Stiefel, 1952]. An additional advantage is that CG can be implemented in a matrix-free fashion, i.e., without explicitly forming the sparse system matrix. Gast et al. [2015] further accelerate the CG solver used in Newton's method by proposing a CG-friendly definiteness fix. Specifically, the CG iterations are terminated whenever the maximum number of iterations is reached or indefiniteness of the Hessian matrix is detected.

While iterative methods can be the only possible choice in high-resolution simulations (e.g., in scientific computing), in real-time simulation scales, sparse direct solvers with pre-computed factorization are hard to beat, as we show in Figure 7.7. Specifically, we test Newton's method with linear systems solved using CG with 5 and 15 iterations, using Jacobi preconditioner. Even with 15 CG iterations, the accuracy is still not the same as with the direct solver while the computational cost becomes high. If we use only five CG iterations the linear system solving time improves, but the convergence rate suffers because the descent directions are not sufficiently effective. The method of Gast et al. [2015] initially outperforms Newton with CG, however, the convergence slows down in subsequent iterations. We also tried to apply CG to our method, in lieu of the direct solver. With 15

CG iterations the convergence is competitive, however, the CG solver is slower.

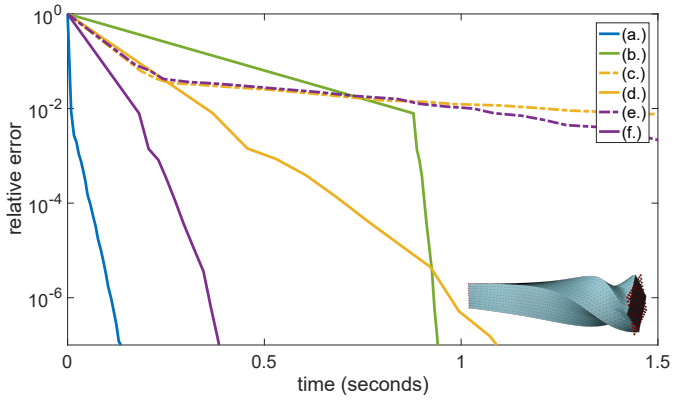


Figure 7.8: Comparison to preconditioned conjugate gradients (PCG) with different preconditioners, tested on the "Twisting bar" example with Neo-Hookean elasticity. Experiment (a.) is L-BFGS using our Hessian approximation as the initial guess. The rest of the experiments (b. - f.) are L-BFGS using the Hessian matrix evaluated at the beginning of each frame solved by the following options: (b.) direct solver; (c.) 5 PCG iterations with incomplete Cholesky (ichol) of the rest-pose Hessian; (d.) 15 PCG iterations with ichol of the rest-pose Hessian; (e.) 5 PCG iterations with ichol of our Hessian approximation; (f.) 15 PCG iterations with ichol of our Hessian approximation;

A carefully chosen preconditioner usually helps the convergence of a CG solver. Figure 7.8 shows an example of the effect of different preconditioners. The green graph is L-BFGS initialized with the Hessian matrix evaluated at the beginning of every frame, solved by a direct solver. This Hessian approximation provides a very nice initial guess for L-BFGS, but its evaluation and factorization is too expensive to execute once per frame. The two yellow graphs use the same Hessian approximation as the green graph, but are solved using preconditioned conjugate gradients (PCG) with 5 and 15 iterations, using incomplete Cholesky factorization of the rest-pose Hessian as a preconditioner. Compared with the green graph, we can see that the PCG solver is more efficient especially at the first several iterations, since it does not require the expensive factorization of the Hessian matrix. However, the convergence starts to slow down at later iterates. The two purple graphs

are using the exact same configuration as the yellow graphs, but use incomplete Cholesky factorization of our Hessian approximation, $\mathbf{M}/h^2 + \mathbf{L}$, as a preconditioner. We can see that our Hessian approximation is a better preconditioner than the rest-pose Hessian.

Chapter 8

Implementation

In this chapter, we discuss the implementation details, including constructing the system matrix, setting attachment constraints, simulating anisotropic materials and tearing effects, handling collision and damping etc.. A C++ implementation can be found on https://github.com/ltt1598/Quasi-Newton-Methods-for-Real-time-Simulation-of-Hyperelastic-Materials.

8.1 Construction of the Laplacian Matrix

The core of implementing our methods is to evaluate the system matrix $\mathbf{M}/h^2 + \mathbf{L}$, where $\mathbf{M}$ is the mass matrix, h is the time step and $\mathbf{L}$ is the Laplacian matrix of the system. This matrix is used as the system matrix for the global step for mass-spring systems and Projective Dynamics, and treated as a Hessian approximation in the quasi-Newton methods for simulating general hyperelastic materials. In our implementation, we use a diagonally lumped matrix to represent the nodal mass of the system. As long as the Laplacian matrix $\mathbf{L}$ is evaluated, we can then compute the system matrix $\mathbf{M}/h^2 + \mathbf{L}$.

