RWTH AACHEN UNIVERSITY

The present work was submitted to the

Visual Computing Institute
Faculty of Mathematics, Computer Science and Natural Sciences
RWTH Aachen University

Comparison of XPBD and Projective Dynamics

Master Thesis

presented by

Dennis Ledwon
Student ID Number 370425

July 2024

First Examiner: Prof. Dr. Jan Bender
Second Examiner: Prof. Dr. Torten Kuhlen

Hiermit versichere ich, diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht zu haben.

I hereby affirm that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Aachen, July 13, 2024

Dennis Ledwon

Contents

Chapter 1

Introduction

Visually pleasing simulations of deformable bodies are the cornerstone of many applications of computer graphics like animated movies and video games. This creates the need for simulation procedures that are capable of handling a variety of non-linear materials including rubber and biological soft tissue such as skin, fat and muscles. Realistic results can be achieved through physics-based simulation, where a body's response to deformation is calculated in accordance with the equations of motion and its physical material model which associates local deformations with energy potentials. Additionally, handling common interactions such as collisions, joints or simply attaching different bodies to each other requires the ability to exert more direct control over the positions of the simulated bodies. Ideally, this is achieved by introducing hard constraints are guaranteed to be satisfied at all time into the simulation.

In real-time applications, the physics update often needs to be performed during a single frame. Since the accurate physics-based simulation is too expensive to fit into the time budget for a single frame, physics-based simulation methods that are suitable for real-time applications trade accuracy and generality for speed. Two popular approaches for real-time physics-based simulations are XPBD and Projective Dynamics (PD). In XPBD, energy terms are expressed in terms of compliant constraints whose contributing particles are moved to closer to physically valid positions by independent constraint projections. On the other hand, PD treats numerical integration of the equations of motion as an optimization problem that can be solved efficiently by restricting energy potentials to a special structure. PD can be extended to more general energy potentials by reinterpreting it through the lense of constrained and unconstrained optimization.

Chapter 2

Related Work

Chapter 3

Foundations

In most approaches for the simulation of physical systems, the motion of the system is assumed to be in accordance with Newton's laws of motion. Due to Newton's second law

$$\boldsymbol{f} = m\boldsymbol{a}, \tag{3.1}$$

it is possible to relate the force $\boldsymbol{f}$ acting on a particle and the resulting acceleration $\boldsymbol{a}$ via the particle mass m. The motion of a particle system can then be described in terms of a system of ordinary differential equations (ODEs). This system of ODEs is commonly referred to as the equations of motion. The equations of motion are integrated over time to arrive at the configuration of the system at the next time step. For general forces, finding an analytical solution to the equations of motion is impossible and numerical integration schemes must be employed instead. In particular, with the exception of PBD, the solvers discussed in this thesis (see Ch. 4) are based on a numerical integration technique called implicit Euler integration [MMC16; BML+14]. In this chapter, we introduce the foundations required to simulate rich effects including elasticity and geometric constraints based on numerical integration of the equations of motion.

The equations of motion due to Newton's second law are reviewed in Section 3.1. Common approaches for numerical integration are introduced in Section 3.2. An alternative formulation of implicit Euler integration as an unconstrained optimization problem called the variational form of implicit Euler integration is presented in Section 3.3. Additional considerations for the numerical integration of the equations of motion in real-time settings are covered in Section 3.4. Simulating deformable bodies via numerical integration requires the computation of elastic forces from deformed configurations. We describe how to achieve this in Section 3.5. The exact elastic forces do not only depend on the deformation of the body, but also on its material. A brief introduction to relevant material models is provided in Section 3.6. Lastly, it is often desirable to exert direct control over

the positions of individual particles by imposing constraints on the simulation. Constrained dynamical simulation is covered in Section 3.7.

3.1 The Equations of Motion

The motion of a spatially discretized system with m particles evolving in time according to Newton's laws can be modeled via the equations of motion

$$\begin{aligned} \dot{\boldsymbol{q}}(t) &= \boldsymbol{v}(t) \\ \dot{\boldsymbol{v}}(t) &= \mathbf{M}^{-1}\boldsymbol{f}(\boldsymbol{q}(t), \boldsymbol{v}(t)), \end{aligned} \tag{3.2}$$

where $\boldsymbol{q}(t)$, $\boldsymbol{v}(t)$, $\boldsymbol{f}(\boldsymbol{q}(t), \boldsymbol{v}(t))$ are the particle positions, particle velocities and forces acting on each particle at time t, respectively, and $\mathbf{M}$ is a diagonal matrix with the particle masses as diagonal entries. From now on, we only consider forces that are independent of velocities and write $\boldsymbol{f}(\boldsymbol{q}(t))$ instead of $\boldsymbol{f}(\boldsymbol{q}(t), \boldsymbol{v}(t))$. Depending on the context, it is $\boldsymbol{q}(t), \boldsymbol{v}(t), \boldsymbol{f}(\boldsymbol{q}(t)) \in \mathbb{R}^{m\times 3}$ and $\mathbf{M} \in \mathbb{R}^{m\times m}$ or $\boldsymbol{q}(t), \boldsymbol{v}(t), \boldsymbol{f}(\boldsymbol{q}(t)) \in \mathbb{R}^{3m}$ and $\mathbf{M} \in \mathbb{R}^{3m\times 3m}$. We define $\dot{\boldsymbol{q}}(t) \coloneqq \frac{d\boldsymbol{q}}{dt}(t)$ and $\dot{\boldsymbol{v}}(t) \coloneqq \frac{d\boldsymbol{v}}{dt}(t)$. When the time step t is obvious from the context, we write $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ instead of $\boldsymbol{q}(t)$ and $\dot{\boldsymbol{q}}(t)$ for time-dependent quantities for the sake of brevity.

The positions $\boldsymbol{q}(t)$ and velocities $\boldsymbol{v}(t)$ at time t can be determined by solving the equations of motion given some initial conditions $\boldsymbol{q}(0) = \boldsymbol{q}_0$ and $\boldsymbol{v}(0) = \boldsymbol{v}_0$. Here, $\boldsymbol{q}_0$ and $\boldsymbol{v}_0$ are the initial positions and velocitities, respectively. A function $\boldsymbol{s}$ is a solution of the equations of motion if

$$\boldsymbol{s}(0) = \begin{pmatrix} \boldsymbol{q}(0) \\ \boldsymbol{v}(0) \end{pmatrix} \text{ and } \dot{\boldsymbol{s}} = \begin{pmatrix} \dot{\boldsymbol{q}} \\ \dot{\boldsymbol{v}} \end{pmatrix}.$$

Given an analytical solution $\boldsymbol{s}$, the positions and velocities of the simulated system at time t can be determined by computing $\boldsymbol{s}(t)$. For general nonlinear forces, analytical solutions of the equations of motion are usually not available. Thus, the equations of motion need to be solved numerically.

3.2 Numerical Integration of The Equations of Motion

Numerical integration schemes for the equations of motion aim at approximating the true positions $\boldsymbol{q}(t_n)$ and velocities $\boldsymbol{v}(t_n)$ at discrete points in time $t_n, n \in \mathbb{N}$ without computing the analytical solution $\boldsymbol{s}$ directly. In the context of this thesis,

$t_n = nh$ for some time step $h \in \mathbb{R}$. Note that the time step is constant. The approximations of $\boldsymbol{q}(t_n)$ and $\boldsymbol{v}(t_n)$ produced by the integration scheme are referred to as $\boldsymbol{q}_n$ and $\boldsymbol{v}_n$, respectively. In the following, we introduce explit, symplectic and implicit Euler integration in Section 3.2.1, 3.2.2 and 3.2.3, respectively.

3.2.1 Explicit Euler Integration

When applied to Equation 3.2, the next position and velocity estimates $\boldsymbol{q}_{n+1}$ and $\boldsymbol{v}_{n+1}$ are derived from the current estimates $\boldsymbol{q}_n$ and $\boldsymbol{v}_n$ via the update formula

$$\begin{aligned}\boldsymbol{q}_{n+1} &= \boldsymbol{q}_n + h\boldsymbol{v}_n \\ \boldsymbol{v}_{n+1} &= \boldsymbol{v}_n + h\mathbf{M}^{-1}\boldsymbol{f}(\boldsymbol{q}_n).\end{aligned}$$

The idea is to simplify the integration of the functions $\dot{\boldsymbol{q}}, \dot{\boldsymbol{v}}$ over the time step h by using constant approximations. Then, time integration is as simple as multiplying the constant function values with the time step h. In the explicit Euler method, we approximate $\dot{\boldsymbol{q}}, \dot{\boldsymbol{v}}$ by their current estimates $\boldsymbol{q}_n, \boldsymbol{v}_n$.

One main criterion for evaluating numerical integration schemes is the global trunction error [CC05], which is the error between $(\boldsymbol{q}_n^T, \boldsymbol{v}_n^T)^T$ and $(\boldsymbol{q}(t_n)^T, \boldsymbol{v}(t_n)^T)^T$ incurred during n applications of the update formula. The global trunction error is the sum of the local trunction error (the error that results in the application of the method over a single time step h) and the propagated truncation error (the error resulting from the approximations produced during the previous steps). It can be shown that the global truncation error of explicit Euler integration over n iterations is $\mathcal{O}(nh)$. Thus, explicit Euler integration is a first-order method.

The main drawback of explicit Euler integration is that its estimates $\boldsymbol{q}_n$ and $\boldsymbol{v}_n$ can often diverge with growing n unless prohibitely small time steps are used, even if the true functions $\boldsymbol{q}$ and $\boldsymbol{v}$ are bounded [CC05]. Such integration schemes are called unstable. In the context of phyiscal simulations, it is also desirable that numerical integration schemes preserve important physical invariants of the equations of motion such as the conservation of momentum and energy. However, in the absence of artificial damping, explicit Euler integration amplifies the energy of the simulated system. For example, it fails unconditionally on the undampened harmonic oscillator [SLM06]. Using the explicit Euler method with larger time steps often manifests itself in exploding simulations. These issues are particularly pronounced during the simulation of stiff systems, i.e. systems with large accelerations.

3.2.2 Symplectic Euler Integration

A variation of the explicit Euler method applied to the equations of motion, called the symplectic Euler method, arises when the new velocities $\boldsymbol{v}_{n+1}$ instead of the old velocities $\boldsymbol{v}_n$ are used in the position update [SD06]. This leads to the update formula

$$\begin{aligned} \boldsymbol{q}_{n+1} &= \boldsymbol{q}_n + h\boldsymbol{v}_{n+1} \\ \boldsymbol{v}_{n+1} &= \boldsymbol{v}_n + h\mathbf{M}^{-1}\boldsymbol{f}(\boldsymbol{q}_n). \end{aligned} \tag{3.3}$$

The symplectic Euler method has the same computational cost as the explicit Euler method. Just like explicit Euler integration, it is a first-order method. However, while it still does not conserve the system's energy exactly, it conserves a quadratic form that is close [SLM06]. In contrast to the explicit Euler method, the symplectic Euler method preserves the amplitude of a swinging pendulum with appropriate time steps [SD06]. Its main drawback is that it also becomes unstable for stiff simulations unless the time step is kept prohibitively small [SLM06].

3.2.3 Implicit Euler Integration

Another popular integration scheme for tackling the equations of motion is implicit Euler integration, defined by the update formula

$$\begin{aligned} \boldsymbol{q}_{n+1} &= \boldsymbol{q}_n + h\boldsymbol{v}_{n+1} \\ \boldsymbol{v}_{n+1} &= \boldsymbol{v}_n + h\mathbf{M}^{-1}\boldsymbol{f}(\boldsymbol{q}_{n+1}). \end{aligned} \tag{3.4}$$

Note how $\boldsymbol{q}_{n+1}$ and $\boldsymbol{v}_{n+1}$ appear on both sides of the equations. Consequently, performing implicit Euler integration includes solving a system of algebraic equations. For general nonlinear forces, the entire system of equations is nonlinear. Thus, implicit Euler integration is more computationally expensive than both explicit and symplectic Euler integration. Despite the added complexity, implicit Euler integration is still only first-order accurate. However, implicit Euler integration remains a popular choice for physical simulation since it can be shown to be unconditionally stable [CC05]. This allows dramatically increasing the size of the time steps. The additional cost of a single implicit Euler step compared to the previously mentioned integration schemes is offset by the fact that fewer steps are required to drive the simulation forward the same amount of time in a stable manner. It is worth poiting out that the stability guarantees of implicit Euler integration require the computation of the exact solution of the NLSE in Equation 3.4. Solving the NLSE commonly involves solving derived LSEs whose system matrices are often ill-conditioned if the equations of motion are stiff (see Sec. 3.7.1).

Thus, implicit Euler integration is still prone to numerical instabilities in practice. Finally, implicit Euler integration is known to exhibit numerical damping [SD06]. This manifests itself in a loss of energy in the simulated system. For example, the amplitude of a swinging pendulum decreases if integrated using the implicit Euler method. It is important to note that the extent of the observed numerical damping depends on the time step h, making it difficult to control.

This brief discussion of numerical integration schemes shows that each of the introduced methods comes with their own sets of advantages and drawbacks. In particular, each of the methods trade off stability in favor of the preservation of physical invariants such as the conservation of energy or vice versa. Additionally, numerical integration of stiff system poses challenges for each of the methods introduced here. While explicit and symplectic Euler integration are analytically unstable without prohibitively small time steps or additional precautions, implicit Euler integration is prone to numerical instabilities for stiff systems. More sophisticated methods for physical simulation (see Sec. 3.7) build on top of the integration schemes introduced here with the goal of alleviating their drawbacks without sacrificing their more advantageous properties.

3.3 Variational Form of Implicit Euler Integration

Instead of solving the system of equations in Equation 3.4 directly, it is possible to approach implicit Euler integration via an equivalent unconstrained minimization problem over the particle positions at the next time step. This formulation is called the variational form of implicit Euler integration [BML+14]. We summarize the derivation of the optimization problem according to Bouaziz et al. [BML+14] below.

By rewriting the first line of Equation 3.4 as

$$\boldsymbol{v}_{n+1} = \frac{1}{h}(\boldsymbol{q}_{n+1} - \boldsymbol{q}_n)$$

and substituting into the velocity update of Equation 3.4 the following equation can be derived

$$\mathbf{M}(\boldsymbol{q}_{n+1} - \boldsymbol{q}_n - h\boldsymbol{v}_n) = h^2(\boldsymbol{f}(\boldsymbol{q}_{n+1})). \tag{3.5}$$

We separate forces $\boldsymbol{f}(\boldsymbol{q})$ into internal forces $\boldsymbol{f}_{\text{int}}(\boldsymbol{q}) = \sum_{i \in \mathcal{I}_{\text{int}}} \boldsymbol{f}^i_{\text{int}}(\boldsymbol{q})$ and external forces $\boldsymbol{f}_{\text{ext}}(\boldsymbol{q}) = \sum_{i \in \mathcal{I}_{\text{ext}}} \boldsymbol{f}^i_{\text{ext}}(\boldsymbol{q})$ with index sets $\mathcal{I}_{\text{int}}$ and $\mathcal{I}_{\text{ext}}$. We consider all external forces to be constant. Internal forces are conservative and defined in terms of scalar potential energy functions ψ_j via $\boldsymbol{f}^j_{\text{int}}(\boldsymbol{q}) = -\nabla\psi_j(\boldsymbol{q})$. Together, we have $\boldsymbol{f}(\boldsymbol{q}) = \boldsymbol{f}_{\text{int}}(\boldsymbol{q}) + \boldsymbol{f}_{\text{ext}} = -\sum_j \nabla\psi_j(\boldsymbol{q}) + \boldsymbol{f}_{\text{ext}}$. Plugging into Equation 3.5, it is

$$\mathbf{M}(\boldsymbol{q}_{n+1} - \boldsymbol{q}_n - h\boldsymbol{v}_n) = h^2 \left(\boldsymbol{f}_{\text{ext}} - \sum_j \nabla \psi_j(\boldsymbol{q}_{n+1}) \right). \tag{3.6}$$

By computing first-order optimality conditions, it is easily verified that the system of equations above is equivalent to the optimization problem

$$\min_{\boldsymbol{q}_{n+1}} \frac{1}{2h^2} \left\| \mathbf{M}^{\frac{1}{2}} (\boldsymbol{q}_{n+1} - \tilde{\boldsymbol{q}}) \right\|_F^2 + \sum_j \psi_j(\boldsymbol{q}_{n+1}). \tag{3.7}$$

where $\tilde{\boldsymbol{q}} = \boldsymbol{q}_n + h\boldsymbol{v}_n + h^2 \mathbf{M}^{-1} \boldsymbol{f}_{\text{ext}}$. We refer to the first and second term of the objective function as the inertial term and the constraint energy, respectively [BML+14]. Thus, the minimization problem requires that the solution minimizes the elastic deformation as best as possible while ensuring that the solution is close to following its momentum plus external forces. The weighting between the inertial term and the constraint energy depends on the particle masses $\mathbf{M}$, the time step h and the material stiffness of the elastic potentials ψ_j. Larger masses and smaller time steps increase the weighting of the inertial term, whereas larger material stiffness increases the weighting of the constraint energy. The solution preserves linear and angular momentum as long as the elastic potentials are rigid motion invariant [BML+14].

Approaching implicit Euler integration in its variational form can often be advantageous. That is because the objective function of Equation 3.7 presents a natural merit function that can be employed to improve the step size along the search direction that arises in common unconstrained optimization algorithms based on Newton's method [NW06]. As an example, the objective function is used in many step length selection algorithms to ensure that step sizes satisfy the Wolfe conditions. In fact, many algorithms for solving nonlinear systems of equations construct approximate merit functions if the objective function from an equivalent optimization problem is not available [NW06].

3.4 Considerations for Real-Time Simulations

As discussed in Section 3.2, both the explicit and symplectic Euler integrators exhibit instabilities when simulating stiff systems without keeping the time step h infeasibly small or adding artificial damping terms. On the other hand, implicit Euler integration is unconditionally stable, even though numerical instabilities might still occur. Since unstable simulations cannot simply be restarted and recovered from in the real-time setting, stability is of utmost importance. For that reason, physical simulations for real-time applications are commonly based on

implicit Euler integration. However, this comes at the cost of artificial numerical damping and significant computational overhead. According to Tournier et al. [TNGF15], it is common that the time budget allotted for integration over one time step is barely enough to perform a single linearization of the NLSE in Equation 3.4. Thus, additional measures are required to make implicit Euler integration viable for real-time applications.

There are two main strategies for speeding up the integration step. The first one is to simply approximate the solution of the NLSE in Equation 3.4. In the simplest case, this consists of solving a linearized version of Equation 3.4 and updating the new positions $\boldsymbol{q}_{n+1}$ and velocities $\boldsymbol{v}_{n+1}$ with the resulting values without iterating any further [BW98]. Applying first-order Taylor expansion to the forces yields the following LSE

$$\begin{aligned} \Delta\boldsymbol{q} &= h\boldsymbol{v}_{n+1} \\ \Delta\boldsymbol{v} &= h\mathbf{M}^{-1}(\boldsymbol{f}(\boldsymbol{q}_n) + \nabla_q \boldsymbol{f}(\boldsymbol{q}_n)\Delta\boldsymbol{q}), \end{aligned} \tag{3.8}$$

where $\Delta\boldsymbol{q} = \boldsymbol{q}_{n+1} - \boldsymbol{q}_n$ and $\Delta\boldsymbol{v} = \boldsymbol{v}_{n+1} - \boldsymbol{v}_n$. By rearranging terms, it is easy to show that $\boldsymbol{v}_{n+1}$ can be determined by solving

$$(\mathbf{M} - h^2\mathbf{K})\boldsymbol{v}_{n+1} = \mathbf{M}\boldsymbol{v}_n + h\boldsymbol{f}(\boldsymbol{q}_n), \tag{3.9}$$

where $\mathbf{K} = \frac{d\boldsymbol{f}}{d\boldsymbol{q}}$. The new positions can be computed via $\boldsymbol{q}_{n+1} = h\boldsymbol{v}_{n+1}$. Borrowing the terminology from [SLM06], we refer to this approach as the linear implicit Euler stepper. Note that iteratively solving the linearized system of equations in Equation 3.8 and updating $\boldsymbol{q}_{n+1}, \boldsymbol{v}_{n+1}$ as described converges to the true solution of the NLSE for implicit Euler integration (see Eq. 3.4). The strategy of approximating the solution of Equation 3.4 is also at the heart of XPBD (see Sec. 4.1.5), even though the manner in which the approximation is achieved differs quite significantly from the linear implicit Euler stepper.

The second strategy is to compute the exact solution of Equation 3.4, but to restrict to a subset of forces whose strucuture allows speeding up the process of solving the NLSE. As an example, note that Equation 3.4 is linear if the featured forces are linear. In this case, $\boldsymbol{q}_{n+1}, \boldsymbol{v}_{n+1}$ can be determined in a single linear solve. A similar approach, but with a less restrictive subset of forces that allows modelling nonlinearities, is used in PD (see Sec. 4.2). The second method is particularly appealing in settings where visual plausibility is prioritized over physical accuracy.

3.5 Elastic Forces For Deformable Bodies

To be able to perform physically accurate simulations of deformable bodies using the numerical integration schemes for the equations of motion introduced in Section 3.2, we need to be able to compute elastic forces $\boldsymbol{f}(\boldsymbol{q})$ from the discretized positions $\boldsymbol{q}$ of a potentially deformed body. Note that the we assume that the elastic forces $\boldsymbol{f}(\boldsymbol{q})$ are determined entirely by the current positions $\boldsymbol{q}$. In particular, this implies that the elastic forces are path-independent. Such materials are called hyperelastic. In the following exposition, we lean heavily on the SIGGRAPH course by Sifakis et al. [SB12]. We start by introducing relevant concepts from continuum mechanics and then describe how efficient implementation can be achieved via suitable discretization schemes.

First, we introduce the deformation function $\boldsymbol{\phi}\colon \Omega \mapsto \mathbb{R}^3$, where $\Omega \subseteq \mathbb{R}^3$ is the volumetric domain occupied by the undeformed body. This domain Ω is called the reference configuration. The deformation function maps every point in the reference configuration to its respective deformed position $\boldsymbol{q} = \boldsymbol{\phi}(\boldsymbol{Q})$. Note that positions in the reference configuration are denoted using capital letters while the deformed positions are referred to via small letters. As an example, the deformation function given by

$$\boldsymbol{\phi}(\boldsymbol{Q}) = 2\mathbf{I}\boldsymbol{Q}$$

describes the deformed configuration that is achieved by stretching the reference domain by a factor of 2 in each direction.

The next step is to associate the deformation function that describes the current deformed configuration with an energy. However, the degree to which different parts of the body are deformed may vary. Thus, we need to be able to relate deformations and strain energies on a local scale. This can be achieved by defining an energy density function $\Psi(\boldsymbol{\phi}, \boldsymbol{Q})$ which measures the strain energy per unit undeformed volume on an infitisemal domain dV around $\boldsymbol{Q}$. Since the energy density function only depends on the local behavior of $\boldsymbol{\phi}$ around $\boldsymbol{Q}$, it can be equivalently expressed in terms of the deformation gradient $\mathbf{F}(\boldsymbol{Q}) \in \mathbb{R}^{3\times 3}$ given by

$$\mathbf{F} = \mathbf{F}(Q) = \frac{d\boldsymbol{\phi}}{d\boldsymbol{Q}}, \tag{3.10}$$

yielding $\Psi(\boldsymbol{\phi}, \boldsymbol{Q}) = \Psi(\mathbf{F}(\boldsymbol{Q}))$. The choice of the energy density depends on the simulated material and is treated as a black box for now. Relevant material models are introduced in Section 3.6. Now, the elastic energy stored in the entirety of the deformed body can be computed by integrating the energy density function over the reference configuration Ω like so

$$E(\boldsymbol{\phi}) = \int_{\Omega} \Psi(\mathbf{F}) d\boldsymbol{Q}. \tag{3.11}$$

For general nonlinear materials, computing the integral in Equation 3.11 is infeasible. Thus, simplifying assumptions are required in order to enable efficient simulation of the physics of deformable bodies. Firstly, we describe the reference configuration Ω as a tetrahedral mesh. Let $\boldsymbol{Q} = (\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_n)^T$ be the vertex positions of the undeformed mesh. Then, the current state of the model can be expressed via the vertex positions in the deformed configuration $\boldsymbol{q} = (\boldsymbol{\phi}(\boldsymbol{Q}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{Q}_n))^T = (\boldsymbol{q_1}, \ldots, \boldsymbol{q}_n)^T$. Instead of using the true, complex deformation function $\boldsymbol{\phi}$, we construct an approximation $\hat{\boldsymbol{\phi}}$ from $\boldsymbol{Q}$ and $\boldsymbol{q}$ using barycentric interpolation. The reconstructed deformation map $\hat{\boldsymbol{\phi}}$ is a piecewise linear function over each tetrahedron given by

$$\hat{\boldsymbol{\phi}}(\boldsymbol{Q}) = \mathbf{F}_i \boldsymbol{Q} + \boldsymbol{b}_i \text{ for all } \boldsymbol{Q} \in \boldsymbol{\mathcal{T}_i},$$

where $\mathbf{F}_i \in \mathbb{R}^{3\times3}$ and $\boldsymbol{b}_i$ are specific to each tetrahedron $\mathcal{T}_i$. Note that $\mathbf{F}_i$ can be interpreted as the constant deformation gradient for all positions in the i-th tetrahedron. Let $\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_4$ and $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_4$ denote the positions of the vertices of tetrahedron $\mathcal{T}_i$ in the reference configuration and the deformed configuration, respectively. Sifakis [SB12] shows that $\mathbf{F}_i$ is given by

$$\mathbf{F}_i = \mathbf{D}_s \mathbf{D}_m^{-1} \tag{3.12}$$

where

$$\mathbf{D}_s = \begin{pmatrix} q_1^x - q_4^x & q_2^x - q_4^x & q_3^x - q_4^x \\ q_1^y - q_4^y & q_2^y - q_4^y & q_3^y - q_4^y \\ q_1^z - q_4^z & q_2^z - q_4^z & q_3^z - q_4^z \end{pmatrix}$$

is the deformed shape matrix and

$$\mathbf{D}_m = \begin{pmatrix} Q_1^x - Q_4^x & Q_2^x - Q_4^x & Q_3^x - Q_4^x \\ Q_1^y - Q_4^y & Q_2^y - Q_4^y & Q_3^y - Q_4^y \\ Q_1^z - Q_4^z & Q_2^z - Q_4^z & Q_3^z - Q_4^z \end{pmatrix}$$

is called the reference shape matrix.

Now, the elastic energy of the deformed body can be computed as a sum over the tetrahedral elements of the mesh via

$$E(\boldsymbol{q}) = \sum_e E_e(\boldsymbol{q}) = \sum_e \int_{\Omega_e} \Psi(\hat{\mathbf{F}}(\boldsymbol{Q}, \boldsymbol{q})) d\boldsymbol{Q},$$

where $\hat{\mathbf{F}}(\boldsymbol{Q}, \boldsymbol{q})$ is the deformation gradient of the reconstructed deformation function $\hat{\boldsymbol{\phi}}$. Since the deformation gradient $\hat{\mathbf{F}}$ is constant across each tetrahedral element, it is further

$$E(\boldsymbol{q}) = \sum_e V_e \Psi(\mathbf{F}_e),$$

where V_e is the volume of the undeformed tetrahedral element e. In contrast to Equation 3.11, this can be computed efficiently. Finally, we are able to specify the elastic forces $\boldsymbol{f}(\boldsymbol{q})$ acting on the particles of the deformed body with positions $\boldsymbol{q}$ by computing

$$\boldsymbol{f}(\boldsymbol{q}) = -\frac{dE(\boldsymbol{q})}{d\boldsymbol{q}}.$$

3.6 Material Models

In Section 3.5, we demonstrated how to compute the elastic forces acting on a particle system with deformed positions $\boldsymbol{q}$, but treated the material-dependent energy density function Ψ as a black box. In this section, we fill in the gaps and describe the material models relevant for this thesis. Note that the material models covered here are isotropic, i.e. their energy density functions satisfy $\Psi(\mathbf{F}) = \Psi(\mathbf{RF})$ for any rotation matrix $\mathbf{R}$ [SB12]. Informally, this means that the response to a deformation is independent of the orientation the deformation is applied in. Additionally, we focus on material models that generate elastic forces that are nonlinear in the deformation gradient. In the following, we introduce the strain material model [BML+14], the classical Neohookean model and a simplified variation thereof [SGK18].

3.6.1 Strain Material Model

The first material model, which we simply refer to as the strain material model [BML+14], is given by

$$\Psi_{\text{strain}}(\boldsymbol{F}) = \frac{k}{2}\left\|\mathbf{U}\mathbf{I}\mathbf{V}^T - \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T\right\|_F^2 = \frac{k}{2}\|\mathbf{I} - \boldsymbol{\Sigma}\|_F^2\,, \tag{3.13}$$

where $\mathbf{F} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$ is the singular value decomposition of the deformation gradient $\mathbf{F}$ and $k \in \mathbb{R}^+$ is a stiffness parameter. Note that $\Psi_{\text{strain}}(\boldsymbol{F}) = 0$ iff $\boldsymbol{\Sigma} = \mathbf{I}$ or equivalently iff $\mathbf{F}$ is a rotational matrix. It is easy to verify that $\mathbf{F}$ is a rotation iff the deformation function $\boldsymbol{\phi}$ is a rigid body transform. Thus, Equation 3.13 can be

interpreted as a quadratic penalty for the distance between the deformation function ϕ and its closest rigid-body transform as measured by the Frobenius norm of the difference of their deformation gradients.

Since the PD solver cannot be applied to classical material models like the St. Venant-Kirchhoff and the Neohookean model, the strain material model is the main tool for achieving simulations of deformable bodies using PD (see Sec. 4.2). While simple, the strain material model comes with a couple of disadvantages. Firstly, the computation of the singular value decomposition of $\mathbf{F}$ is expensive. Secondly, the restorative elastic forces arising from the strain material model are proportional to the aforementioned distance between the deformation and its closest rigid-body transform. As such, the strain material model is too restrictive to model complex materials such as skin or flesh [LBK17].

3.6.2 Neohookean Material Model

A popular material model that is more suitable for simulating biological materials is the Neohookean material model and its variations [SGK18]. Its popularity is reflected by the fact that it has been implemented with a variety of different solvers including PBD [BKCW14], XPBD [MM21], ADMM [OBLN17] and a quasi-Newton solver based on PD [LBK17]. The classical version of the Neohookean material model is given below

$$\Psi_{\text{Neo}}(\mathbf{F}) = \frac{\mu}{2}(\text{tr}(\mathbf{F}^T\mathbf{F}) - 3) - \mu\log(\det(\mathbf{F})) + \frac{\lambda}{2}\log(\det(\mathbf{F}))^2. \tag{3.14}$$

Here, μ and λ are the material-specific Lamé parameters. The first term is minimized if $\mathbf{F} = \mathbf{0}$, which happens if the deformed volume has collapsed to a single point. Note that $\det(\mathbf{F})$ can be considered a measure for the infitisemal local change in volume caused by the deformation ϕ. Then, the last two terms can be interpreted as volume-preserving, since $\log(\det(\mathbf{F}))$ approaches negative infinity as $\det(\mathbf{F})$ approaches 0. In particular, the volume-preserving terms strongly counteract the compression of the modelled material encouraged by the first term. This way, the classical Neohookean model achieves rest stability.

The main issue with the Neohookean material model is that the volume-preserving terms can grow unbounded when $\det(\mathbf{F})$ approaches 0 and are infeasible for $\det(\mathbf{F}) < 0$. Note that while configurations where $\det(\mathbf{F}) < 0$ are theoretically impossible to achieve, they can often occur in practice as a result of geometric constraints, instability of time integration techniques or inadequate convergence of numerical solvers [SB12]. Thus, Ψ_{Neo} is not inversion stable [SGK18].

3.6.3 Simplified Neohookean Material Model

To address the issues with the classical Neohookean material model, Smith et al. propose a modified version of Ψ_{Neo} where the volume-preserving terms are replaced by the bounded, well-defined and invertible term $\frac{\lambda}{2}(\det(\mathbf{F}) - 1)^2$. This yields the following energy density

$$\Psi(\mathbf{F}) = \frac{\mu}{2}(\text{tr}(\mathbf{F}^T\mathbf{F}) - 3) + \frac{\lambda}{2}(\det(\mathbf{F}) - 1)^2.$$

While this ensures inversion stability, Smith et al. show that its Piola-Kirchhoff tensor

$$\mathbf{P}(\mathbf{F}) = \frac{d\Psi(\mathbf{F})}{d\mathbf{F}}$$

is non-zero at undeformed configurations, i.e. $\mathbf{P}(\mathbf{I}) \neq \mathbf{0}$. Consequently, Ψ is not rest stable. In particular, the effective rest state is achieved when elements are slightly deflated. The authors correct this by adjusting the volume-preserving term so that elements are inflated again, yielding the following energy density

$$\Psi_{\text{Smith}}(\mathbf{F}) = \frac{\mu}{2}(\text{tr}(\mathbf{F}^T\mathbf{F}) - 3) + \frac{\lambda}{2}\left(\det(\mathbf{F}) - \left(1 + \frac{\mu}{\lambda}\right)\right)^2. \tag{3.15}$$

We refer to this material model as the simplified Neohookean model.

Equipped with these energy density functions, we have all the tools available that are required to compute the elastic forces acting on the particles of a deformed body as described in Section 3.5. Note that the material models covered here are by no means exhaustive. Again, the SIGGRAPH tutorial by Sifakis [SB12] is a good place to gain a better overview for the interested reader.

3.7 Constrained Dynamic Simulation

Visually pleasing and physically accurate simulations of dynamical systems require modeling a wide range of materials and interactions between participating bodies. Common requirements that prove challenging are the ability to simulate stiff materials and to enforce hard constraints on the positions of the simulated bodies. Among others, hard constraints involve attaching particles to a fixed point in space or to each other, implementing joints and resolving interpenetrations between bodies. Unfortunately, in their naive form, the numerical integration schemes from Section 3.2 are unsuitable for application to simulations including these effects. The main difficulty lies in the fact that stiff materials and

hard constraints introduce stiffness into the simulated system, causing stability issues and numerical problems. This is obvious for stiff materials, where even small deformations can cause large restorative forces. For hard constraints, note that to fit into the equations of motion in Equation 3.2, corresponding forces that ensure that the constraints stay satisfied at all times need to be defined. It is possible to show that hard constraints are the physical limit of strong potential forces [SLM06], increasing the stiffness of the equations of motion.

In this section, we focus on different strategies for modelling hard constraints in physical real-time simulations and discuss their properties. Heavily inspired by Tournier et al. [TNGF15], the approaches covered in this section are analyzed in the context of real-time adaptions of implicit Euler integration that only perform a single linear solve per time step (see Sec. 3.4). While both XPBD (see Sec. 4.1.5) and PD (see Sec. 4.2) differ significantly from these methods, the challenges encountered in modelling hard constraints are largely transferrable. Handling hard constraints via stiff penalty forces as outlined above is covered in more detail in Section 3.7.1. Adding hard constraints to the equations of motion without introducing stiff constraint forces via a Differential Algebraic Equation (DAE) is explored in Section 3.7.2. Finally, adding regularization to the resulting DAE in a physcially meaningful manner yields a method called compliant constraints. Compliant constraints are introduced in Section 3.7.3.