For a **mass spring system**, the Laplacian matrix $\mathbf{L}$ is defined as:

$$\mathbf{L} = \sum_{j=1}^{m} k_j \mathbf{G}_j \mathbf{G}_j^{\mathsf{T}} \tag{8.1}$$

where $\mathbf{G}_j \in \mathbb{R}^{n\times 1}$ is the incidence vector of the j-th spring, i.e. $\mathbf{G}_j(j1) = 1$, $\mathbf{G}_j(j2) = -1$ and zero otherwise, assuming $j1$ and $j2$ are the indices for the endpoints of the j-th spring. Therefore the contribution of the Laplacian matrix $\mathbf{L}$ from the j-th spring can be explicitly computed as:

$$\mathbf{L}_j = k_j \begin{array}{c} \begin{matrix} & j1 & & j2 & \end{matrix} \\ \begin{bmatrix} \dots & \dots & \dots & \dots & \dots \\ \dots & 1 & \dots & -1 & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & -1 & \dots & 1 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \end{array} \begin{array}{l} \\ j1 \\ \\ j2 \\ \end{array} \tag{8.2}$$

where only four elements corresponding to the indices of the endpoints of the j-th spring is non-zero.

Eq. 8.2 is exactly a weighted graph Laplacian matrix where the weight is defined as the stiffness of the spring. We simply sum up the contribution from each spring to construct the Laplacian matrix: $\mathbf{L} = \sum_{j=1}^{m} \mathbf{L}_j$.

The laplacian matrix of a three dimensional **finite element model** is defined almost the same $\mathbf{L} = \sum_{j=1}^{m} k_j \mathbf{G}_j \mathbf{G}_j^{\mathsf{T}}$, except for the definition of $\mathbf{G}_j$ matrix is overwritten as $\mathbf{G}_j = \left(\mathbf{D}_{mj}^{-\mathsf{T}} \mathbf{A}_j\right)^{\mathsf{T}} \in \mathbb{R}^{n\times 3}$. Here $\mathbf{D}_{mj}^{-\mathsf{T}} = [\mathbf{X}_{j1} - \mathbf{X}_{j4}, \mathbf{X}_{j2} - \mathbf{X}_{j4}, \mathbf{X}_{j3} - \mathbf{X}_{j4}]^{-\mathsf{T}}$ is the inverse transpose of the rest-pose edge matrix of the j-th element, and $\mathbf{A}_j$ is defined in Eq. 6.3. If we denote matrix $\mathbf{D}_{mj}^{-1}\mathbf{D}_{mj}^{-\mathsf{T}}$ as a matrix $\mathbf{D}$:

$$\mathbf{D}_{mj}^{-1}\mathbf{D}_{mj}^{-\mathsf{T}} = \mathbf{D} = \begin{bmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \\ d_{31} & d_{32} & d_{33} \end{bmatrix} \tag{8.3}$$

we can compute the contribution from the j-th element as:

$$\mathbf{L}_j = k_j V_j \begin{array}{c} \begin{array}{ccccccccc} & j1 & & j2 & & j3 & & j4 & \end{array} \\ \begin{bmatrix} \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & d_{11} & \dots & d_{12} & \dots & d_{13} & \dots & -\sum_{l=1}^{3} d_{1l} & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & d_{21} & \dots & d_{22} & \dots & d_{23} & \dots & -\sum_{l=1}^{3} d_{2l} & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & d_{31} & \dots & d_{32} & \dots & d_{33} & \dots & -\sum_{l=1}^{3} d_{3l} & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & -\sum_{k=1}^{3} d_{k1} & \dots & -\sum_{k=1}^{3} d_{k2} & \dots & -\sum_{k=1}^{3} d_{k3} & \dots & \sum_{k,l=1}^{3} d_{kl} & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{bmatrix} \end{array} \begin{array}{c} \\ j1 \\ \\ j2 \\ \\ j3 \\ \\ j4 \\ \end{array} \tag{8.4}$$

where k_j is the stiffness parameter and V_j is the rest-pose volume of the j-th element. For nonlinear materials, the stiffness parameter k is evaluated according to Eq. 7.14 using linear regression. For example, if we a pick Neo-Hookean model with $x_{start} = 80\%$ and $x_{end} = 120\%$, we can compute the coresponding $k = 2.0260\mu + 1.0480\lambda$ where μ and λ are second and first Lamé coefficients. Other than these 16 elements, every number in this matrix $\mathbf{L}_j$ is zero. The overall Laplacian matrix of a finite element model is again: $\mathbf{L} = \sum_{j=1}^{m} \mathbf{L}_j$.

8.2 Attachment Constraints

Attachment constraints are used to pin a vertex at a certain point, or to move a vertex towards a wanted position. For example, the red dots in Figure 5.2 are the target positions for the attachment constraints. In our implementation, attachment constraints are treated as soft constraints which can be seen as zero-length springs with the following energy representation:

$$E_{attachment}(\mathbf{x}) = \frac{k_{attachment}}{2} ||\mathbf{x} - \mathbf{x}_a||^2 \tag{8.5}$$

where $k_{attachment}$ is the stiffness of the attachment spring, $\mathbf{x}$ is the position of the attached vertex and $\mathbf{x}_a$ is the target position. Since $E_{attachment}$ is a special energy with a constant

Hessian matrix, we can simply encode the Hessian of an attachment spring into our Hessian approximation:

$$\mathbf{L}_{attachment,i} = k_{attachment,i} \begin{matrix} \\ \\ i \end{matrix} \begin{bmatrix} \dots & \dots & \dots \\ \dots & 1 & \dots \\ \dots & \dots & \dots \end{bmatrix} \begin{matrix} \\ \\ i \end{matrix} \tag{8.6}$$

where $k_{attachment,i}$ is the stiffness of an attachment spring attaching $\mathbf{x}_i$ to a certain position. $\mathbf{L}_{attachment,i}$ is the contribution to the entire Laplacian matrix $\mathbf{L}$ in our Hessian approximation $\mathbf{M}/h^2 + \mathbf{L}$, like we did for mass-spring systems and FEM systems.