3.7.1 Penalty Forces

As discussed above, it is possible to show that hard constraints are the physical limit of strong potential forces [SLM06]. Thus, the simplest approach to modelling hard constraints is to introduce strong forces that enforce that the constraints are satisfied at all times to the equations of motion. We demonstrate the downsides of this method by example of a distance constraint.

Consider a distance constraint C_{dist} that fixes the distance $l = ||\boldsymbol{q}_i - \boldsymbol{q}_j||$ between two particles $\boldsymbol{q}_i$ and $\boldsymbol{q}_j$ to some distance $d \in \mathbb{R}^+$. If $C_{\text{dist}}(\boldsymbol{q}) = l - d$, then the constraint is satisfied if $C_{\text{dist}}(\boldsymbol{q}) = 0$. Next, we define the energy potential $\psi_{\text{dist}}(\boldsymbol{q}) = \frac{k}{2} C_{\text{dist}}(\boldsymbol{q})^2$, where $k \in \mathbb{R}^+$ is some large stiffness value. Since the energy potential is zero if C_{dist} is satisfied and grows as the distance between $\boldsymbol{q}_i$ and $\boldsymbol{q}_j$ increases, the resulting force will drive the distance towards d. Upon closer inspection, ψ_{dist} is exactly the potential energy of a Hookean spring with stiffness k. The forces $\boldsymbol{f}_i$ and $\boldsymbol{f}_j$ applied to the particles $\boldsymbol{q}_i$ and $\boldsymbol{q}_j$ by the spring are given by $\boldsymbol{f}_i = -\boldsymbol{f}_j = \beta(\boldsymbol{q}_i - \boldsymbol{q}_j)/l$, where $\beta = -\partial\psi_{\text{dist}}/\partial l = kC_{\text{dist}}(\boldsymbol{q})$. Since $||\boldsymbol{f}_i|| = ||\boldsymbol{f}_j|| = \beta$, the restorative spring forces have infinite magnitude in the limit of infinite stiffness k if $C_{\text{dist}}(\boldsymbol{q}) \neq 0$.

Integration using the linear implicit Euler stepper (see Sec. 3.4) is achieved by plugging the spring forces into Equation 3.9 and solving for the new velocities $\boldsymbol{v}_{n+1}$. For the sake of convenience, we restate Equation 3.9 below

$$(\mathbf{M} - h^2\mathbf{K})\boldsymbol{v}_{n+1} = \mathbf{M}\boldsymbol{v}_n + h\boldsymbol{f}(\boldsymbol{q}_n).$$

Note that $\mathbf{K}$ is typically non-singular since elastic forces are invariant under rigid body transforms. When using large stiffness values k, the entries of $\mathbf{K}$ are large (due to large restorative forces) and dominate the entries of the system matrix

$$\mathbf{H} = \mathbf{M} - h^2\mathbf{K}. \tag{3.16}$$

In these cases, $\mathbf{H}$ is almost non-singular as well, leading to numerical issues and poor convergence for many solvers. Finally, recall that the inertial term of the objective function of the variational form of implicit Euler integration discourages moving particles away from their inertial positions during implicit Euler integration (see Sec. 3.3). Thus, if $C_{\text{dist}}(\boldsymbol{q}) \neq 0$, the spring stiffness k needs to be large enough to overcome both competing forces and the penalty for moving particles away from their inertial positions. As a result, it often takes multiple time steps to ensure that a distance constraint that was violated in the initial state is satisfied again.

In the example above, the potential energy ψ_{dist} is defined in terms of the constraint function C_{dist} via $\psi_{\text{spring}}(\boldsymbol{q}) = \frac{k}{2}C_{\text{dist}}(\boldsymbol{q})^2$. However, it is possible to generalize this approach to other geometric constraints C yielding energy potentials of the form

$$\psi(\boldsymbol{q}) = \frac{k}{2}C(\boldsymbol{q})^2. \tag{3.17}$$

The resulting forces are called penalty forces.

By constructing different geometric displacement functions, various properties such as the bending angle between triangles and in-plane shearing of triangles can be controlled via the corresponding quadratic energy potentials [BW98]. Geometric displacement functions with the desired effect are often intuitive and simple to define. However, as the corresponding energy potentials are not physically derived, choosing stiffness parameters that correspond to measurable physical properties of the simulated material and orchestrating multiple constraints becomes challenging [SLM06; NMK+06]. Finally, the numerical issues outlined for distance constraints generally apply to penalty forces derived from other geometric constraints as well.

3.7.2 Hard Constraints

In Section 3.7.1, we showed how to construct penalty forces from geometric constraint functions $C(\boldsymbol{q})$ that are satisfied if $C(\boldsymbol{q}) = 0$. Instead of constructing penalty forces, geometric hard constraints can also be enforced by amending the equations of motion with algebraic equations of the form $C(\boldsymbol{q}) = 0$. Additionally, constraint forces are required to link the algebraic constraint equations with the ODE describing the motion of the system. Let $\boldsymbol{C} = [C_1, \ldots, C_r]^T, r \in \mathbb{N}^+$ be a vector of constraints. Then, it can be shown that the constraint forces $\boldsymbol{f_C}$ applied to the particles have to be in the following form to avoid adding linear and angular momentum to the system,

$$\boldsymbol{f_C} = \nabla \boldsymbol{C}(\boldsymbol{q})^T \boldsymbol{\lambda} \tag{3.18}$$

where $\boldsymbol{\lambda}$ are the Lagrange multipliers of the constraints [Bar96]. Together, this leads to the system

$$\begin{aligned} \dot{\boldsymbol{q}} &= \boldsymbol{v} \\ \mathbf{M}\dot{\boldsymbol{v}} &= \boldsymbol{f}_{\text{ext}} + \nabla \boldsymbol{C}(\boldsymbol{q})^T \boldsymbol{\lambda} \\ \boldsymbol{C}(\boldsymbol{q}) &= \mathbf{0}, \end{aligned} \tag{3.19}$$

which is called a Differential Algebraic Equation (DAE) [UR95; Bar96].

Note that the constraint $\boldsymbol{C}(\boldsymbol{q})$ is formulated in terms of the particle positions $\boldsymbol{q}$, whereas the equations of motion work on velocities $\boldsymbol{v}$ and accelerations $\boldsymbol{a} \coloneqq \dot{\boldsymbol{v}}$. To make both formulations compatible with each other, the constraints need to be differentiated with respect to time once or twice so that the algebraic equation $\boldsymbol{C}(\boldsymbol{q}) = 0$ is restated in terms of $\boldsymbol{v}$ or $\boldsymbol{a}$. Differentiation of $\boldsymbol{C}(\boldsymbol{q}) = \mathbf{0}$ yields

$$\begin{aligned} \nabla \boldsymbol{C}(\boldsymbol{q})\boldsymbol{v} &= 0 \\ \nabla \boldsymbol{C}(\boldsymbol{q})\boldsymbol{a} &= \boldsymbol{c}(\boldsymbol{q}, \boldsymbol{v}) \end{aligned}$$

for some $\boldsymbol{c}(\boldsymbol{q}, \boldsymbol{v})$. The DAE can now be expressed as follows [UR95]

$$\begin{pmatrix} \mathbf{M} & -\nabla \boldsymbol{C}(\boldsymbol{q})^T \\ \nabla \boldsymbol{C}(\boldsymbol{q}) & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{a} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_{\text{ext}} \\ \boldsymbol{c}(\boldsymbol{v}). \end{pmatrix} \tag{3.20}$$

The lower block-row of the system is satisfied for accelerations that drive the system towards positions $\boldsymbol{q}$ for which $\boldsymbol{C}(\boldsymbol{q}) = \mathbf{0}$. This is indicated by the lower-right zero block in the system matrix. Thus, the system does not have a solution for contradictory constraints.

In [TNGF15], the authors suggest applying implicit integration schemes to the system directly resulting in the following LSE

$$\begin{pmatrix} \mathbf{M} & -\boldsymbol{C}(\boldsymbol{q}_n)^T \\ \boldsymbol{C}(\boldsymbol{q}_n) & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{v}_{n+1} \\ \boldsymbol{\mu}_{n+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{p} + h\boldsymbol{f}_{\text{ext}} \\ 0 \end{pmatrix}, \tag{3.21}$$

where $\mu = h\lambda$ and $\boldsymbol{p} = \mathbf{M}\boldsymbol{v}$. Here, the external forces $\boldsymbol{f}_{\text{ext}}$ and the constraint gradients $\boldsymbol{C}(\boldsymbol{q})$ are considered constant across the time step and $\boldsymbol{C}(\boldsymbol{q}_{n+1})$ is not approximated using Taylor expansion like it is in [BW98]. If internal forces are taken into account, the upper-left matrix $\mathbf{M}$ is replaced by the matrix $\mathbf{H}$ from Equation 3.16.

According to [SLM06], modelling hard constraints via the DAE in Equation 3.20 is notoriously unstable due to the fact that constraints need to be satisfied exactly. Additionally, instabilities in the traverse direction of the constraints occur when the tensile force with respect to particle masses is large when using hard constraints [TNGF15].

3.7.3 Compliant Constraints

By combining ideas from hard constraints (see Sec. 3.7.2) and penalty forces (see Sec. 3.7.1), it is possible to formulate the system matrix for hard constraints such that constraints do not have to be enforced exactly. In this approach, called compliant constraints, the constraints are combined with Newton's ODE (see Eq. 3.2) in a way that allows relaxation of constraints in a physically meaningful manner [SLM06]. The key insight is that constraints of the form $C_j(\boldsymbol{q})$ are the physical limit of strong forces from potentials of the form $\frac{k_j}{2}C_j(\boldsymbol{q})^2$ with high stiffness values k_j. It can be advantageous to express the system equations in terms of small inverse stiffnesses, also called compliances, over working with large stiffness values directly. In particular, setting the compliance to zero often provides an elegant avenue for modelling infinite stiffness.

Let $\boldsymbol{C} = [C_i, \ldots, C_r]^T$ be the vector function whose entries consist of the individual constraint functions C_j. The potential energy for $\boldsymbol{C}$ is then defined as:

$$\psi(\boldsymbol{q}) = \frac{1}{2}\boldsymbol{C}(\boldsymbol{q})^T\alpha^{-1}\boldsymbol{C}(\boldsymbol{q}) \tag{3.22}$$

where α is a positive diagonal matrix given by $\text{diag}(1/k_1, \ldots, 1/k_r)$. The resulting forces are given by

$$\boldsymbol{f}_{\boldsymbol{C}} = \nabla\psi(\boldsymbol{q}) = -\nabla\boldsymbol{C}(\boldsymbol{q})^T\alpha^{-1}\boldsymbol{C}(\boldsymbol{q}). \tag{3.23}$$

Next, artificial variables called Lagrange multipliers $\boldsymbol{\lambda} = -\alpha^{-1}\boldsymbol{C}$ are introduced, yielding

$$\boldsymbol{f}_C = \nabla \boldsymbol{C}(\boldsymbol{q})^T \boldsymbol{\lambda} \tag{3.24}$$

This leads to the following DAE:

$$\begin{aligned} \dot{\boldsymbol{q}} &= \boldsymbol{v} \\ \mathbf{M}\dot{\boldsymbol{v}} &= \boldsymbol{f}_{\text{ext}} + \nabla \boldsymbol{C}(\boldsymbol{q})^T \boldsymbol{\lambda} \\ \alpha \boldsymbol{\lambda}(\boldsymbol{q}) &= -\boldsymbol{C}(\boldsymbol{q}) \end{aligned} \tag{3.25}$$

Note, that if $\alpha = \mathbf{0}$, the formulation from hard constraints is recovered.

Usually, the DAE is solved by employing some numerical integration scheme (see Sec. 3.2) which eventually requires the solution of a linear system of equations. Here, the goal is to arrive at a formulation where both the system matrix and the right side of the resulting LSE do not contain references to the large stiffness values α^{-1}. This is achieved by treating $\boldsymbol{\lambda} = -\alpha^{-1}\boldsymbol{C}(\boldsymbol{q})$ as an unknown, pulling it out of the system matrix and hiding all occurrences of the large α^{-1} in there. To this end, it is often necessary to make simplifying assumptions suitable for the problem at hand. As an example, if the DAE is solved via backward differentiation, making the assumption that $\nabla \boldsymbol{C}$ is constant across the time step allows pulling $\boldsymbol{\lambda}$ out of the system matrix entirely [TNGF15]. The entries of the resulting system matrix and the corresponding right side are small, since they do not depend on the large stiffness terms. Backwards differentiation while assuming that $\nabla \boldsymbol{C}$ is constant across the time step yields

$$-\alpha \lambda_{n+1} = -\boldsymbol{C}(\boldsymbol{q}_{n+1}) \approx -\boldsymbol{C}(\boldsymbol{q}_n) - h \nabla \boldsymbol{C}(\boldsymbol{q}_n) \boldsymbol{v}_{n+1},$$

leading to the following LSE [TNGF15]

$$\begin{pmatrix} \mathbf{M} & -\nabla \boldsymbol{C}(\boldsymbol{q}_n)^T \\ \nabla \boldsymbol{C}(\boldsymbol{q}_n) & \frac{1}{h^2}\alpha \end{pmatrix} \begin{pmatrix} \boldsymbol{v}_{n+1} \\ \boldsymbol{\mu}_{n+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{p} + h\boldsymbol{f}_e \\ -\frac{1}{h}\boldsymbol{C}(\boldsymbol{q}_n) \end{pmatrix}, \tag{3.26}$$

where $\boldsymbol{\mu} = h\boldsymbol{\lambda}$.

This formulation comes with a couple of advantages. Firstly, relaxing the constraints by keeping a finite but large penalty parameter helps counteracting numerical problems in the presence of over defined or degenerate constraints. In comparison to the system matrix that arises for hard constraints in Equation 3.21, the lower-right zero block is replaced by the compliance matrix α in Equation 3.26 which prevents the system matrix from becoming singular in the presence of redundant constraints. Thus, the compliant constraint formulation can be interpreted as a regularization of the hard constraint formulation in Equation 3.21 [TNGF15]. Secondly, setting $\alpha = \mathbf{0}$ allows modelling infinite stiffness since α^{-1} does not occur in the system matrix or right side of Equation 3.26.

Chapter 4

Methods

4.1 Position Based Dynamics

As discussed in Section 3.1, classical approaches for dynamics simulation are force-based. Forces are accumulated and resulting accelerations are computed based on Newton's second law. These accelerations are then integrated over time via numerical integration. If successful, this strategy yields physically accurate results. However, designing integration schemes that are robust and stable, particularly in the presence of stiff forces, is challenging. Corresponding issues often manifest themselves in the context of contact and collision handling. In real-time applications, physically accurate results are often not required. Thus, algorithms that yield visually plausible simulations in a robust and stable manner are preferred. To address these needs, Müller et al. [MHHR06] propose manipulating positions directly on a per-constraint basis without integrating accelerations of velocities in an approach called Position Based Dynamics (PBD). This way, collisions can simply be handled one-by-one by projecting particles to valid locations instead of by integrating accelerations from stiff forces, leading to improved robustness and controllability.

The main drawback of PBD is that constraints become arbitrarily stiff when the iteration count is increased or when the time step is decreased. Macklin et al. [MMC16] devise an extension of PBD called extended Position Based Dynamics (XPBD) that is derived from the implicit integration of the equations of motion (see Eq. 3.2) with constraint potentials based on PBD constraints. The overall structure of the PBD algorithms is preserved with minor changes to the projection of individual constraints. XPBD reduces the coupling of stiffness to iteration count and time step and relates constraints to corresponding, well-defined constraint forces. According to Macklin et al., XPBD and PBD are equivalent in the limit of infinite stiffness.

Since PBD and XPBD only differ in the way individual constraints are projected, we give a general overview over PBD-style algorithms in Section 4.1.1. The details of individual constraint projection in PBD and XPBD are covered in Section 4.1.2 and Section 4.1.5, respectively. A comprehensive discussion of the properties of both solvers is provided in Section 4.1.3 and Section 4.1.6, respectively. Lastly, we show how to simulate deformable bodies using physically-based material models with PBD in Section 4.1.4 and with XPBD in Section 4.1.7.

4.1.1 Overview Over the PBD Framework

Both PBD and XPBD share the same algorithmic structure. Let a dynamic object be defined by a set of m vertices with inverse masses w_i, positions $\boldsymbol{q}_i$ and velocities $\boldsymbol{v}_i$. Additionally, the motion of the object is governed by $r \in \mathbb{N}$ constraints of the form

$$C\colon \mathbb{R}^{3m} \to \mathbb{R}, \boldsymbol{q} \mapsto C(\boldsymbol{q})$$

where j is the constraint index. Note how constraints are defined solely in terms of particle positions. Equality and inequality constraints are satisfied if $C(\boldsymbol{q}) = 0$ and $C(\boldsymbol{q}) \geq 0$, respectively. In PBD, each constraint has an additional stiffness parameter $k_j \in [0, 1]$. Each constraint has a cardinality $n_j \in \mathbb{N}$ and particle indices $i_1, \ldots, i_{n_j}$ of particles that actively contribute to the constraint value. In other words, for $l \in [1, \ldots, r]$ with $l \notin \{i_1, \ldots, i_{n_j}\}$ it is $\nabla_{\boldsymbol{p}_l} C_j(\boldsymbol{q}) = \boldsymbol{0}$.

An overview over the PBD framework is given in Algorithm 1 [MHHR06]. PBD and XPBD work by moving the particles according to their current velocities and the external forces acting on them and using the resulting positions as a starting point for constraint projection. This is achieved by performing symplectic Euler integration (lines 3-4). The resulting positions are projected onto the constraint manifolds of the constraints (line 5). Projecting a constraint means changing the positions of involved particles such that the constraint is satisfied and linear and angular momentum are preserved. The projected positions are used to carry out an implicit velocity update (line 7) and eventually passed on to the next time step (line 8) in correspondence with a Verlet integration step. Note that the only difference between PBD and XPBD is the constraint projection in PROJECTCONSTRAINTS (line 5).