8.3 Anisotropic Materials

Our numerical methods, including the L-BFGS acceleration, can be directly applied also to anisotropic material models. We verified this on an elastic cube model with corotated base material enhanced with additional anisotropic stiffness term $\frac{\kappa}{2}(\|\mathbf{F}\mathbf{d}\| - 1)^2$, where $\mathbf{F}$ is the deformation gradient and $\mathbf{d}$ is the (rest-pose) direction of anisotropy. This corresponds to the directional reinforcement of the material which is common, e.g., in biological soft tissues containing collagenous fibers. The Hessian matrix of an anisotropic material is different from its x-, y- and z-directions, therefore it is not recommended to use the same Hessian approximation for all three dimensions as we did in Chapter 7. However, since our Hessian approximation $\mathbf{M}/h^2 + \mathbf{L}$ is guaranteed to be positive definite, the direction $-[\mathbf{M}/h^2+\mathbf{L}]^{-1}\nabla g(\mathbf{x})$ is guaranteed to be a descent direction as well. One way to make use of our quasi-Newton acceleration is to ignore the Hessian contribution from the anisotropic components of a material completely in the Hessian approximation. We incorporate the information of an anisotropic material only in the objective energy and the gradient in Alg. 3. This will approach will slightly slow down the convergence of our method, depending on the stretch of the anisotropic components. The result of our method can be seen in Figure 8.1.

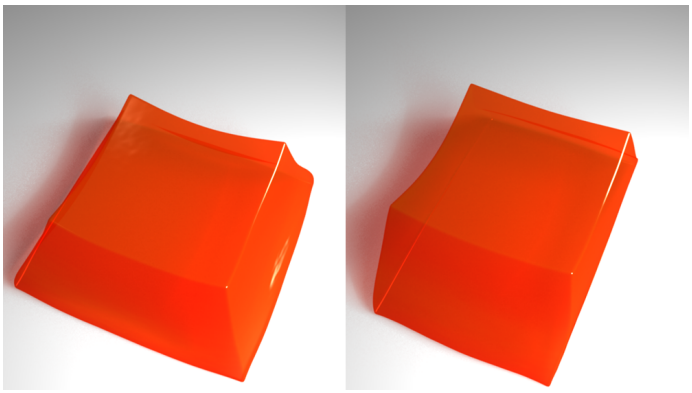


Figure 8.1: Dropping an elastic cube ($\mu = 10$, $\lambda = 100$) on the ground. Left: deformation using isotropic elasticity (linear corotated model). Right: the result after adding anisotropic stiffness ($\kappa = 50$).

8.4 Tearing and Cutting

Tearing and cutting effects, as shown in Figure 8.2, change the topology of a mesh, therefore change the corresponding Laplacian matrix as well. In our implementation, we simply remove an element from the gradient and objective evaluation once decide that element needs to be removed from the system during tearing. A straightforward idea to simulate this effect is to update and re-factorize our Hessian approximation $\mathbf{M}/h^2 + \mathbf{L}$ after the system detects the topological change. Note that only the Hessian components of those elements along or on the cutting edges need to be updated. Therefore the computational cost is not as expensive as re-computing the Hessian matrix. Another approach is to ignore the topological change from the Hessian approximation after the tearing or cutting. The rationale behind this approach is the same with how we treat anisotropic materials in Section 8.3 – as long as we can always get a descent direction, all we need to pay is a slightly higher number of iterations for using a less accurate Hessian approximation. To be more specific, we remove the contribution from a "torn" element when computing the gradient and objective in line 4 and line 10 of Alg. 3, while keeping the hessian approximation $\mathbf{M}/h^2 + \mathbf{L}$ in line 5 the same. We use this approach to simulate the result in Figure 8.2

in real time. We can also combine these two ideas together: observing that the tearing and cutting effects are permanent once made, we can update the Hessian approximation on a background thread, and update the topological change information to the foreground simulation thread only when the background thread finishes the factorization of the updated matrix.

Figure 8.2: Simulating a flag being torn apart by the wind.

8.5 Collisions

A classical approach to enforcing non-penetration constraints between deformable solids is to 1) detect collisions and 2) resolve them using temporarily instantiated repulsion springs, which bring the volume of undesired overlaps to zero [Harmon et al., 2011, McAdams et al., 2011]. However, in Projective Dynamics the primary emphasis is on computational efficiency and therefore only simplified collision resolution strategies have been proposed by Bouaziz et al. [2014]. Specifically, Projective Dynamics offers two possible strategies. The first strategy is a two-phase method, where collisions are resolved in a separate post-processing step using projections, similar to Position Based Dynamics. The drawback of this approach is the fact that collision projections are oblivious to elasticity and inertia of the simulated objects. The second approach used in Projective Dynamics is more physically

realistic, but introduces additional computational overhead. Specifically, temporarily-instantiated repulsion springs are added to the total energy. This leads to physically realistic results, but the drawback is that the matrix $\mathbf{M}/h^2 + \mathbf{L}$ needs to be re-factorized whenever the set of repulsion springs is updated – typically, at the beginning of each frame.