For general, nonlinear constraints, moving the initial estimates from the symplectic Euler integration to positions that satisfy the constraints requires solving a nonlinear system of equations. Solving this system of equations is further complicated by the presence of inequality constraints, which need to be added or removed depending on whether they are satisfied during the current iteration. Thus, Müller et al. [MHHR06] opt for a nonlinear adaptation of the Gauss-Seidel solver in their

original PBD solver. Macklin et al. [MMC16] adapt this approach in XPBD. Just like the original Gauss-Seidel algorithm, which is only suitable for linear systems of equations, constraints are solved independently one after another. During each constraint solve, only the particles that contribute to the current constraint are moved while all the other particle positions remain untouched. Additionally, position updates from the projection of a constraint are immediately visible during the projection of the constraints following thereafter. Inequality constraints that are already satisfied are simply skipped. During each solver iteration, all constraints are cycled through once.

Algorithm 1 Position Based Dynamics Framework

1: **procedure** SOLVEPBD($\boldsymbol{q}_n$, $\boldsymbol{v}_n$, f_{ext}, h)
2: $\boldsymbol{q} = \boldsymbol{q}_n, \boldsymbol{v} = \boldsymbol{v}_n$
3: **for** all vertices i **do** $\boldsymbol{v}_i = \boldsymbol{v}_i + h w_i \boldsymbol{f}_{\text{ext}}(\boldsymbol{x}_i)$
4: **for** all vertices i **do** $\boldsymbol{p}_i = \boldsymbol{q}_i + h\boldsymbol{v}_i$
5: PROJECTCONSTRAINTS$(C_1, \ldots, C_r, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_m)$ (Algorithm 2 for PBD, Algorithm 3 for XPBD)
6: **for** all vertices i **do**
7: $\boldsymbol{v}_i = (\boldsymbol{p}_i - \boldsymbol{q}_i)/h$
8: $\boldsymbol{q}_i = \boldsymbol{p}_i$
9: **end for**
10: **return with** $\boldsymbol{q}_{n+1} = \boldsymbol{q}, \boldsymbol{v}_{n+1} = \boldsymbol{v}$
11: **end procedure**

Due to the fact that PBD is a geometrical method that is not derived from Newton's laws of motion (see Sec. 4.1.2) and that constraints are solved locally one after each other, Müller et al. [MHHR06] take great care that projections for internal constraints, i.e. constraints that are independent of rigid-body motion, preserve linear and angular momentum. Otherwise, internal constraints may introduce ghost forces which manifest themselves in artificial rigid-body motion [MHHR06]. Of course, non-internal constraints such as collision or attachment constraints may have global effects on an object. For internal constraints, it is easy to show that both momenta are automatically preserved if the PBD position updates are performed in the direction of the mass-weighted constraint gradient [MHHR06]. Even though XPBD – unlike PBD – is in fact derived from Newton's second law, Macklin et al. [MMC16] arrive at position updates that are multiples of the mass-weighted constraint gradient as well after a couple of simplifying assumptions (see Sec. 4.1.5). Thus PBD and XPBD projections are performed along the same direction and only differ in their scaling factors. The update formulas for a single constraint update in PBD and XPBD are derived in Section 4.1.2 and

Section 4.1.5 and the resulting algorithms for projecting all constraints acting on simulated bodies are given in Algorithm 2 and Algorithm 3, respectively.

4.1.2 PBD Constraint Projection

Mueller et al. [MHHR06] derive the projection of a single constraint in PBD as follows. Let C be a constraint of cardinality n_c acting on particles $i_1, \ldots, i_{n_c}$ with predicted positions $\boldsymbol{p}_{i_1}, \ldots, \boldsymbol{p}_{i_{n_c}}$. Let k_c be the constraint stiffness. The goal is to find a position update $\Delta\boldsymbol{p}$ such that

$$C(\boldsymbol{p} + \Delta\boldsymbol{p}) = 0. \tag{4.1}$$

To preserve linear and angular momenta, $\Delta\boldsymbol{p}$ is required to be in the direction of the mass-weighted constraint gradient, or formally

$$\Delta\boldsymbol{p} = \lambda \mathbf{W} \nabla C(\boldsymbol{p}) \tag{4.2}$$

for some $\lambda \in \mathbb{R}$ and $\mathbf{W} = \text{diag}(w_1, w_1, w_1, \ldots, w_m, w_m, w_m)$. Plugging into Equation 4.1 and approximating by first-order Taylor expansion yields

$$C(\boldsymbol{p} + \lambda \mathbf{W} \nabla C(\boldsymbol{p})) \approx C(\boldsymbol{p}) + \nabla C(\boldsymbol{p})^T \lambda \mathbf{W} \nabla C(\boldsymbol{p}) = 0.$$

Solving for λ yields

$$\lambda = -\frac{C(\boldsymbol{p})}{\sum_{i \in \{i_1, \ldots, i_{n_c}\}} w_i |\nabla_{\boldsymbol{p}_i} C(\boldsymbol{p})|^2}. \tag{4.3}$$

Plugging λ into Equation 4.2 results in the final position update

$$\Delta\boldsymbol{p} = -\frac{C(\boldsymbol{p})}{\sum_{i \in \{i_1, \ldots, i_{n_c}\}} w_i |\nabla_{\boldsymbol{p}_i} C(\boldsymbol{p})|^2} \mathbf{W} \nabla C(\boldsymbol{p}). \tag{4.4}$$

For the position of a single point $\boldsymbol{p}_i$, this gives the update

$$\Delta\boldsymbol{p}_i = -\frac{C(\boldsymbol{p})}{\sum_{i \in \{i_1, \ldots, i_{n_c}\}} w_i |\nabla_{\boldsymbol{p}_i} C(\boldsymbol{p})|^2} w_i \nabla_{\boldsymbol{p}_i} C(\boldsymbol{p}) \tag{4.5}$$

Finally, the stiffness k_c of the constraint needs to be taken into account. The simplest way is to simply multiply the projection update $\Delta\boldsymbol{p}$ by k_c. However, after multiple iterations of the solver, the effect of the stiffness on the update is nonlinear. Consider a distance constraint with rest length 0 acting on predictions $\boldsymbol{p}_{i_1}, \boldsymbol{p}_{i_2}$ given by

$$C_{\text{dist}}(\boldsymbol{p}) = |\boldsymbol{p}_{i_1} - \boldsymbol{p}_{i_2}|. \tag{4.6}$$

Then, after n_s solver iterations the remaning error is going to be $|\boldsymbol{p}_{i_1} - \boldsymbol{p}_{i_2}|(1-k)^{n_s}$. Müller et al. suggest establishing a linear relationship by multiplying corrections by $k' = 1-(1-k)^{1/n_s}$. This way, the error becomes $|\boldsymbol{p}_{i_1} - \boldsymbol{p}_{i_2}|(1-k)$ after n_s solver iterations. A summary of the constraint solver is given in Algorithm 2.

Algorithm 2 PBD Constraint Solver

1: **procedure** PROJECTCONSTRAINTS($C_1, \ldots, C_r, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_m$)
2: **for** all iterations n_s **do**
3: **for** all constraints C_j with cardinality n_j, particle indices $i_1, \ldots, i_{n_j}$ **do**
4: **if** C_j is an inequality constraint and $C_j(\boldsymbol{p}) \geq 0$ **then**
5: **continue** to next constraint
6: **end if**
7: **for** all particles $i \in \{i_1, \ldots, i_{n_j}\}$ **do**
8: $\Delta \boldsymbol{p}_i = -\frac{C_j(\boldsymbol{p})}{\sum_{i \in \{i_1,\ldots,i_{n_j}\}} w_i |\nabla_{\boldsymbol{p}_i} C_j(\boldsymbol{p})|^2} w_i \nabla_{\boldsymbol{p}_i} C_j(\boldsymbol{p})$
9: $\boldsymbol{p}_i = \boldsymbol{p}_i + k\Delta \boldsymbol{p}_i$ or $\boldsymbol{p}_i = \boldsymbol{p}_i + (1-(1-k)^{1/n_s})\Delta \boldsymbol{p}_i$
10: **end for**
11: **end for**
12: **end for**
13: **return with result** $\boldsymbol{p}$
14: **end procedure**

4.1.3 Properties of PBD

Due to its simplicity and controllability, PBD is a popular choice for real-time simulations where visually plausible results are sufficient. In the following section, we take a closer look at the properties of the PBD solver.

Gauss-Seidel Solver. At the core of the PBD algorithm is the nonlinear Gauss-Seidel-type solver for constraint projections. Immediately making position updates from one constraint projection visible in the following constraint projections enables faster propagation of constraints through the simulated body [MHHR06]. However, the same property makes parallelization of the constraint projections in lines 8-9 of Algorithm 2 more challenging. Synchronization is required to make sure that constraints that involve the same particle do not run into race conditions. Alternatively, graph coloring algorithms where constraints of the same color are guaranteed to work on separate sets of particles can be employed. Due to the fact that constraints are handled individually, the solver is incapable of finding

a compromise between contradicting constraints [MHHR06; BML+14]. In fact, oscillations can occur in over-constrainted situations. The exact result depends on the order in which constraints are handled.

Relation to the Newton-Raphson Method. The position update due to a single constraint C in Equation 4.4 is related to the Newton-Raphson method for finding roots of nonlinear functions $f\colon \mathbb{R} \to \mathbb{R}$ [MHHR06]. There, the current guess $x_i \in \mathbb{R}$ for a root of f is refined using the following update formula

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_0)}. \tag{4.7}$$

Indeed, applying the Newton-Raphson update to

$$f\colon \mathbb{R} \to \mathbb{R}, \lambda \mapsto C(\boldsymbol{p} + \lambda \mathbf{W} \nabla C(\boldsymbol{p})) \tag{4.8}$$

yields

$$\lambda_{i+1} = \lambda_i - \frac{C(\boldsymbol{p}_i + \lambda_i \mathbf{W} \nabla C(\boldsymbol{p}_i))}{\nabla C(\boldsymbol{p}_i + \lambda_i \mathbf{W} \nabla C(\boldsymbol{p}_i))^T \mathbf{W} \nabla C(\boldsymbol{p}_i)}.$$

With $\lambda_0 = 0$, the update formula for the first iteration is

$$\lambda_1 = -\frac{C(\boldsymbol{p}_0)}{\nabla C(\boldsymbol{p}_0)^T \mathbf{W} \nabla C(\boldsymbol{p}_0)}.$$

Here, subscripts denote the current iteration instead of the particle index. Note that this is exactly the same as the formula for λ used in the constraint solver given in Equation 4.3. Thus, a single constraint projection corresponds to the first iteration of the Newton-Raphson method applied to Equation 4.8 with $\lambda_0 = 0$. It is worth pointing out that the correspondence breaks down after the first iteration of PBD constraint projections. To see this, consider the second PBD iteration for the constraint C under the assumption that the positions of the particles that C acts on are not altered by any other constraint projections. Then, the second constraint projection is simply

$$\lambda_2 = -\frac{C(\boldsymbol{p_1})}{\nabla C(\boldsymbol{p}_1)^T \mathbf{W} \nabla C(\boldsymbol{p}_1)}.$$

However, since the positions after the first iteration are given by $\boldsymbol{p}_1 = \boldsymbol{p}_0 + \lambda_1 \mathbf{W} \nabla C(\boldsymbol{p_0})$, the corresponding Newton-Raphson update is equivalent to

$$\lambda_2 = \lambda_1 - \frac{C(\boldsymbol{p}_1)}{\nabla C(\boldsymbol{p}_1)^T \mathbf{W} \nabla C(\boldsymbol{p}_0)}.$$

Due to these differences, it is not clear whether convergence and stability guarantees of the Newton-Raphson solver are applicable to PBD constraint projection.

Stability. Müller et al. [MHHR06] claim that PBD is unconditionally stable since the projected positions $\boldsymbol{p}_i$ computed by the constraint solver are physically valid in the sense that all constraints are satisfied and no extrapolation into the future takes place in lines 7-8 of Algorithm 1. They further state that the only source of instabilities is the constraint solver itself, which is based on the Newton-Raphson method. The position updates in Equation 4.5 are independent of the time step and solely depend on the shape of the constraints.

At this point, it is worth taking into consideration that the constraint solver does not always succeed at moving particles to physically valid positions as implied. As mentioned above, oscillations can occur if there are contradictory constraints. In particular, constraint projections that are performed towards the end might undo progress achieved by previous projections. Moreover, we have shown above that the correspondence between the PBD constraint projections and the Newton-Raphson method breaks down after the first iteration. However, since it is known that the Newton-Raphson solver may fail to converge if the initial guess λ_0 is not sufficiently close to a true root λ^* of Equation 4.8, it is expected that the same issue can occur for repeated applications of PBD constraint projections. Lastly, for general nonlinear constraints, it is the shape of the constraint at the current configuration that matters for the stability of the position update. Whether Newton-Raphson iterations are effective or not cannot be answered for a function – or in this case for a constraint – in its entirety, but only in the proximity of specific values.

Dependence of Stiffness On Iterations and Time Step Size. The main disadvantage of PBD is the fact that the stiffness depends on the iteration count and the chosen time step [MHHR06]. Again, we take a look at a distance constraint with rest length 0 (see Eq. 4.6). As discussed in Section 4.1.2, the remaining error after n_s solver iterations is simply $|\boldsymbol{p}_{i_1} - \boldsymbol{p}_{i_2}|(1-k)^{n_s}$. In the limit of infinite iterations

$$\lim_{n_s \to \infty} (|\boldsymbol{p}_{i_1} - \boldsymbol{p}_{i_2}|(1-k)^{n_s}) = 0,$$

meaning that the distance constraint becomes infinitely stiff, regardless of the exact value of k_c. If instead $k' = 1 - (1-k)^{1/n_s}$ is used, then the error after n_s solver iterations becomes $|\boldsymbol{p}_{i_1} - \boldsymbol{p}_{i_2}|(1-k)$. Thus, infinite stiffness due to large iteration counts is prevented in this setting. However, the perceived stiffness still depends on the time step. In the limit of infinitely short time steps, the material is going to appear infinitely stiff. Generally, soft bodies can only be simulated by restricting the number of PBD iterations.

Hard Constraints. In the previous paragraph, we pointed out that PBD constraints become infinitely stiff in the limit of infinite iterations. Thus, hard con-

straints can trivially be implemented in PBD by increasing the iteration count. Note that this increases the perceived stiffness of all constraints acting on the simulated system. Thus, balancing the stiffness requirements of different constraints is a major challenge in PBD.

Preservation of Individual Momenta. While the simulated object's global linear and angular momentum are preserved, the linear momenta of individual particles are at risk of being washed out by the PBD constraint solver [BML+14]. This is because even though the structure of the position updates preserves global momentum, there is no punishment for moving individual particles away from their inertial positions. Generally, the penalty for moving particles away from their inertial positions should increase with growing particle masses. However, in the PBD position update in Equation 4.5 it is only the ratio of the particle masses that matters. This can be seen by multiplying all inverse masses w_i in Equation 4.5 with a constant factor $a \in \mathbb{R}^+$:

$$\begin{aligned}
\Delta \boldsymbol{p}_i &= -\frac{C(\boldsymbol{p})}{\sum_{j\in\{i_1,\dots,i_{n_c}\}} a w_j |\nabla_{\boldsymbol{p}_j} C(\boldsymbol{p})|^2} a w_i \nabla_{\boldsymbol{p}_i} C(\boldsymbol{p}) \\
&= -\frac{C(\boldsymbol{p})}{\sum_{j\in\{i_1,\dots,i_{n_c}\}} \frac{a w_j}{a w_i} |\nabla_{\boldsymbol{p}_j} C(\boldsymbol{p})|^2} \nabla_{\boldsymbol{p}_i} C(\boldsymbol{p}) \\
&= -\frac{C(\boldsymbol{p})}{\sum_{j\in\{i_1,\dots,i_{n_c}\}} \frac{w_j}{w_i} |\nabla_{\boldsymbol{p}_j} C(\boldsymbol{p})|^2} \nabla_{\boldsymbol{p}_i} C(\boldsymbol{p})
\end{aligned} \tag{4.9}$$

Note that the factor a gets cancelled out, meaning that increasing or decreasing the weights of all particles in the simulation by a constant factor does not affect position updates. Washing out of individual momenta also becomes evident in the limit of infinite iterations while multiplying with the stiffness k_c directly, or in the limit of infinitely short time steps. In both cases, the simulated material will appear infinitely stiff, meaning that momenta of individual particles are not necessarily preserved.

4.1.4 Simulating Deformable Bodies Using PBD

As discussed in Section 4.1.2, the PBD constraint projection is designed to solve geometric constraints. However, Bender et al. [BKCW14] simulate continuous elastic materials by deriving constraints from physical material models. Let Ψ be the energy density of the material model and let E_e, $\mathbf{F}_e$ and V_e be the elastic energy, deformation gradient and volume of a tetrahedral element e, respectively (see Sec. 3.5). Then, the authors introduce a constraint C_e with $C_e(\boldsymbol{q}) = E_e(\boldsymbol{q}) =$

$V_e\Psi(\mathbf{F}_e)$ for each tetrahedral element, with small adjustments to achieve robust inversion handling. In short, the tetrahedral energies are simply repurposed as geometric constraints. Note that this approach for simulating deformable bodies comes with the disadvantages inherent to PBD (see Sec. 4.1.3). Most notably, the perceived stiffness of the material is dependent on the time step size and the number of iterations. In the limit of infinite iterations and zero time step size, the simulated body will appear rigid. As a result, the simulations are not physically accurate, even though the constraints are derived directly from the energy density Ψ. Still, complex physical phenomena such as lateral contraction can be simulated with PBD using this setup [BKCW14].