The quasi-Newton interpretation invites a new approach to collision response which is physically realistic, but avoids expensive re-factorizations. Specifically, for each inter-penetration found by collision detection, we introduce an energy term:

$$E_{\text{collision}}(\mathbf{x}) = \frac{k_{\text{collision}}}{2}((\mathbf{x} - \mathbf{x}_s)^{\mathsf{T}}\mathbf{n})^2 \tag{8.7}$$

where $k_{\text{collision}}$ is the collision stiffness, $\mathbf{x}$ is a vertex being detected as a colliding vertex, $\mathbf{x}_s$ is its projection on the surface and $\mathbf{n}$ is the surface normal. This constraint pushes the collided vertex to the tangent plane. It is important to add this term to our total energy $E(\mathbf{x})$ only if the vertex is in collision or contact. Whenever the relative velocity between the vertex and the collider indicates separation, the $E_{\text{collision}}(\mathbf{x})$ term is discarded (otherwise it would correspond to unrealistic "glue" forces). This is done once at the beginning of each iteration (just before line 3 in Alg. 3). The rest of our algorithm (lines 6-10 of Alg. 3) is unaffected by these updates, i.e., the unilateral nature of the collision constraints is handled correctly without any further processing.

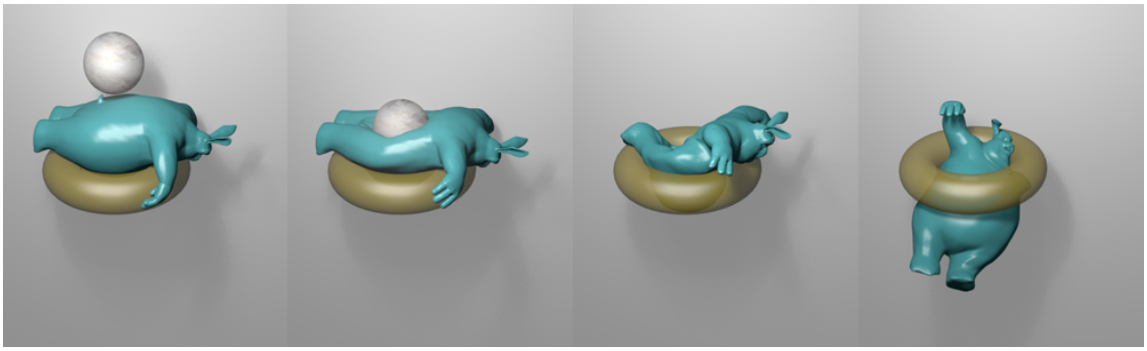


Figure 8.3: Our method is capable of simulating complex collision scenarios, such as squeezing the Big Bunny through a torus. The Big Bunny uses corotated elasticity with $\mu = 5$ and $\lambda = 200$.

The key idea of our approach is to leverage the quasi-Newton approximation for collision processing. In particular, we account for the $E_{\text{collision}}(\mathbf{x})$ terms when evaluating the energy and its gradients, but we ignore their contributions to the $\mathbf{M}/h^2 + \mathbf{L}$ matrix. This means that we form a somewhat more aggressive approximation of the Hessian, with the benefit that the system matrix will never need to be re-factorized. The line search process (lines 6-10 in Alg. 3) guarantees that energy will decrease in spite of this more aggressive approximation. The only trade-off we observed in our experiments is that the number of line search iterations may increase, which is a small cost to pay for avoiding re-factorizations. We observed that even in challenging collision scenarios, such as when squeezing a Big Bunny through a torus, the approach behaves robustly and successfully resolves all collisions, see Figure 8.3.

8.6 Damping

A simple method to introduce viscous damping into our formulation is as follows. Recall that the term $\mathbf{y}$ from Eq. 3.30 is simply the result of inertia when all internal forces are ignored, i.e., $\mathbf{y} = \mathbf{x}_n + h\mathbf{v}_n + h^2\mathbf{M}^{-1}\mathbf{f}_{ext}$. Damping can be achieved simply by setting $\mathbf{y}$ to $\mathbf{x}_n + h\tilde{\mathbf{v}}_n + h^2\mathbf{M}^{-1}\mathbf{f}_{ext}$, where we replaced $\mathbf{v}_n$ with a damped velocity $\tilde{\mathbf{v}}_n$. We use only a very simple damping model—ether drag [Su et al., 2013], which sets $\tilde{\mathbf{v}}_n := \alpha\mathbf{v}_n$, where $\alpha \in [0, 1]$ is a parameter, typically very close to 1. However, any damping model can be used with our method, such as the rigid-body modes preserving drag [Müller et al., 2007] or truly material-only stiffness-proportional damping [Nealen et al., 2006].