4.1.5 XPBD Constraint Projection

The derivation of XPBD [MMC16] starts from the position-level implicit Euler update formula for the equations of motion (see Eq. 3.6), which is restated here again for the sake of convenience

$$\mathbf{M}(\boldsymbol{q}_{n+1} - \boldsymbol{q}_n - h\boldsymbol{v}_n) = h^2 \left(\boldsymbol{f}_{\text{ext}} - \sum_j \nabla\psi_j(\boldsymbol{q}_{n+1}) \right). \tag{4.10}$$

Let m be the number of particles in the simulated body and r be the number of conservative potentials ψ_j with $j \in [1, r]$. In the context of XPBD, $\boldsymbol{q}, \boldsymbol{v}, \boldsymbol{f}_{\text{ext}} \in \mathbb{R}^{3m}$ and $\psi_j \colon \mathbb{R}^{3m} \to \mathbb{R}$. Simple manipulation of Equation 4.10 yields

$$\mathbf{M}\left(\boldsymbol{q}_{n+1} - \tilde{\boldsymbol{q}}\right) = -h^2 \sum_j \nabla\psi_j(\boldsymbol{q}_{n+1}), \tag{4.11}$$

where $\tilde{\boldsymbol{q}} = \boldsymbol{q}_n + h\boldsymbol{v}_n + h^2\mathbf{M}^{-1}\boldsymbol{f}_{\text{ext}}$ is the predicted, inertial position. XPBD builds on top of the compliant constraint formulation discussed in Section 3.7.3. In the compliant constraint framework, each ψ_j can be written in terms of some positional constraint function C_j

$$\psi_j(\boldsymbol{q}) = \frac{1}{2}\alpha_j^{-1} C_j(\boldsymbol{q})^2, \tag{4.12}$$

where α_j is the inverse stiffness of the constraint. If the constraint functions are grouped into a vector-valued function $\boldsymbol{C}$ with $\boldsymbol{C}(\boldsymbol{q}) = [C_1(\boldsymbol{q}), \ldots, C_r(\boldsymbol{q})]^T$ and the inverse stiffnesses are aggregrated into the diagonal matrix $\alpha = \text{diag}(\alpha_1, \ldots, \alpha_r)$, then

$$\psi(\boldsymbol{q}) \coloneqq \sum_j \psi_j(\boldsymbol{q}) = \frac{1}{2}\boldsymbol{C}(\boldsymbol{q})^T \alpha^{-1} \boldsymbol{C}(\boldsymbol{q}). \tag{4.13}$$

where ψ is the combined internal energy potential. The force from the internal potential is given by

$$\boldsymbol{f}_{\text{int}} = -\nabla\psi(\boldsymbol{q}) = -\nabla\boldsymbol{C}(\boldsymbol{q})^T\alpha^{-1}\boldsymbol{C}(\boldsymbol{q}). \tag{4.14}$$

Plugging the internal force $\boldsymbol{f}_{\text{int}}$ into Equation 4.11 and pulling h^2 into the compliance matrix α results in

$$\mathbf{M}(\boldsymbol{q}_{n+1} - \tilde{\boldsymbol{q}}) = -\nabla C(\boldsymbol{q}_{n+1})^T\tilde{\alpha}^{-1}\boldsymbol{C}(\boldsymbol{q}_{n+1}),$$

where $\tilde{\alpha} = \frac{\alpha}{h^2}$. Now, the internal force $\boldsymbol{f}_{\text{int}}$ is split into a directional and a scalar component by introducing the Lagrange multiplier

$$\boldsymbol{\lambda} = -\tilde{\alpha}^{-1}\boldsymbol{C}(\boldsymbol{q}). \tag{4.15}$$

This leads to the following nonlinear system of equations in terms of $\boldsymbol{q}_{n+1}$ and $\boldsymbol{\lambda}_{n+1}$:

$$\mathbf{M}(\boldsymbol{q}_{n+1} - \tilde{\boldsymbol{q}}) - \nabla(\boldsymbol{q}_{n+1})^T\boldsymbol{\lambda}_{n+1} = \mathbf{0} \tag{4.16}$$

$$\boldsymbol{C}(\boldsymbol{q}_{n+1}) + \tilde{\alpha}\boldsymbol{\lambda}_{n+1} = \mathbf{0}. \tag{4.17}$$

The left-hand side of Equation 4.16 and Equation 4.17 are referred to as $\boldsymbol{g}$ and $\boldsymbol{h}$, respectively. The nonlinear system of equations is solved using a fixed-point iteration based on Newton's method. We replace the index $(n+1)$ indicating the current time step by the index of the current guess in the fixed-point iteration indicated by $(i+1)$ for the sake of clarity. During each iteration, guesses $\boldsymbol{q}_i, \boldsymbol{\lambda}_i$ for a solution of the nonlinear system are improved by updates $\Delta\boldsymbol{q}, \Delta\boldsymbol{\lambda}$ to yield new iterates $\boldsymbol{q}_{i+1}, \boldsymbol{\lambda}_{i+1}$. The updates are determined by solving the following linear system of equations, which arises from the linearization of Equation 4.16 and Equation 4.17:

$$\begin{pmatrix} \mathbf{K} & -\nabla\boldsymbol{C}^T(\boldsymbol{q}_i) \\ \nabla\boldsymbol{C}(\boldsymbol{q}_i) & \tilde{\alpha} \end{pmatrix} \begin{pmatrix} \Delta\boldsymbol{q} \\ \Delta\boldsymbol{\lambda} \end{pmatrix} = - \begin{pmatrix} \boldsymbol{g}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) \\ \boldsymbol{h}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) \end{pmatrix}, \tag{4.18}$$

Here, $\mathbf{K}$ is partial derivative of $\boldsymbol{g}$ with respect to $\boldsymbol{q}$ at $\boldsymbol{q}_i$ given by

$$\mathbf{K} = \frac{\partial\boldsymbol{g}}{\partial\boldsymbol{q}}(\boldsymbol{q}_i) = \mathbf{M} - \frac{\partial\nabla\boldsymbol{C}(\boldsymbol{q_i})^T\boldsymbol{\lambda_i}}{\partial\boldsymbol{q}}. \tag{4.19}$$

Note how the second term on the right side corresponds to the geometric stiffness $d\boldsymbol{f}(\boldsymbol{q})/d\boldsymbol{q}$ [TNGF15]. We refer to the system matrix of Equation 4.18 as $\mathbf{H}$. At this point, two simplifying assumptions are made.

Assumption 1. Computing the geometric stiffness in $\mathbf{K}$ requires evaluating second derivatives of the constraint functions C_j. This is expensive and error-prone. To avoid the challenges of computing second derivatives and to re-establish a connection to PBD (see Sec. 4.1.2), Macklin et al. [MMC16] drop the geometric stiffness by approximating

$$\mathbf{K} \approx \mathbf{M}. \tag{4.20}$$

According to the authors, this simplification does not affect the solution that the fixed-point iteration converges to. However, altering the system matrix decreases the convergence rate akin to a Quasi-Newton method for solving nonlinear systems of equations.

Assumption 2. Macklin et al. [MMC16] further assume that

$$\boldsymbol{g}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) = \mathbf{0}. \tag{4.21}$$

If initial guesses $\boldsymbol{q}_0 = \tilde{\boldsymbol{q}}$ and $\boldsymbol{\lambda}_0 = \mathbf{0}$ are used, plugging into Equation 4.16 shows that this assumption is trivially satisfied during the first iteration. To understand the justification for further iterations, it is helpful to take a look at the simplified version of Equation 4.18 with both assumptions in place:

$$\begin{pmatrix} \mathbf{M} & -\nabla \boldsymbol{C}^T(\boldsymbol{q}_i) \\ \nabla \boldsymbol{C}(\boldsymbol{q}_i) & \tilde{\alpha} \end{pmatrix} \begin{pmatrix} \Delta \boldsymbol{p} \\ \Delta \boldsymbol{\lambda} \end{pmatrix} = - \begin{pmatrix} \mathbf{0} \\ \boldsymbol{h}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) \end{pmatrix}. \tag{4.22}$$

We refer to the system matrix as $\mathbf{H}_{\text{simp}}$. After the first iteration, the upper row of Equation 4.22 is satisfied. Thus, after the first iteration with $\boldsymbol{q}_0 = \tilde{\boldsymbol{q}}$ and $\boldsymbol{\lambda}_0 = \mathbf{0}$, it is

$$\begin{aligned} \mathbf{0} = \mathbf{M}\Delta \boldsymbol{q} - \nabla \boldsymbol{C}(\boldsymbol{q}_0)^T \Delta \boldsymbol{\lambda} &= \mathbf{M}(\boldsymbol{q}_1 - \boldsymbol{q}_0) - \nabla \boldsymbol{C}(\boldsymbol{q}_0)^T (\boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_0) \\ &= \mathbf{M}(\boldsymbol{q}_1 - \tilde{\boldsymbol{q}}) - \nabla \boldsymbol{C}(\boldsymbol{q}_0)^T \boldsymbol{\lambda}_1. \end{aligned} \tag{4.23}$$

Using Equation 4.23, $\boldsymbol{g}(\boldsymbol{q}_1, \boldsymbol{\lambda}_1)$ can be rewritten as

$$\begin{aligned} &\boldsymbol{g}(\boldsymbol{q}_1, \boldsymbol{\lambda}_1) = \boldsymbol{g}(\boldsymbol{q}_1, \boldsymbol{\lambda}_1) - \mathbf{0} \\ &= \mathbf{M}(\boldsymbol{q}_1 - \tilde{\boldsymbol{q}}) - \nabla \boldsymbol{C}(\boldsymbol{q}_1)^T \boldsymbol{\lambda}_1 - \mathbf{0} \\ &= \mathbf{M}(\boldsymbol{q}_1 - \tilde{\boldsymbol{q}}) - \nabla \boldsymbol{C}(\boldsymbol{q}_1)^T \boldsymbol{\lambda}_1 - \mathbf{M}(\boldsymbol{q}_1 - \tilde{\boldsymbol{q}}) - \nabla \boldsymbol{C}(\boldsymbol{q}_0)^T \boldsymbol{\lambda}_1 \\ &= \nabla \boldsymbol{C}(\boldsymbol{q}_0)^T \boldsymbol{\lambda}_1 - \nabla \boldsymbol{C}(\boldsymbol{q}_1)^T \boldsymbol{\lambda}_1 \\ &= (\nabla \boldsymbol{C}(\boldsymbol{q}_0)^T - \nabla \boldsymbol{C}(\boldsymbol{q}_1)^T) \boldsymbol{\lambda}_1 \end{aligned} \tag{4.24}$$

Note that if $\nabla \boldsymbol{C}(\boldsymbol{q}_0)^T = \nabla \boldsymbol{C}(\boldsymbol{q}_1)^T$, then $\boldsymbol{g}(\boldsymbol{q}_1, \boldsymbol{\lambda}_1) = \mathbf{0}$. Thus, Macklin et al. [MMC16] argue that $\boldsymbol{g}(\boldsymbol{q}_1, \boldsymbol{\lambda}_1) \approx \mathbf{0}$, as long as $\nabla \boldsymbol{C}(\boldsymbol{q})$ does not change too quickly.

Since the mass matrix $\mathbf{M}$ in the upper-left block of $\mathbf{H}_{\text{simp}}$ is invertible by design, it is possible to take the Schur complement with respect to $\mathbf{M}$ to obtain a reduced system in terms of $\Delta\boldsymbol{\lambda}$:

$$(\nabla \boldsymbol{C}(\boldsymbol{q}_i)\mathbf{M}^{-1}\nabla \boldsymbol{C}(\boldsymbol{q}_i)^T + \tilde{\alpha})\Delta\boldsymbol{\lambda} = -\boldsymbol{C}(\boldsymbol{q}_i) - \tilde{\alpha}\boldsymbol{\lambda}_i \tag{4.25}$$

The position update $\Delta\boldsymbol{q}$ can be derived from $\Delta\boldsymbol{\lambda}$ via the formula

$$\Delta\boldsymbol{q} = \mathbf{W}\nabla \boldsymbol{C}(\boldsymbol{q}_i)^T\Delta\boldsymbol{\lambda}, \tag{4.26}$$

where $\mathbf{W} = \mathbf{M}^{-1}$

Up until here, all constraints were handled together during each iteration. To make a connection to PBD and to return to the framework of a nonlinear Gauss-Seidel solver Section 4.1.1, it is necessary to specify how to solve a single constraint. To that end we rewrite Equation 4.25 for a single constraint C_j and get the update for its scalar Lagrange multiplier λ_j by computing

$$\Delta\lambda_j = \frac{-C_j(\boldsymbol{q}_i) - \tilde{\alpha}_j\lambda_{ji}}{\nabla C_j(\boldsymbol{q}_i)\mathbf{W}\nabla C_j(\boldsymbol{q}_i)^T + \tilde{\alpha}_j}. \tag{4.27}$$

Here, λ_{ji} is the value of the Lagrange multiplier of the j-th constraint after the i-th solver iteration. The position update for a single particle with index l contributing to C_j becomes

$$\Delta\boldsymbol{q}_l = w_l\nabla_{\boldsymbol{q}_l}C_j(\boldsymbol{q}_i)^T\Delta\lambda_j. \tag{4.28}$$

Note that $\Delta\lambda_j$ is scalar. Thus, the position update is a multiple of the mass-weighted gradient, just like in PBD (see Eq. 4.5).

In summary, we simply compute $\Delta\lambda_j$ via Equation 4.27 and use it to update $\lambda_{j+1} = \lambda_j + \Delta\lambda$ and to determine $\Delta\boldsymbol{q}$ via Equation 4.26 while solving the j-th constraint. This leads to a natural extension of the PBD algorithm, where the general structure in Algorithm 1 is preserved. The only changes occur in the computation of the scaling factor for the mass-weighted constraint gradient in PROJECTCONSTRAINTS in line 5. The XPBD version of PROJECTCONSTRAINTS is given in Algorithm 3. Note that Algorithm 3 is specified in terms of the projection points $\boldsymbol{p}$ or $\boldsymbol{p}_i$ instead of the positions $\boldsymbol{q}$ or $\boldsymbol{q}_i$ used in Equation 4.26 and Equation 4.27 to maintain notational consistency with the PBD solver in Algorithm 2.

Algorithm 3 XPBD Constraint Solver

procedure PROJECTCONSTRAINTS($C_1, \dots, C_r, \boldsymbol{p}_1, \dots, \boldsymbol{p}_m$)
  for all constraints C_j **do** $\lambda_j = 0$
  for all iterations n_s **do**
    for all constraints C_j with cardinality n_j, particle indices $i_1, \dots, i_{n_j}$, Lagrange multiplier λ_j **do**
      if C_j is an inequality constraint and $C_j(\boldsymbol{p}) \geq 0$ **then**
        continue to next constraint
      end if
      $\Delta\lambda_j = \frac{-C_j(\boldsymbol{p}) - \tilde{\alpha}_j \lambda_j}{\nabla C_j(\boldsymbol{p}) \mathbf{W} \nabla C_j(\boldsymbol{p})^T + \tilde{\alpha}_j}$
      $\lambda_j = \lambda_j + \Delta\lambda_j$
      for all particles $i \in \{i_1, \dots, i_{n_j}\}$ **do**
        $\Delta\boldsymbol{p}_i = \Delta\lambda_j w_i \nabla_{\boldsymbol{p}_i} C_j(\boldsymbol{p})$
        $\boldsymbol{p}_i = \boldsymbol{p}_i + \Delta\boldsymbol{p}_i$
      end for
    end for
  end for
  return with result $\boldsymbol{p}$
end procedure

4.1.6 Properties of XPBD

XPBD is a natural extension of PBD that addresses some of PBD's shortcoming while maintaining the simplicity of the original algorithm. Due to the similarity of both algorithms, PBD implementations can readily be extended to XPBD at the minor cost of storing an additional variable per constraint. In the following, the properties of XPBD are discussed.

Relation between XPBD and PBD. The derivation of the XPBD constraint projection builds on the concept of compliant constraints developed by Servin et al. [SLM06]. As discussed in Section 3.7.3, the compliant constraint formulation often allows handling hard constraints by setting $\alpha = \mathbf{0}$. Indeed, a closer look at the PBD update formula in Equation 4.5 and the XPBD update formulas in Equation 4.27, Equation 4.28 reveals that XPBD and PBD are equivalent if the compliance term α_j of constraint C_j is zero. This matches with the observation that bodies simulated by PBD are infinitely stiff in the limit of infinite iterations (see Sec. 4.1.3). Note that this correspondence between XPBD and PBD gets lost if the stiffnesses are not moved out of the constraints and moved into the compliances. To see this, consider the energy potential $\psi(\boldsymbol{q}) = \frac{1}{2}\alpha^{-1}C(\boldsymbol{q})^2$. If we

construct alternative constraints C' with $C'(\boldsymbol{q}) = \alpha^{-1/2}C(\boldsymbol{q})$, we can alternatively express the energy potential ψ as follows

$$\psi(\boldsymbol{q}) = \frac{1}{2}C'(\boldsymbol{q})^2 = \frac{1}{2}\alpha'^{-1}C'(\boldsymbol{q})^2,$$

with compliance $\alpha' = 1$ independent of material stiffness. In this setting, modelling infinite stiffness by setting $\alpha' = 0$ does not work.

If $\alpha_j \neq 0$, the compliance terms in Equation 4.27 regularize the constraint in such a way that the constraint force is limited and corresponds to the constraint potential [MMC16]. This addresses the issue of coupling between iteration count and stiffness in the original PBD algorithm (see Sec. 4.1.3). Since the time step is later baked into $\tilde{\alpha}_j$, coupling between time step size and stiffness is also removed.

Hard Constraints. As discussed above, the XPBD update equations are equivalent to the PBD update equations for constraints with zero compliance. Since PBD constraints appear infinitely stiff in the limit of infinite iterations, this provides an avenue for modelling hard constraints with XPBD. In contrast to PBD, increasing the iteration count does not cause the perceived stiffness of constraints with non-zero compliance to increase as a result of the compliant constraint formulation. Thus, hard constraints and soft constraints can be handled together effortlessly with XPBD.

Generality. The XPBD derivation builds on the assumption that energy potentials ψ fit into the compliant constraint framework (see Eq. 4.12). Thus, implementing an energy potential ψ in XPBD requires constructing a constraint C such that $\psi(\boldsymbol{q}) = \frac{1}{2}\alpha^{-1}C(\boldsymbol{q})^2$. Since $\alpha \geq 0$ and $C(\boldsymbol{q})^2 \geq 0$ for all $\boldsymbol{q}$ irrespective of C, this can only be achieved for non-negative energy potentials ψ. Thus, XPBD is not compatible with energy potentials that can take on negative values without further modifications. However, we can take advantage of the fact that adding a constant offset $c \in \mathbb{R}$ to ψ does not affect the resulting forces $\boldsymbol{f} = d\psi/d\boldsymbol{q}$ to construct an equivalent energy potential ψ' if ψ is bounded from below. Let $b \in \mathbb{R}$ be a lower bound of ψ. Then, an equivalent non-negative energy potential ψ' can be constructed by setting

$$\psi'(\boldsymbol{q}) = \psi(\boldsymbol{q}) + |b|. \tag{4.29}$$

Note that this strategy does not work if ψ does not have a lower bound. For example, the Neohookean energy density from Equation 3.14 is unbounded from below due to the volume-preserving log terms. Consequently, it is imcompatible with the XPBD solver.

Simplifying Assumptions. As mentioned in Section 4.1.5, assumption 1 (see Eq. 4.20) corresponds to dropping the geometric stiffness $\delta \nabla \boldsymbol{C}(\boldsymbol{q_i})^T \boldsymbol{\lambda_i} / \delta \boldsymbol{q}$. It is worth noting that with $\boldsymbol{\lambda}_0 = \mathbf{0}$, this assumption is trivially satisfied during the first iteration. According to Equation 4.15, after a sufficient number of iterations it is $\boldsymbol{\lambda}_i \approx \tilde{\alpha} \boldsymbol{C}(\boldsymbol{q}_i)$, meaning that the entries of the geometric stiffness grow with increasing constraint stiffness and iteration count. Thus, with stiffer constraints and larger iteration counts, assumption 1 becomes more aggressive. Tournier et al. [TNGF15] state that the iterative nature of PBD-style solvers ameliorates instabilities in the transverse direction of stiff constraints that is often observed as a result of neglecting geometric stiffness.

Macklin et al. [MMC16] claim that replacing $\mathbf{H}$ with $\mathbf{H}_{\text{simp}}$ in assumption 1 can be interpreted as applying a quasi-Newton method – also known as Broyden methods – to the NLSE given by Equation 4.16 and Equation 4.17, only affecting the convergence rate. While it is true that changing the Hessian matrix in Newton's method for unconstrained optimization to some positive definite approximation only changes the convergence rate under mild conditions [NW06], it is difficult to verify whether this also holds for the simplification from $\mathbf{H}$ to $\mathbf{H}_{\text{simp}}$ in the context of solving NLSEs. There, it is often the case that quasi-Newton methods are not guaranteed to converge at all if aggressive approximations of the system matrix are performed [NW06]. This is because there is no natural merit function available to help with the selection of step sizes along the suggested update directions. In their XPBD derivation, Macklin et al. [MMC16] do point out that a line search strategy might be required to keep the fixed-point iteration based on Equation 4.18 robust, but it is also important to mention that approximations of the system matrix without an appropriate line search procedure in place are potentially more aggressive.

Assumption 2 in the XPBD derivation is justified by the observation that Equation 4.23 and $\boldsymbol{g}(\boldsymbol{q}_1, \boldsymbol{\lambda}_1)$ are the same, except that $\nabla \boldsymbol{C}(\boldsymbol{q}_0)^T$ is replaced by $\nabla \boldsymbol{C}(\boldsymbol{q}_1)^T$. Thus, Macklin et al. [MMC16] claim that Equation 4.23 is close to zero as well if $\boldsymbol{g}(\boldsymbol{q}_0, \boldsymbol{\lambda}_0) = \mathbf{0}$, which is true by definition of $\boldsymbol{q}_0$ and $\boldsymbol{\lambda}_0$. However, this neglects the fact that $\nabla \boldsymbol{C}(\boldsymbol{q}_i)$ occurs in a product with $\boldsymbol{\lambda}_i$ in $\boldsymbol{g}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i)$. As mentioned in the discussion of assumption 1, $\boldsymbol{\lambda}_i$ gets very large for stiff constraints after a sufficient number of iterations. Thus, even small changes to $\nabla \boldsymbol{C}(\boldsymbol{q}_i)$ eventually drive the value of $\boldsymbol{g}$ away from zero significantly. As a result, assumption 2 becomes more aggressive with stiffer constraints and larger iteration counts, just like observed for assumption 1. Additionally, assuming that $\boldsymbol{g}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) = \mathbf{0}$ leads to a change of the right side between the LSEs in Equation 4.18 and Equation 4.22. In contrast to the changes to the system matrix during assumption 1, this does affect the solution the solver converges to. Thus, due to assumption 2, the XPBD solver cannot be said to solve the original implicit equations of motion [MMC16].

Preservation of Individual Momenta. When investigating the properties of PBD (see Sec. 4.1.3), we mentioned that multiplying all particle masses m_i with a constant positive factor $a \in \mathbb{R}^+$ does not impact the PBD update, highlighting that there is no punishment for moving individual particles from their inertial positions in PBD. The fact that both positions and Lagrange multipliers are updated during each iteration of the XPBD solver makes an analysis similar to Equation 4.9 for XPBD challenging for arbitrary iterations i. However, it is simple to look at the effect of scaling all particle masses by a on the position update during the first iteration, assuming that $\boldsymbol{\lambda} = \mathbf{0}$. The resulting update of particle i is given by

$$\Delta \boldsymbol{q}_i = -\frac{C(\boldsymbol{q})}{\sum_{j \in \{i_1,\dots,i_{n_c}\}} \frac{w_j}{w_i} |\nabla_{\boldsymbol{q}_j} C(\boldsymbol{q})|^2 + \tilde{\alpha} a m_i} \nabla_{\boldsymbol{q}_i} C(\boldsymbol{q}). \tag{4.30}$$

Note how the factor a does occur in the second summand in the denominator in a way that decreases the size of the position update if a is increased. However, it appears in a product with the small $\tilde{\alpha}$. Thus, the effect of increasing all particle masses on the position update is still rather small. It is worth pointing out that the inclusion of a in the position update of the first iteration penalizes moving particles with large masses in all directions. Ideally, moving particles towards the inertial positions $\tilde{\boldsymbol{q}}$ should be encouraged whereas as moving particles away from $\tilde{\boldsymbol{q}}$ should be discouraged. This directionality is only possible if $\tilde{\boldsymbol{q}}$ is not simply used as an initial guess for the XPBD solver, but also used explicitly in the update equations. However, setting $\boldsymbol{g}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) = \mathbf{0}$ in assumption 2 removes all occurrences of the inertial positions from Equation 4.22.

Initial Guess For $\boldsymbol{\lambda}_0$. Both assumptions 1 and 2 exploit the choice $\boldsymbol{\lambda}_0 = \mathbf{0}$. It is worth mentioning that this precludes the use of the more natural initial guess given by $\boldsymbol{\lambda}_0 = -\tilde{\alpha}\boldsymbol{C}(\tilde{\boldsymbol{q}})$. This candidate results from plugging the inertial positions into the definition of the Lagrange multipliers in Equation 4.15. Picking an initial guess that is as close as possible to the true solution is important for keeping the linearization error during the fixed-point iteration based on Newton's method in the XPBD derivation small.

Numerical Damping. While motion due to external forces is handled by the symplectic Euler integration in lines 3-4 of Algorithm 1, the way elasticity is handled is derived from the implicit equations of motions (see Eq. 4.10). As a result elastic forces are subjected to numerical damping that gets more severe with growing time step sizes in XPBD (see Sec. 3.2). For this reason, even though baking the time step into $\tilde{\alpha}$ in the compliant constraint formulation reduces coupling between time step and stiffness, the perceived stiffness of simulated materials is still time step dependent to some extent.

Gauss-Seidel Solver. Finally, it is worth pointing out that the implications of using a Gauss-Seidel-type solver discussed in the context of PBD (see Sec. 4.1.3) apply to XPBD as well.

4.1.7 Simulating Deformable Bodies Using XPBD

To simulate deformable bodies using the material models presented in Section 3.6, we introduce constraints C_j with compliance α_j for each tetrahedral element such that the corresponding constraint energy potential ψ_j satisfies

$$\psi_j(\boldsymbol{q}) = \frac{1}{2}\alpha_j^{-1} C_j(\boldsymbol{q})^2 = V_j \Psi_{\text{mat}}(\mathbf{F}_j). \tag{4.31}$$

Here, V_j is the volume of the undeformed tetrahedron, Ψ_{mat} is the energy density of the material model and $\mathbf{F}_j$ is the deformation gradient of the deformed tetrahedron. As discussed in Section 4.1.6, this cannot be achieved for the classical Neohookean material model in Equation 3.14. In the following, we show how to implement the strain material model (see Eq. 3.13) and the simplified Neohookean material model (see Eq. 3.15) in XPBD.

Strain Material Model. For the strain material model Ψ_{strain} in Equation 3.13, Equation 4.31 is satisfied by setting

$$\alpha_j = \frac{1}{kV_j} \text{ and } C_j(\boldsymbol{q}) = \left\| \mathbf{I} - \boldsymbol{\Sigma}_j \right\|_F,$$

where k is the stiffness of the strain material model and $\boldsymbol{\Sigma}_j$ is the diagonal matrix whose entries are the singular values of $\mathbf{F}_j$. Note that k only appears in the compliance α_j, but not in of the constraint C_j. Thus, the behavior of PBD can be recovered by setting $\alpha_j = 0$.

Simplified Neohookean Material Model. For the simplified Neohookean material model in Equation 3.15, constructing suitable constraints C_j and compliances α_j is more complicated. This is due to the fact that Ψ_{Smith} can take on negative values. First, we observe that Ψ_{Smith} is bounded from below by rearranging terms as follows

$$\begin{aligned}
\Psi_{\text{Smith}}(\mathbf{F}) &= \frac{\mu}{2}(\text{tr}(\mathbf{F}^T\mathbf{F}) - 3) + \frac{\lambda}{2}\left(\det(\mathbf{F}) - \left(1 + \frac{\mu}{\lambda}\right)\right)^2 \\
&= \frac{\mu}{2}\text{tr}(\mathbf{F}^T\mathbf{F}) + \frac{\lambda}{2}\left(\det(\mathbf{F}) - \left(1 + \frac{\mu}{\lambda}\right)\right)^2 - \frac{3\mu}{2}.
\end{aligned}$$

Note that the first two terms are non-negative, while the last term is negative and independent of $\boldsymbol{q}$. Consequently, $-\frac{3\mu}{2}$ is a lower bound of Ψ_{Smith} and the equivalent non-negative energy density Ψ'_{Smith} given by

$$\Psi'_{\text{Smith}}(F) = \frac{\mu}{2}\text{tr}(\mathbf{F}^T\mathbf{F}) + \frac{\lambda}{2}\left(\det(\mathbf{F}) - \left(1 + \frac{\mu}{\lambda}\right)\right)^2$$

can be constructed (see Sec. 4.1.6). We refer to the first and second term of Ψ'_{Smith} as the deviatoric and hydrostatic term, respectively.

To prepare pulling out the large Lamé coefficients μ and λ that control the stiffness and incompressibility of the simulated material, we factor out the larger coefficient $x \coloneqq \max(\mu, \lambda)$ as follows

$$\Psi'_{\text{Smith}}(F) = \frac{x}{2}\left(\frac{\mu}{x}\text{tr}(\mathbf{F}^T\mathbf{F}) + \frac{\lambda}{x}\left(\det(\mathbf{F}) - \left(1 + \frac{\mu}{\lambda}\right)\right)^2\right).$$

Now, Equation 4.31 is satisfied for $\Psi_{\text{mat}} = \Psi'_{\text{Smith}}$ if we set

$$\alpha_j = \frac{1}{xV_j} \text{ and } C_j(\boldsymbol{q}) = \sqrt{\frac{\mu}{x}\text{tr}(\mathbf{F}_j^T\mathbf{F}_j) + \frac{\lambda}{x}\left(\det(\mathbf{F}_j) - \left(1 + \frac{\mu}{\lambda}\right)\right)^2}.$$

We refer to this type of constraint as the combined Neohookean constraint. Note that x does not only appear in the compliance α_j, but in the constraint C_j as well. Thus, this formulation can only be applied directly if $\alpha_j \neq 0$. Otherwise, we need to adjust C_j in order to maintain consistency by either computing $C_j^\mu \coloneqq \lim_{\mu\to\infty} C_j$ or $C_j^\lambda \coloneqq \lim_{\lambda\to\infty} C_j$. If $x = \mu$, then C_j^μ is undefined since $\lim_{\mu\to\infty}$ diverges. However, if $x = \lambda$, it is easy to see that C_j^λ must be

$$C_j^\lambda(\boldsymbol{q}) = \lim_{\lambda\to\infty} C_j(\boldsymbol{q}) = \sqrt{(\det(\mathbf{F}_j) - 1)^2} = \det(\mathbf{F}_j) - 1. \tag{4.32}$$

Note that C_j^λ only enforces that the volume of its tetrahedron is preserved. Still, it does not prevent stretching the tetrahdron in one direction if compressing it in another direction balances out the change in volume.

Macklin et al. [MM21] suggest a different way of implementing the simplified Neohookean material model with XPBD. The trick is to introduce not one, but two constraints for each tetrahedral element. The authors construct separate constraints C_j^D with compliance α_j^D and C_j^H with compliance α_j^H for the deviatoric and hydrostatic terms of Ψ'_{Smith}, respectively, given by

$$C_j^D(\boldsymbol{q}) = \sqrt{\text{tr}(\mathbf{F}_j^T\mathbf{F}_j)}, \quad \alpha_j^D = \frac{1}{\mu V_j} \tag{4.33}$$

and

$$C_j^H(\boldsymbol{q}) = \det(\mathbf{F}_j) - \left(1 + \frac{\mu}{\lambda}\right), \quad \alpha_j^H = \frac{1}{\lambda V_j}. \tag{4.34}$$

It is easily verified that

$$\psi_j(\boldsymbol{q}) = \frac{1}{2}(\alpha_j^D)^{-1} C_j^D(\boldsymbol{q})^2 + \frac{1}{2}(\alpha_j^H)^{-1} C_j^H(\boldsymbol{q})^2 = V_j \Psi'_{\text{Smith}}(\mathbf{F}_j).$$

The advantage of splitting constraints becomes apparent when modelling hard incompressibility by setting $\alpha_j^H = 0$. Since λ appears in both $\alpha_j^H = 0$ and C_j^H, it is necessary to adjust C_j^H by computing $\lim_{\lambda\to\infty} C_j^H$ again. It is

$$\lim_{\lambda\to\infty} C_j^H(\boldsymbol{q}) = \det(\mathbf{F}_j) - 1,$$

which is the same as what was observed for combined Neohookean constraints in Equation 4.32. However, this time, there is an independent deviatoric constraint C_j^D that additionally drives the tetrahedron towards configurations where the principal stretches of the deformation gradient $\mathbf{F}_j$ are not unevenly distributed. Note that setting $\alpha_j^D = 0$ is impossible again, since $\lim_{\mu\to\infty} C_j^H$ diverges.

On the other hand, constructing separate constraints for the deviatoric and hydrostatic terms also introduces problems. In Section 3.6.3, we discussed that the hydrostatic term needs to be corrected to ensure the rest stability of the simplified Neohookean material model. The correction is chosen such that tetrahedral elements are artificially inflated by exactly the right amount to make up for the deflation caused by the deviatoric term. The derivation of the correction term relies on the assumption that both the deviatoric and the hydrostatic term act on the same positions simultaneously [SGK18]. However, due to the nature of the Gauss-Seidel-type solver used in XPBD, the constraints C_j^D and C_j^H change the tetrahedral positions after each other. Thus, elements are inflated and deflated sequentially during each solver iteration. This way, rest stability cannot be ensured. The issue is particularly severe for simulations with large Lamé coefficient μ, since this increases the degree to which both C_j^D and C_j^H deflate and inflate the elements, respectively. Similarly, we expect larger time steps to make matters worse.

4.2 Projective Dynamics

In the approaches to physical simulations via implicit time integration that we have encountered so far, a new linear system needs to be solved at every time step. If the linear system is solved directly, this can quickly become prohibitively

expensive for large simulations since a new matrix factorization needs to be computed every time a new sytem needs to be solved. In XPBD, this issue is dealt with by using an iterative solver. In Projective Dynamics (PD), Bouaziz et al. [BML+14] instead restrict energy potentials to a specific structure which allows for efficient implicit time integration via alternating steps of local and global optimization [BML+14]. The local optimization steps are comprised of per-constraint projections of particle positions onto constraint manifolds. The global optimization step combines the results from the individual local projection steps while taking into consideration global effects including inertia and external forces. This is achieved by solving a linear system of equations whose system matrix is constant across time steps. Since the local steps can be carried out in parallel and the factorization for the system matrix of the global step can be precomputed and reused, physical simulations that are restricted to energy potentials from the PD framework can be solved efficiently. In Section 4.2.1, an overview over the PD solver is provided. There, we focus on introducing the broad structure of PD energy potentials and the way it benefits the convergence properties of the solver. In Section 4.2.2, an exact formulation of PD energy potentials is provided. This formulation is used in Section 4.2.3 to fill in the gaps in the overview given in Section 4.2.1 and to give a complete description of the PD solver. The properties of the PD solver are discussed in Section 4.2.4. We show how to simulate deformable bodies using the strain material model (see Eq. 3.13) in Section 4.2.5. Lastly, we make a connection between the PD solver and Quasi-Newton methods for unconstrained optimization in Section 4.2.6.