Chapter 9

Future Work

The key component underneath our methods from Chapter 5 to Chapter 7 is the constant system matrix $\mathbf{M}/h^2 + \mathbf{L}$. We evaluate and factorize this matrix once at the beginning of the simulation, so that during the simulation when a linear system needs to be solved, we only need to perform a forward and a backward substitution. This matrix dependents on the nodal mass of a mesh, the time step size of the simulation (we usually use 33 ms for realtime applications), the stiffness of each element and the topology of the system. We assume all those quantities are unlikely to change during a simulation. This assumption forms the foundation of all the accelerations we made to simulate deformable materials. But it might be violated in some challenging cases where the speedup from our methods reduces. Our system matrix can not encode the unpredictable information from time-varying events such as collisions, tearing and cutting. Although several approaches are mentioned in Section 8.4 and Section 8.5, we are still looking for better numerical solutions.

Collision by itself is already a challenging problem. The classical model of repulsion springs [McAdams et al., 2011] which we adopted in our implementation is analogous to an active set method that adds / removes constraints during the iterations of the algorithm. It is possible that this approach will end up cycling, however, we have never observed this in practice. One possible workaround is to limit the number of iterations, possibly leaving some collision constraints unresolved. In collision-dominant simulations, more advanced

algorithms may be necessary. Another limitation is that in our current implementation, we treat collisions as soft constraints with relatively stronger stiffness compared to the elastic models. One possible way to resolve hard collision constraints is to use Lagrangian multipliers, by solving the KKT system using its Schur complement [Ichim et al., 2016]. However, in cases with many collision constraints, the Schur complement becomes impractically large. Another possible approach to treating hard collision constraints is the Augmented Lagrangian method [Deng et al., 2013]. Fast and robust collision resolution in challenging scenarios is a problem which deserves significant attention in future work.

Another direction is to explore the possibility of parallelizing our methods. The local step in Projective Dynamics or the gradient evaluation step in our quasi-Newton method is trivially parallelizable. However we can parallelize the global step or the descent direction evaluation step depends on the linear solver we choose. We currently rely on a constant Cholesky factorization of our system matrix so that only forward and backward substitutions are needed during the runtime. The substitutions are fast in a sequential implementation on the CPU for moderate sized problems, but might not be the best solution for solving larger systems in parallel because of two reasons. First, the forward and backward substitutions are by nature sequential: each unknown can only be solved whenever all the unknowns in the sub-triangle are solved. Second, the factorization of a sparse matrix, even with a reasonable reordering, typically contains one or two magnitudes more non-zero elements compared to the original matrix, making it hard to scale to gigantic systems. Recently, Wang et al. [2015, 2016] use a Jacobi method on the GPU to solve Projective Dynamics problems and further accelerate it using a Chebyshev semi-iterative method; Fratarcangeli et al. [2016] accelerate the linear system solve in Projective Dynamics using a novel graph-coloring approach to achieve fast Gauss-Seidel solves on the GPU. Both of them show that iterative solvers could be the parallelization-friendly alternatives. The dotted blue graph in Figure 7.7 shows that even in a CPU implementation, a conjugate gradient (CG) method with only 15 iterations is not too worse compared to the solution obtained by our pre-factorized direct solver. We believe that CG can be a competitive candidate for

implementing our methods in parallel.

Integrating stiff or even rigid material is another challenging problem. In the real world, deformable bodies usually do not exist on their own, but more commonly coupled with rigid components. For example, in an anatomical simulation, deformable flesh is attached to rigid bones, and simulating only deformable part would not be interesting enough. In theory, one can modify our deformable body simulation framework to simulate stiff materials or even rigid bodies by increasing the stiffness of the elements close to infinite. However, it is numerically hard to simulate such kind materials efficiently. As shown in Figure 5.3, the stiffness of the material affects the convergence of our method directly: the stiffer material we simulate, the slower our solver will converge. Tournier et al. [2015] show that an extremely stiff systems can not be simulated well even using Newton's method. In order to solve this problem, they propose a compliant constraint based framework, supporting both soft elastic materials and hard constraints at the same time. However, the performance of their work is still far away from the realtime requirement. We would like to explore fast numerics for simulating stiff materials in the future.

Damping is a common physics phenomenon, simple post-processing damping models are fast in the cost of plausibility. Ether drag [Su et al., 2013] reduces the linear and angular momentum to a system. PBD damping [Müller et al., 2007] preserves the momentum quantities, but in a non-physical way. Recently, Li et al. [2018] studies the matrix in Projective Dynamics and proposes a Laplacian damping method, combining the efficiency of our methods with an approximate model of Rayleigh damping. For completeness of our method, we want to study a better damping method in the future.

Our current methods are designed for the backward Euler integration. But we do not intend to limit our method with the backward Euler scheme. Applying our Hessian approximation to Eq. 3.31 should allow us to accelerate implicit midpoint integration as well. However, the implicit midpoint integration is not guaranteed to be stable under large timesteps. Dinev et al. [2018] propose a method to stabilize implicit midpoint using a Projective Dynamics solver. We believe that in order to achieve realistic simulations of

deformable objects, a fast, stable yet not artificially damped time integration method is worth exploring as well.

Chapter 10

Conclusion

This thesis work summarizes the major projects I did during my Ph.D. study, from fast simulation of mass spring systems [Liu et al., 2013], to Projective Dynamics [Bouaziz et al., 2014], to a more general quasi-Newton simulation of a variety of different materials [Liu et al., 2017]. The goal is to push the simulation of deformable body towards real-time.