4.2.1 Overview Over PD

Projective Dynamics starts from the variational form of implicit Euler integration of the equations of motion (see Eq. 3.7), which is restated here again for the sake of convenience

$$\min_{\mathbf{q}_{n+1}} \frac{1}{2h^2} \left\| \mathbf{M}^{\frac{1}{2}} (\mathbf{q}_{n+1} - \tilde{\mathbf{q}}) \right\|_F^2 + \sum_j \psi_j(\mathbf{q}_{n+1}). \tag{4.35}$$

Recall that in the context of PD, the particle positions are stored in matrices, i.e. $\mathbf{q}_{n+1}, \tilde{\mathbf{q}} \in \mathbb{R}^{m\times 3}$ and $\mathbf{M} \in \mathbb{R}^{m\times m}$, where $m \in \mathbb{N}$ is the number of particles in the simulated body. In this section, we simply write $\mathbf{q}$ instead of $\mathbf{q}_{n+1}$ to improve legibility. The solution of this unconstrained optimization problem corresponds to the particle positions of the simulated body at the next time step (see Sec. 3.2) and can be determined using general purpose algorithms for unconstrained optimization such as Newton's method. Naive application of line search methods can often lead to unfeasibly high runtimes for real-time applications. The core

idea of PD is to speed up the optimization by restricting the energy potentials ψ_j to a structure that allows for efficient minimization. This allows substituting the line search methods – which are designed for the minimization of general nonlinear objective functions – with a specialized solver that exploits the structure of PD energy potentials.

Consider the restriction of energy potentials ψ_j to quadratic functions of $\mathbf{q}$. Then, as the sum of the inertial term (always quadratic) and the constraint energy (quadratic by assumption) the entire objective function of Equation 4.35 is quadratic in $\mathbf{q}$. Thus, the solution to the minimization problem can be found in a single linear solve [NW06]. While this choice of energy potentials does enable efficient implicit Euler integration of the equations of motion, it comes at the sacrifice of generality. In particular, quadratic energy potentials always yield linear forces. Linear forces generally do not suffice to model realistic elastic behavior [WRO11]. Now, the challenge lies in enhancing quadratic energy potentials to allow expressing some degree of nonlinearity while still keeping the minimization efficient.

In PD, Bouaziz et al. [BML+14] achieve this by introducing energy potentials that roughly measure the square of the distance of the current particle positions to their projections onto some – potentially nonlinear – constraint manifold $\boldsymbol{C}_j$. In this setting, the energy potential ψ_j is defined entirely by the squared distance function d_j and the corresponding constraint manifold $\boldsymbol{C}_j$. By picking an appropriate distance measure, the squared distance d_j can be made quadratic in $\mathbf{q}$ (see Sec. 4.2.2). Since the constraint manifold $\boldsymbol{C}_j$ is potentially nonlinear, the energy potential ψ_j can produce nonlinear forces as well.

Assume that the constraint has cardinality $n_j \in \mathbb{N}$ and acts on the particles with indices $i_1, \ldots, i_{n_j}$ and let $\mathbf{q}_j \in \mathbb{R}^{n_j \times 3}$ be

$$\mathbf{q}_j = \begin{pmatrix} \mathbf{q}_{i_1}^T \\ \cdots \\ \mathbf{q}_{i_{n_j}}^T \end{pmatrix}.$$

Then, Bouaziz et al. [BML+14] introduce auxiliary variables $\mathbf{p}_j \in \mathbb{R}^{n_j \times 3}$ per constraint and define the energy potential in terms of the function W_j given by

$$W_j(\mathbf{q}_j, \mathbf{p}_j) = d_j(\mathbf{q}_j, \mathbf{p}_j) + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j). \tag{4.36}$$

Here, $\delta_{\boldsymbol{C}_j}$ is an indicator function with

$$\delta_{\boldsymbol{C}_j}(\mathbf{p}_j) = \begin{cases} 0, & \text{if } \mathbf{p}_j \text{ lies on the constraint manifold } \boldsymbol{C}_j \\ \infty, & \text{otherwise.} \end{cases} \tag{4.37}$$

Consider $\mathbf{p}_{\mathbf{q}_j} \coloneqq \underset{\mathbf{p}_j}{\operatorname{argmin}}\ W(\mathbf{q}_j, \mathbf{p}_j)$ while keeping $\mathbf{q}_j$ fixed. Then obviously $\delta_{\boldsymbol{C}_j}(\mathbf{p}_{\mathbf{q_j}}) = 0$, meaning that $\mathbf{p}_{\mathbf{q_j}}$ lies on $\boldsymbol{C}_j$. Additionally, for all other $\mathbf{p}^* \in \mathbb{R}^{n_j \times 3}$ with $\delta_{\boldsymbol{C}_j}(\mathbf{p}^*) = 0$ it is $d_j(\mathbf{q}_j, \mathbf{p}_{\mathbf{q}_j}) \leq d_j(\mathbf{q}_j, \mathbf{p}^*)$. Together, $\mathbf{p}_{\mathbf{q}_j}$ is the configuration on the constraint manifold that is the closest to the configuration specified by positions $\mathbf{q}_j$ in terms of d_j. Thus, $\mathbf{p}_{\mathbf{q}_j}$ can be interpreted as the projection of $\mathbf{q}_j$ onto the constraint manifold $\boldsymbol{C}_j$. Note that d_j itself is not a distance function, but $\sqrt{d_j}$ is (see Sec. 4.2.2). Still, we follow the example of Bouaziz et al. [BML+14] and define closeness between pairs of configurations in terms of d_j instead of $\sqrt{d_j}$. This is justified since for two pairs of configurations $\mathbf{q}, \mathbf{p}$ and $\mathbf{q}', \mathbf{p}'$ it is

$$d_j(\mathbf{q}, \mathbf{p}) < d_j(\mathbf{q}', \mathbf{p}') \iff \sqrt{d_j}(\mathbf{q}, \mathbf{p}) < \sqrt{d_j}(\mathbf{q}', \mathbf{p}').$$

Bouaziz et al. [BML+14] define the energy potential ψ_j as

$$\psi_j(\mathbf{q}) = \min_{\mathbf{p}_j} W(\mathbf{S}_j\mathbf{q}, \mathbf{p}_j) = \min_{\mathbf{p}_j} d_j(\mathbf{q}_j, \mathbf{p}_j) + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j) = d_j(\mathbf{q}_j, \mathbf{p}_{\mathbf{q}_j}), \quad (4.38)$$

where $\mathbf{S}_j$ is the matrix that maps $\mathbf{q}$ to $\mathbf{q}_j$. This is exactly the squared distance as measured by d_j between configuration $\mathbf{q}_j$ and its projection $\mathbf{p}_{\mathbf{q}_j}$ onto the constraint manifold $\boldsymbol{C}_j$. Plugging the energy potentials into the variational form of implicit Euler integration (see Eq. 4.35) yields

$$\min_{\mathbf{q}, \mathbf{p}_j} \frac{1}{2h^2} \left\| \mathbf{M}^{\frac{1}{2}}(\mathbf{q} - \tilde{\mathbf{q}}) \right\|_F^2 + \sum_j d_j(\mathbf{S}_j\mathbf{q}, \mathbf{p}_j) + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j). \quad (4.39)$$

Here, with abuse of notation, $\boldsymbol{p}_j$ denotes either the auxiliary variable of the j-th constraint or the family of auxiliary variables $(\boldsymbol{p}_j)_{j \in \mathcal{J}}$, where $\mathcal{J}$ is the index set of the constraints. Note that if the projections $\mathbf{p}_{\mathbf{q}_j}$ are known in advance and kept fixed for all constraint manifolds C_j, then the minimization problem becomes

$$\min_{\mathbf{q}} \frac{1}{2h^2} \left\| \mathbf{M}^{\frac{1}{2}}(\mathbf{q} - \tilde{\mathbf{q}}) \right\|_F^2 + \sum_j d_j(\mathbf{S}_j\mathbf{q}, \mathbf{p}_{\mathbf{q}_j}). \quad (4.40)$$

In this case, the objective function is a quadratic function of $\mathbf{q}$ again. As a result, it can be optimized in a single linear step.

These insights suggest a local/global structure for the PD solver. In the local step, the projection points $\mathbf{p}_{\mathbf{q}_j}$ are computed for each constraint. Finding the projection points $\mathbf{p}_{\mathbf{q}_j}$ corresponds to minimizing Equation 4.39 over the projection variables $\mathbf{p}_j$ while keeping the positions $\mathbf{q}$ fixed, yielding the following optimization problem

$$\min_{\mathbf{p}_j} \sum_j d_j(\mathbf{S}_j\mathbf{q}, \mathbf{p}_j) + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j). \quad (4.41)$$

Since each constraint has its own auxiliary projection variables, this minimization can be carried out independently for each constraint. In the global step, Equation 4.40 is minimized, which is equivalent to minimizing Equation 4.39 over the positions $\mathbf{q}$ while keeping the projection variables $\mathbf{p}_j$ fixed. Local and global steps are repeated for a fixed number of iterations during each time step. Finally, the resulting positions are used as the positions of the next time step.

4.2.2 PD Energy Potentials

To arrive at a working implementation of the PD solver outlined in Section 4.2.1, a squared distance function d_j and a projection operator onto some constraint manifold $\boldsymbol{C}_j$ need to be provided for each energy potential ψ_j. While the projection operators vary significantly between different types of constraints, Bouaziz et al. [BML+14] always use squared distance functions d_j of the form

$$d_j(\mathbf{q}_j, \mathbf{p}_j) = \frac{w_j}{2} \left\| \mathbf{A}_j \mathbf{q}_j - \mathbf{B}_j \mathbf{p}_j \right\|_F^2, \tag{4.42}$$

where $\mathbf{A}_j, \mathbf{B}_j$ are matrices of appropriate dimensions and $w_j \in \mathbb{R}$ is the weight assigned to the j-th constraint. Plugging the definition of d_j in Equation 4.42 into the PD energy potentials in Equation 4.38 yields

$$\psi_j(\mathbf{q}) = \min_{\mathbf{p}_j} \frac{w_j}{2} \left\| \mathbf{A}_j \mathbf{q}_j - \mathbf{B}_j \mathbf{p}_j \right\|_F^2 + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j). \tag{4.43}$$

Note that $\sqrt{d_j}$ is the distance function that is induced by a Frobenius norm with weights $\mathbf{A}_j, \mathbf{B}_j$ in the standard manner. Formally, d_j itself cannot be considered a distance function since it violates the absolute homogeneity property, i.e.

$$d_j(a\mathbf{q}_j, a\mathbf{p}_j) = a^2 d_j(\mathbf{q}_j, \mathbf{p}_j) \neq |a| d_j(\mathbf{q}_j, \mathbf{p}_j) \text{ for } a \in \mathbb{R}.$$

4.2.3 Projective Implicit Euler Solver

The PD solver can be derived by plugging the squared distance functions from Equation 4.42 into the variational form of implicit Euler integration for PD energy potentials (see Eq. 4.39). This yields the optimization problem given by

$$\min_{\mathbf{q}, \mathbf{p}_j} \frac{1}{2h^2} \left\| \mathbf{M}^{\frac{1}{2}} (\mathbf{q} - \tilde{\mathbf{q}}) \right\|_F^2 + \sum_j \frac{w_j}{2} \left\| \mathbf{A}_j \mathbf{S}_j \mathbf{q} - \mathbf{B}_j \mathbf{p}_j \right\|_F^2 + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j) \tag{4.44}$$

To make the relation between the projection variables $\mathbf{p}_j$ and the relevant particle positions $\mathbf{q}_j$ more apparent, we required $\dim(\mathbf{S}_j \mathbf{q}) = \dim(\mathbf{p}_j)$ so far. However,

the notation can be simplified by introducing the alternative constraint manifold $\boldsymbol{C}_j' = \{\mathbf{B}_j\mathbf{p} \mid \mathbf{p} \in \boldsymbol{C}_j\}$. Of course, this requires changing the shape of the projection variables. If $\mathbf{B}_j \in \mathbb{R}^{r\times n_j}$, then $\mathbf{p}_j \in \mathbb{R}^{r\times 3}$. If further $\mathbf{A}_j\mathbf{S}_j$ are combined into a single matrix $\mathbf{G}_j$, an equivalent optimization problem given by

$$\min_{\mathbf{q},\mathbf{p}_j} \tilde{g}(\mathbf{q},\mathbf{p}_j) = \min_{\mathbf{q},\mathbf{p}_j} \frac{1}{2h^2}\left\|\mathbf{M}^{1/2}(\mathbf{q}-\tilde{\mathbf{q}})\right\|_F^2 + \sum_j \frac{w_j}{2}\left\|\mathbf{G}_j\mathbf{q}-\mathbf{p}_j\right\|_F^2 + \delta_{\boldsymbol{C}_j'}(\mathbf{p}_j) \tag{4.45}$$

can be derived. From now on, we simply write $\boldsymbol{C}_j$ instead of $\boldsymbol{C}_j'$.

As discussed in Section 4.2.1, the local step consists of minimizing the objective function Equation 4.45 over the auxiliary variables $\mathbf{p}_j$ while keeping the positions $\mathbf{q}$ fixed. For each energy potential, we solve the following minimization problem

$$\min_{\mathbf{p}_j} \tilde{g}(\mathbf{q},\mathbf{p}_j) = \min_{\mathbf{p}_j} \frac{w_j}{2}\left\|\mathbf{G}_j\mathbf{q}-\mathbf{p}_j\right\|_F^2 + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j). \tag{4.46}$$

In the global step, the minimization problem Equation 4.45 is optimized over the positions $\mathbf{q}$ while keeping the auxiliary variables $\mathbf{p}_j$ fixed. The optimization problem for the global solve is given by

$$\min_{\mathbf{q}} \tilde{g}(\mathbf{q},\mathbf{p}_j) = \min_{\mathbf{q}} \frac{1}{2h^2}\left\|\mathbf{M}^{1/2}(\mathbf{q}-\tilde{\mathbf{q}})\right\|_F^2 + \sum_j \frac{w_j}{2}\left\|\mathbf{G}_j\mathbf{q}-\mathbf{p_j}\right\|_F^2. \tag{4.47}$$

The gradient of the objective function with respect to the positions $\nabla_{\mathbf{q}}\tilde{g}(\mathbf{q},\mathbf{p}_j)$ is given by

$$\nabla_{\mathbf{q}}\tilde{g}(\mathbf{q},\mathbf{p}_j) = \frac{1}{h^2}\mathbf{M}(\mathbf{q}-\tilde{\mathbf{q}}) + \sum_j w_j\mathbf{G}_j^T\mathbf{G}_j\mathbf{q} - \sum_j w_j\mathbf{G}_j^T\mathbf{p}_j. \tag{4.48}$$

By design of the PD energy potentials, the objective function of the global optimization problem is quadratic in the positions $\mathbf{q}$. Consequently, the minimization can be carried out in a single step by picking $\mathbf{q}$ such that the first-order optimality conditions are satisfied [NW06]. This leads to the following system of equations

$$(\frac{1}{h^2}\mathbf{M} + \sum_j w_j\mathbf{G}_j^T\mathbf{G}_j)\mathbf{q} = \frac{1}{h^2}\mathbf{M}\tilde{\boldsymbol{q}} + \sum_j w_j\mathbf{G}_j^T\mathbf{p}_j. \tag{4.49}$$

In the rest of Section 4.2 we refer to the system matrix of the global system by $\mathbf{S} \coloneqq \frac{1}{h^2}\mathbf{M} + \sum_j w_j\mathbf{G}_j^T\mathbf{G}_j$.

Note that $\mathbf{S}$ is constant as long as the constraint set remains unchanged. The right side needs to be recomputed in every iteration as the projections $\mathbf{p}_j$ change during the local optimization steps. An overview over the algorithm is given in Algorithm 4.

Algorithm 4 Projective Implicit Euler Solver

procedure SOLVEPD($\mathbf{q}_n$, $\mathbf{v}_n$, $\mathbf{f}_{\text{ext}}$, h)
  $\tilde{\mathbf{q}} = \mathbf{q}_n + h\mathbf{v}_n + h^2\mathbf{M}^{-1}\mathbf{f}_{\text{ext}}$
  $\mathbf{q}_k = \tilde{\mathbf{q}}$
  for all iterations **do**
    for constraints j **do**
      $\mathbf{p}_j = \min_{\mathbf{p}_j} \frac{w_j}{2}\left\|\mathbf{G}_j\mathbf{q}_k - \mathbf{p}_j\right\|_F^2 + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j)$
    end for
    $\mathbf{q}_k \leftarrow$ solution of $\mathbf{S}\mathbf{q} = \frac{1}{h^2}\mathbf{M}\tilde{\mathbf{q}} + \sum_j w_j\mathbf{G}_j^T\mathbf{p}_j$.
  end for
  return with $\mathbf{q}_{n+1} = \mathbf{q}_k, \mathbf{v}_{n+1} = (\mathbf{q}_{n+1} - \mathbf{q}_n)/h$
end procedure

4.2.4 Properties of PD

Projective dynamics is an efficient algorithm for implicit euler integration of the equations of motion with energy potentials of the form presented in Equation 4.43. Its properties are discussed in detail in this section.

Efficient Implementation. The structure of the PD energy potentials allows for Algorithm 4 to be implemented efficiently. Since the constraint projections in Equation 4.46 can be carried out independently, the local optimization step lends itself to massive parallelization. Further, because the system matrix $\mathbf{S}$ is constant, its prefactorization can be computed at initialization, enabling efficient solves of the linear system in the global optimization step. Note that since $\mathbf{q} \in \mathbb{R}^{m\times 3}$ in Equation 4.49, the global system can be solved independently and in parallel for each coordinate.