Fast simulation of mass-spring systems [Liu et al., 2013] proposes a numerical method for implicit Euler time stepping of mass-spring system dynamics. The technique is based on block coordinate descent, which gives it different properties than the traditional Newton's method. The method can approximate the solution in a limited amount of computational time, making it particularly attractive for real-time applications—we demonstrate real-time cloth with quality similar to the exact solution. The proposed algorithm can also be useful for quick simulation preview and for bootstrapping Newton's method.

Projective Dynamics [Bouaziz et al., 2014] generalizes the idea of [Liu et al., 2013], supporting more general spatial discretizations of deformable objects including finite element discretizations.It introduces an implicit constraint-based solver for real-time simulation. The approach is based on an abstract, constraint-based description of the physical system making the method general in its use to simulate a large variety of different geometries and materials. Projective Dynamics derives a broad set of constraints directly from continuous energies using proper discretization that make the solver robust to non-uniform meshing

with different resolutions.

Both [Liu et al., 2013] and [Bouaziz et al., 2014] apply a local/global solver to solve their numerical problems. The solver only requires the definition of a projection operator for a given constraints (local solve), making it very easy to implement. Furthermore, the global solve only requires solving a linear system, where the system matrix $\mathbf{M}/h^2 + \mathbf{L}$ is constant if the number of constraints is kept fixed, leading to efficient computation. Due to the independence of the local solves, the approach is also very well suited for parallelism, further boosting performance.

The quasi-Newton methods for real-time simulation of hyperelastic materials [Liu et al., 2017] further broadens the supported materials from Projective Dynamics [Bouaziz et al., 2014] and improves the convergence of the numerical solver. The key to this approach is the insight that Projective Dynamics can be re-formulated as a quasi-Newton method. Aided with line search, this work obtains a robust simulator supporting many practical material models. The quasi-Newton formulation also allows us to further accelerate convergence by combining itself with L-BFGS. Even though L-BFGS is sensitive to initial Hessian approximation, the prior knowledge we gained from [Liu et al., 2013] and [Bouaziz et al., 2014] suggests a particularly effective Hessian initialization $\mathbf{M}/h^2 + \mathbf{L}$ which yields fast convergence. Most of the experiments use ten iterations of quasi-Newton iterations which is typically more accurate than one iteration of Newton's method, while being about ten times faster and easier to implement.

I wish that these methods will encourage further investigation of time integration techniques and the underlying nonlinear numerical problems. Traditionally, real-time physics is considered to be approximate but fast, while off-line physics is accurate but slow. By keeping improving the performance and accuracy of the solver, I hope our work will eventually blur the boundary between real-time and off-line physically-based simulations of deformable objects.

Bibliography

Steven S. An, Theodore Kim, and Doug L. James. Optimizing cubature for efficient integration of subspace deformations. *ACM Trans. Graph.*, 27:165:1–165:10, 2008. 7

David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proc. of ACM SIGGRAPH*, pages 43–54, 1998. 6, 36, 41

Jernej Barbič and Doug L James. Real-time subspace integration for st. venant-kirchhoff deformable models. In *ACM Trans. Graph.*, volume 24, pages 982–990. ACM, 2005. 7

Adam W Bargteil and Elaine Cohen. Animation of deformable bodies with quadratic bézier finite elements. *ACM Trans. Graph.*, 33(3):27, 2014. 28

Klaus Jürgen Bathe and Arthur P Cimento. Some practical procedures for the solution of nonlinear finite element equations. *Computer Methods in Applied Mechanics and Engineering*, 22:59–85, 1980. 7

Jan Bender, Dan Koschier, Patrick Charrier, and Daniel Weber. Position-based simulation of continuous materials. *Computers & Graphics*, 44:1–10, 2014a. 8

Jan Bender, Matthias Müller, Miguel A Otaduy, Matthias Teschner, and Miles Macklin. A survey on position-based simulation methods in computer graphics. In *Comput. Graph. Forum*, volume 33, pages 228–251, 2014b. 8

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective

dynamics: fusing constraint projections for fast simulation. *ACM Transactions on Graphics (TOG)*, 33(4):154, 2014. 74, 81, 82

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004. 6, 25, 40

Isaac Chao, Ulrich Pinkall, Patrick Sanan, and Peter Schröder. A simple geometric model for elastic deformations. *ACM Trans. Graph.*, 29(4):38, 2010. 4, 46

Desai Chen, David Levin, Shinjiro Sueda, and Wojciech Matusik. Data-driven finite elements for geometry and material design. *ACM Trans. Graph.*, 34:74:1–74:10, 2015. 7

Kwang-Jin Choi and Hyeong-Seok Ko. Stable but responsive cloth. In *ACM SIGGRAPH 2005 Courses*, page 1. ACM, 2005. 4

Bailin Deng, Sofien Bouaziz, Mario Deuss, Juyong Zhang, Yuliy Schwartzburg, and Mark Pauly. Exploring local modifications for constrained meshes. In *Comput. Graph. Forum*, volume 32, pages 11–20, 2013. 78

Peter Deuflhard. *Newton methods for nonlinear problems: affine invariance and adaptive algorithms*. Springer Science & Business Media, 2011. 7

Dimitar Dinev, Tiantian Liu, and Ladislav Kavan. Stabilizing integrators for real-time physics. *ACM Transactions on Graphics (TOG)*, 37(1):9, 2018. 79

J Fish, M Pandheeradi, and V Belsky. An efficient multilevel solution scheme for large scale non-linear systems. *International Journal for Numerical Methods in Engineering*, 38:1597–1610, 1995. 7

Marco Fratarcangeli, Valentina Tibaldo, and Fabio Pellacini. Vivace: A practical gauss-seidel method for stable soft body dynamics. *ACM Transactions on Graphics (TOG)*, 35 (6):214, 2016. 78

Theodore F Gast, Craig Schroeder, Alexey Stomakhin, Chenfanfu Jiang, and Joseph M Teran. Optimization integrator for large time steps. *Visualization and Comp. Graph., IEEE Transactions on*, 21:1103–1115, 2015. 66

Joachim Georgii and Rüdiger Westermann. A multigrid framework for real-time simulation of deformable bodies. *Computers & Graphics*, 30(3):408–415, 2006. 6

Rony Goldenthal, David Harmon, Raanan Fattal, Michel Bercovier, and Eitan Grinspun. Efficient simulation of inextensible cloth. *ACM Trans. Graph.*, 26:49:1–49:7, 2007. 5, 32

David Harmon and Denis Zorin. Subspace integration with local deformations. *ACM Trans. Graph.*, 32:107:1–107:10, 2013. 7

David Harmon, Daniele Panozzo, Olga Sorkine, and Denis Zorin. Interference-aware geometric modeling. In *ACM Trans. Graph.*, volume 30, pages 137:1–137:10, 2011. 74

Florian Hecht, Yeon Jin Lee, Jonathan R. Shewchuk, and James F. O'Brien. Updated sparse cholesky factors for corotational elastodynamics. *ACM Trans. Graph.*, 31:123:1–123:13, 2012. 8

Magnus Rudolph Hestenes and Eduard Stiefel. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS, 1952. 66

Alexandru-Eugen Ichim, Ladislav Kavan, Merlin Nimier-David, and Mark Pauly. Building and animating user-specific volumetric face rigs. In *Proc. EG/ACM Symp. Computer Animation*, pages 107–117, 2016. 78

Geoffrey Irving, Joseph Teran, and Ronald Fedkiw. Invertible finite elements for robust simulation of large deformation. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 131–140. Eurographics Association, 2004. 5, 6, 31, 53

Liliya Kharevych, Weiwei Yang, Yiying Tong, Eva Kanso, Jerrold E Marsden, Peter Schröder, and Matthieu Desbrun. Geometric, variational integrators for computer animation. pages 43–51, 2006. 6, 21

Tae-Yong Kim, Nuttapong Chentanez, and Matthias Müller-Fischer. Long range attachments-a method to simulate inextensible clothing in computer games. In *Proc. EG/ACM Symp. Computer Animation*, pages 305–310, 2012. 8

Shahar Z. Kovalsky, Meirav Galun, and Yaron Lipman. Accelerated quadratic proxy for geometric optimization. *ACM Trans. Graph.*, 35:134:1–134:11, 2016. 8

Jing Li, Tiantian Liu, and Ladislav Kavan. Laplacian Damping for Projective Dynamics. In *Workshop on Virtual Reality Interaction and Physical Simulation*. The Eurographics Association, 2018. 79

Siwang Li, Jin Huang, Fernando de Goes, Xiaogang Jin, Hujun Bao, and Mathieu Desbrun. Space-time editing of elastic motion through material optimization and reduction. *ACM Trans. Graph.*, 33:108:1–108:10, 2014. 7

Tiantian Liu, Adam W Bargteil, James F O'Brien, and Ladislav Kavan. Fast simulation of mass-spring systems. *ACM Transactions on Graphics (TOG)*, 32(6):214, 2013. 81, 82

Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. Quasi-newton methods for real-time simulation of hyperelastic materials. *ACM Transactions on Graphics (TOG)*, 36(3):23, 2017. 81, 82

Miles Macklin and Matthias Müller. Position based fluids. *ACM Trans. Graph.*, 32: 104:1–104:12, 2013. 8

Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified particle physics for real-time applications. *ACM Trans. Graph.*, 33:153:1–153:12, 2014. 8

Miles Macklin, Matthias Müller, and Nuttapong Chentanez. Xpbd: Position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games*, pages 49–54. ACM, 2016. 5, 9, 31

Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. Example-based elastic materials. In *ACM Trans. Graph.*, volume 30, page 72. ACM, 2011. 6, 23, 36

Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph.*, 30:37:1–37:12, 2011. 6, 74, 77

Matthias Müller. Hierarchical Position Based Dynamics. In *Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS" (2008)*. The Eurographics Association, 2008. 6, 8

Matthias Müller and Nuttapong Chentanez. Solid simulation with oriented particles. In *ACM Trans. Graph.*, volume 30, pages 92:1–92:10, 2011. 8

Matthias Müller and Markus Gross. Interactive virtual materials. In *Proceedings of Graphics Interface*, pages 239–246, 2004. 8

Matthias Müller and Markus Gross. Interactive virtual materials. In *Proceedings of Graphics Interface 2004*, pages 239–246. Canadian Human-Computer Communications Society, 2004. 5

Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler. Stable real-time deformations. In *Proc. EG/ACM Symp. Computer Animation*, pages 49–54, 2002. 7, 64

Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. In *ACM Trans. Graph.*, volume 24, pages 471–478, 2005. 8

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007. 1, 5, 8, 31, 42, 76, 79

Matthias Müller, Nuttapong Chentanez, Tae-Yong Kim, and Miles Macklin. Strain based dynamics. In *Proc. EG/ACM Symp. Computer Animation*, volume 2, pages 149–157, 2014. 8

Matthias Müller, Nuttapong Chentanez, Tae-Yong Kim, and Miles Macklin. Air meshes for robust collision handling. *ACM Trans. Graph.*, 34:133:1–133:9, 2015. 8

Rahul Narain, Armin Samii, and James F O'Brien. Adaptive anisotropic remeshing for cloth simulation. *ACM Trans. Graph.*, 31:152:1–152:10, 2012. 5, 43

Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics. In *Computer graphics forum*, volume 25, pages 809–836. Wiley Online Library, 2006. 76

Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer Verlag, 2006. 2, 53, 56, 58, 62

William H Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007. 5, 21, 36

Michael Rabinovich, Roi Poranne, Daniele Panozzo, and Olga Sorkine-Hornung. Scalable locally injective mappings. *ACM Trans. Graph.*, 36(2):16, 2017. 8

A.R. Rivers and D.L. James. FastLSM: fast lattice shape matching for robust real-time deformation. *ACM Trans. Graph.*, 26:82:1–82:6, 2007. 8

Andrew Selle, Michael Lentine, and Ronald Fedkiw. A mass spring model for hair simulation. *ACM Trans. Graph.*, 27(3):64, 2008. 4

Eftychios Sifakis and Jernej Barbic. Fem simulation of 3d deformable solids: a practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses*, page 20. ACM, 2012. 5, 14, 28, 29, 53, 54, 63

Olga Sorkine. Least-squares rigid motion using svd. *Technical notes*, 120(3):52, 2009. 48

Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Symposium on Geometry processing*, volume 4, 2007. 39

Jos Stam. Nucleus: towards a unified dynamics solver for computer graphics. In *IEEE Int. Conf. on CAD and Comput. Graph.*, pages 1–11, 2009. 8

Ari Stern and Mathieu Desbrun. Discrete geometric mechanics for variational time integrators. In *ACM SIGGRAPH Courses*, pages 75–80. ACM, 2006. 6, 21

Jonathan Su, Rahul Sheth, and Ronald Fedkiw. Energy conservation for the simulation of deformable bodies. *IEEE transactions on visualization and computer graphics*, 19(2): 189–200, 2013. 76, 79

Rasmus Tamstorf, Toby Jones, and Stephen F. McCormick. Smoothed aggregation multigrid for cloth simulation. *ACM Trans. Graph.*, 34:245:1–245:13, 2015. 6

Yun Teng, Miguel A. Otaduy, and Theodore Kim. Simulating articulated subspace self-contact. *ACM Trans. Graph.*, 33:106:1–106:9, 2014. 7

Yun Teng, Mark Meyer, Tony DeRose, and Theodore Kim. Subspace condensation: Full space adaptivity for subspace deformations. *ACM Trans. Graph.*, 34:76:1–76:9, 2015. ISSN 0730-0301. 7

Joseph Teran, Eftychios Sifakis, Geoffrey Irving, and Ronald Fedkiw. Robust quasistatic finite elements and flesh simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 181–190. ACM, 2005. 2

Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *Computer Graphics (Proceedings of SIGGRAPH)*, volume 21, pages 205–214, 1987. 4, 6

Matthias Teschner, Bruno Heidelberger, Matthias Muller, and Markus Gross. A versatile and robust model for geometrically complex deformable solids. In *Computer Graphics International, 2004. Proceedings*, pages 312–319. IEEE, 2004. 4

Bernhard Thomaszewski, Simon Pabst, and Wolfgang Straßer. Continuum-based strain limiting. In *Comput. Graph. Forum*, volume 28, pages 569–576, 2009. 5

Maxime Tournier, Matthieu Nesme, Benjamin Gilles, and François Faure. Stable constrained dynamics. *ACM Trans. Graph.*, 34:132:1–132:10, 2015. 5, 6, 79

KC Valanis and Robert F Landel. The strain-energy function of a hyperelastic material in terms of the extension ratios. *Journal of Applied Physics*, 38(7):2997–3002, 1967. 3, 5, 54

Huamin Wang. A chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Trans. Graph.*, 34:246:1–246:9, 2015. 78

Huamin Wang and Yin Yang. Descent methods for elastic body simulation on the gpu. *ACM Transactions on Graphics (TOG)*, 35(6):212, 2016. 78

Huamin Wang, James O'Brien, and Ravi Ramamoorthi. Multi-resolution isotropic strain limiting. *ACM Trans. Graph.*, 29:156:1–156:10, 2010. 6

Hongyi Xu, Funshing Sin, Yufeng Zhu, and Jernej Barbič. Nonlinear material design using principal stretches. *ACM Trans. Graph.*, 34(4):75, 2015. 1, 3, 5, 54, 60