Generality. The last property follows from the fact that the squared distance functions d_j in Equation 4.42 have no dependencies between x-, y- and z-coordinates. This detail demonstrates that restricting to PD energy potentials comes at the cost of generality: Many arbitrary nonlinear elastic potentials, particulary those that have dependencies between x-, y- and z-coordinates, cannot be expressed in terms of PD elastic potentials. Further, since PD energy potentials ψ_j are defined in

terms of a squared distance function d_j, the resulting forces are always proportional to the distance from the constraint manifold $\boldsymbol{C}_j$ [OBLN17]. Many classical energies like the Neohookean and St. Venant-Kirchhoff energies do not fit into the PD framework [LBK17]. On the other hand, the authors show that a variety of constraints, including strain constraints, bending constraints, collisions and position constraints can be expressed in terms of PD potentials and handled by the PD solver in a unified manner [BML+14]. Where applicable, the constraints are derived from continuous energies, leaving them reasonably independent to the underlying meshing.

Hard Constraints. There is no obvious way to implement hard constraints with the PD solver. However, hard constraints can be emulated by introducing energy potentials with arbitrarily large stiffness values, effectively introducing penalty forces to the equations of motion. This approach is discussed in detail in Section 3.7.1.

Convergence. While a simplified minimization problem is constructed by restricting to PD energy potentials, the solver attempts to find the true solution of Equation 4.45 instead of computing an approximation. The objective function is quadratic, bounded below and both local and global steps are guaranteed to weakly decrease it. As a result, the optimization converges without additional safeguards, even if non-convex constraint manifolds are used in the energy potentials. However, this property alone does not ensure that PD converges to the positions that minimize the objective function of the variational form of implicit Euler integration. According to Bouaziz et al. [BML+14], the true implicit positions preserve linear and angular momentum if the elastic potential is rigid motion invariant. It is not clear whether this property is satisfied for the positions that PD converges to as well.

Changing Constraint Sets. Lastly, it is important to note that the PD solver is not suited for handling frequently changing constraint sets. For example, every time a collision is detected, a new constraint needs to be added to the simulation and the global system matrix $\mathbf{S}$ needs to be refactorized. This can slow down the PD solver quite significantly and can lead to unpredictable solver speeds that are infeasible in the context of real-time simulations.

4.2.5 Simulating Deformable Bodies Using PD

As discussed in Section 4.2.4, classical material models such as the St. Venant-Kirchhoff and Neohookean model cannot be expressed in terms of PD energy

potentials and are incompatible with the PD solver as a result. However, we demonstrate how the strain material model in Equation 3.13 can be used with PD. For each tetrahedron in the tetrahedral mesh of the simulated body, we need to construct a PD energy potential ψ_i such that

$$\psi_j(\mathbf{q}) = \min_{\mathbf{p}_j} \frac{w_j}{2} \left\| \mathbf{A}_j \mathbf{q}_j - \mathbf{B}_j \mathbf{p}_j \right\|_F^2 + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j) = V_j \psi_{\text{strain}}(\mathbf{F}_j),$$

where V_j is the volume of the undeformed tetrahedron. Recall that $\mathbf{F}_j$ is the constant deformation gradient over the volume of the tetrahedron and can be computed via Equation 3.12.

Since the energy potential of a single tetrahedron solely depends on its four tetrahedral vertices, it is $\mathbf{q}_j, \mathbf{p}_j \in \mathbb{R}^{4\times 3}$. We set the constraint manifold $\boldsymbol{C}_j$ to the set of all matrices $\mathbf{p} \in \mathbb{R}^{4\times 3}$ that correspond to undeformed tetrahedra if the rows of $\mathbf{p}$ are interpreted as the positions of the tetrahedral vertices. Let $\mathbf{D} \in \mathbb{R}^{3\times 4}$ be the matrix that maps such matrices $\mathbf{p}$ to the transpose of their deformation gradient $\mathbf{F}_{\mathbf{p}}^T$. Then $\boldsymbol{C}_j$ can be defined as

$$\boldsymbol{C}_j = \{\mathbf{p} \mid \mathbf{D}\mathbf{p} = \mathbf{F}_{\mathbf{p}}^T \in \text{SO}(3)\}, \tag{4.50}$$

where $\text{SO}(3)$ is the group of three-dimensional rotational matrices. We set $\mathbf{A}_j = \mathbf{B}_j = \mathbf{D}$ and $w_j = k_j V_j$, where V_j is the volume of the undeformed tetrahedron and k_j is a user-defined stiffness value. Together, it is

$$\begin{aligned} \psi_j(\mathbf{q}) = \min_{\mathbf{p}_j} W(\mathbf{S}_j \mathbf{q}, \mathbf{p}_j) &= \min_{\mathbf{p}_j} \frac{k_j V_j}{2} \left\| \mathbf{D}\mathbf{q}_j - \mathbf{D}\mathbf{p}_j \right\|_F^2 + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j) \\ &= \min_{\mathbf{F}_{\mathbf{P}_j}} \frac{k_j V_j}{2} \left\| \mathbf{F}_j^T - \mathbf{F}_{\mathbf{p}_j}^T \right\|_F^2 + \delta_{\text{SO}(3)}(\mathbf{F}_{\mathbf{p}_j}). \end{aligned} \tag{4.51}$$

It is easy to show that $\mathbf{F}_{\mathbf{p}_j}$ can be computed as $\mathbf{F}_{\mathbf{p}_j} = \mathbf{U}\mathbf{I}\mathbf{V}^T$ where $\mathbf{F}_j = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$ is the singular value decomposition of the deformation gradient $\mathbf{F}_j$. Plugging into Equation 4.51 yields

$$\psi_j(\mathbf{q}) = \frac{k_j V_j}{2} \left\| \mathbf{U}^T \boldsymbol{\Sigma} \mathbf{V} - \mathbf{U}^T \mathbf{I} \mathbf{V} \right\|_F^2 = \frac{k_j V_j}{2} \|\boldsymbol{\Sigma} - \mathbf{I}\|_F^2 = V_j \psi_{\text{strain}}(\mathbf{F}_i),$$

showing that the strain material model can be expressed in terms of PD energy potentials.

4.2.6 PD as a Special Case of Quasi-Newton Methods

In PD, the variational form of implicit Euler integration that results from restricting to PD energy potentials (see Eq. 4.45) is solved by using a specialized lo-

cal/global alternating minimization technique (see Sec. 4.2.3). If the projection points $\mathbf{p}_{\mathbf{q}}^{j}$ with

$$\mathbf{p}_{\mathbf{q}_j} = \underset{\mathbf{p}_j}{\operatorname{argmin}}\ \psi_j(\mathbf{q}) = \underset{\mathbf{p}_j}{\operatorname{argmin}}\ \frac{w_j}{2}\left\|\mathbf{G}_j\mathbf{q} - \mathbf{p}_j\right\|_F^2 + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j)$$

are considered functions of $\mathbf{q}$ with $\mathbf{p}_j(\mathbf{q}) = \mathbf{p}_{\mathbf{q}}^{j}$, then the equivalent optimization problem

$$\min_{\mathbf{q}} g(\mathbf{q}) = \min_{\mathbf{q}} \frac{1}{2h^2}\left\|\mathbf{M}^{1/2}(\mathbf{q} - \tilde{\mathbf{q}})\right\|_F^2 + \sum_j \frac{w_j}{2}\left\|\mathbf{G}_j\mathbf{q} - \mathbf{p_j}(\mathbf{q})\right\|_F^2 \tag{4.52}$$

can be solved using general purpose algorithms for unconstrained optimization. We show that the PD solver can be interpreted as a Quasi-Newton method applied to the objective function g in Equation 4.52 [LBK17].

We start by briefly introducing Newton's method, which is one of the most popular algorithms for unconstrained optimization. For a more detailed introduction, we refer the interested reader to the relevant chapters in the textbook on numerical optimization by Nocedal and Wright [NW06]. Newton's method is an iterative procedure that starts with some initial guess $\mathbf{q}_0$ and then iteratively improves it by applying updates in a direction that is guaranteed to reduce the objective function until a local minimum is achieved. Applying Newton's method to the objective function g in Equation 4.52 yields the following update formula

$$\mathbf{q}_{k+1} = \mathbf{q}_k - \alpha_k \nabla^2 g(\mathbf{q}_k)^{-1} \nabla g(\mathbf{q}_k) = \mathbf{q}_k - \alpha_k \mathbf{r}_k^N. \tag{4.53}$$

Here, $\nabla^2 g(\mathbf{q}_k)$ is the Hessian matrix of g at $\mathbf{q}_k$, $\mathbf{r}_k^N = \nabla^2 g(\mathbf{q}_k)^{-1}\nabla g(\mathbf{q}_k)$ is called the Newton direction and $\alpha_k \in \mathbb{R}^+$ is called the step size. If $\nabla^2 g(\mathbf{q}_k)$ is invertible and α_k is sufficiently small, the Newton update is guaranteed to decrease the objective function. However, to ensure that each Newton update makes sufficient progress towards a local minimum, it is desirable to keep α_k as large as possible. Newton's method exhibits favorable convergence properties, but the computation and storage of the inverse Hessian matrix $\nabla^2 g(\mathbf{q}_k)^{-1}$ is often prohibitively expensive. This gives rise to the so-called Quasi-Newton methods. There, the Hessian matrix $\nabla^2 g(\mathbf{q}_k)$ is approximated by some matrix $\mathbf{B}_k$ that is faster to compute, but still yields suitable search directions $\mathbf{r}_k$. The update formula for Quasi-Newton methods is given by

$$\mathbf{q}_{k+1} = \mathbf{q}_k - \alpha_k \mathbf{B}_k^{-1} \nabla g(\mathbf{q}_k). \tag{4.54}$$

Now, consider the Quasi-Newton update formula with constant step size $\alpha_k = 1$ and Hessian approximation $\mathbf{B}_k = \mathbf{S}$, where $\mathbf{S}$ is the system matrix from the global optimization in PD (see Eq. 4.49), given by

$$\mathbf{q}_{k+1} = \mathbf{q}_k - \mathbf{S}_k^{-1} \nabla g(\mathbf{q}_k). \tag{4.55}$$

Liu et al. [LBK17] show that ∇g is equal to

$$\nabla g(\mathbf{q}) = \frac{1}{h^2}\mathbf{M}(\mathbf{q} - \tilde{\mathbf{q}}) + \sum_j w_j \mathbf{G}_j^T \mathbf{G}_j \mathbf{q} - \sum_j w_j \mathbf{G}_j^T \mathbf{p}_j(\mathbf{q}). \tag{4.56}$$

Surprisingly, this is the same as the gradient $\nabla_{\mathbf{q}} \tilde{g}(\mathbf{q}, \mathbf{p}_j)$ of the global optimization problem of the original PD algorithm (see Eq. 4.48). Thus, the gradient is unaffected by whether $\mathbf{p}_j$ is considered constant or a function $\mathbf{p}_j(\mathbf{q})$ of the positions $\mathbf{q}$. Plugging $\nabla g(\boldsymbol{q}_k)$ into Equation 4.55 results in

$$\begin{aligned}
\mathbf{q}_{k+1} &= \mathbf{q}_k - \mathbf{S}^{-1} \left(\frac{1}{h^2}\mathbf{M}(\mathbf{q}_k - \tilde{\mathbf{q}}) + \sum_j w_j \mathbf{G}_j^T \mathbf{G}_j \mathbf{q}_k - \sum_j w_j \mathbf{G}_j^T \mathbf{p}_j(\mathbf{q}_k) \right) \\
&= \mathbf{q}_k - \mathbf{S}^{-1} \left(\mathbf{S}\mathbf{q}_k - \frac{1}{h^2}\tilde{\mathbf{q}} - \sum_j w_j \mathbf{G}_j^T \mathbf{p}_j(\mathbf{q}_k) \right) \\
&= \mathbf{S}^{-1} \left(\frac{1}{h^2}\tilde{\mathbf{q}} + \sum_j w_j \mathbf{G}_j^T \mathbf{p}_j(\mathbf{q}_k) \right).
\end{aligned}$$

Note that this is exactly the solution of the global linear system from PD in Equation 4.49. Together, this shows that performing a Quasi-Newton step with Hessian approximation $\mathbf{B}_k = \mathbf{S}$ and step size $\alpha_k = 1$ on the minimization problem in Equation 4.52 is equivalent to performing a local/global iteration of the PD solver (Algorithm 4). The Quasi-Newton version of PD is summarized in Algorithm 5.

Algorithm 5 Projective Dynamics as a Quasi-Newton Method

procedure SOLVEPDVIAQN($\mathbf{q}_n$, $\mathbf{v}_n$, $\mathbf{f}_{\text{ext}}$, h)
  $\tilde{\mathbf{q}} = \mathbf{q}_n + h\mathbf{v}_n + h^2\mathbf{M}^{-1}\mathbf{f}_{\text{ext}}$
  $\mathbf{q}_k = \tilde{\mathbf{q}}$
  for all iterations **do**
    $\mathbf{p}_k \leftarrow$ solution of $\mathbf{S}\mathbf{p} = -\nabla g(\mathbf{q}_k)$
    $\mathbf{q}_k = \mathbf{q}_k + \mathbf{p}_k$
  end for
  return with result $\mathbf{q}_{n+1} = \mathbf{q}_k, \mathbf{v}_{n+1} = (\mathbf{q}_{n+1} - \mathbf{q}_n)/h$
end procedure

The insight that PD is a special case of Quasi-Newton methods applied to Equation 4.52 can be leveraged to bring performance improvements to the original PD solver and to design a natural extension of PD to energies that do not fit into the original PD framework. Both are discussed in Section 4.3.

4.3 Quasi-Newton Methods for Physical Simulations

In Section 4.2.6, we discussed that the PD solver for the minimization problem in Equation 4.45 can be interpreted as a special case of a Quasi-Newton method with constant Hessian approximation and step size $\alpha = 1$ applied to the corresponding minimization problem Equation 4.52. The formulation that is suitable for Quasi-Newton methods

$$\min_{\mathbf{q}} g(\mathbf{q}) = \min_{\mathbf{q}} \frac{1}{2h^2} \left\| \mathbf{M}^{1/2}(\mathbf{q} - \tilde{\mathbf{q}}) \right\|_F^2 + \sum_j \frac{w_j}{2} \left\| \mathbf{G}_j \mathbf{q} - \mathbf{p_j}(\mathbf{q}) \right\|_F^2 \tag{4.57}$$

and the global system matrix $\mathbf{S}$ serving as the constant Hessian approximation

$$\mathbf{S} \coloneqq \frac{1}{h^2}\mathbf{M} + \sum_j w_j \mathbf{G}_j^T \mathbf{G}_j \tag{4.58}$$

are restated for convenience of the reader. This highlights one of the weaknesses of the PD solver. Since the Hessian approximation is constant, the PD solver does not take advantage of local curvature information around the current positions $\boldsymbol{q}_k$ during its position update. This is in contrast to Newton's method, where the curvature of the objective function around $\boldsymbol{q}_k$ is captured by the Hessian matrix $\nabla^2 g(\mathbf{q}_k)$ and is used to find an effective search direction. Most Quasi-Newton methods aim at computing Hessian approximations $\mathbf{B}_k$ that are as close to $\nabla^2 g(\mathbf{q}_k)$ as possible and capture the local curvature of the objective function reasonably well as a result. The L-BFGS method is a Quasi-Newton method that achieves this by computing a Hessian approximation $\mathbf{B}_k$ from some initial Hessian approximation $\mathbf{B}_0$, the previous $l \in \mathbb{N}^+$ iterates $\mathbf{q}_{k-1}, \ldots, \mathbf{q}_{k-l}$ and their gradients $\nabla g(\mathbf{q}_{k-1}), \ldots, \nabla g(\mathbf{q}_{k-l})$ [NW06]. The choice of the initial Hessian approximation $\mathbf{B}_0$ has a strong impact on the performance of the L-BFGS method. Usually, a scaled version of the identity matrix is used. Liu et al. [LBK17] suggest that setting $\mathbf{B}_0 = \mathbf{S}$ combines the strengths of the PD solver and L-BFGS method and gives rise to a powerful approach to implicit Euler integration when applied to the optimization problem in Equation 4.57. In the rest of this thesis, we refer to the resulting solver as the QN solver. It is discussed in detail in Section 4.3.1.

Since Quasi-Newton methods can be applied to the variational form of implicit Euler integration with general conservative energy potentials (see Eq. 3.7),

this suggests a natural extension of the approach mentioned above to energies that do not fit into the original PD framework. However, if general conservative energy potentials are used, it is not obvious how to construct the initial Hessian approximation $\mathbf{S}$ anymore. Liu et al. suggest a way to emulate the global system matrix $\mathbf{S}$ for energies that can be written in the Valanis-Landel form, which includes Neo-hookean and St. Venant-Kirchhoff energies. This extension is covered in Section 4.3.2. Finally, the properties of the QN solver are discussed in Section 4.3.3.

4.3.1 Quasi-Newton Methods for PD Energy Potentials

An overview over the L-BFGS method with initial Hessian approximation $\mathbf{B}_0 = \mathbf{S}$ applied to the optimization problem in Equation 4.57 is given in Algorithm 6. For more details on the L-BFGS method, we again refer to the book on numerical optimization by Nocedal and Wright [NW06]. To start off the L-BFGS solver, an initial guess $\mathbf{q}_0$ for a local minimizer of the objective function in Equation 4.57 is required. In the QN solver, the initial guess is set to the inertial positions, i.e. $\mathbf{q}_0 = \tilde{\mathbf{q}}$ (lines 3-4). For each iteration k, the search direction $\mathbf{p}_k = -\mathbf{B}_k^{-1}\nabla g(\mathbf{q}_k)$ is computed in lines 6-7. In lines 8-12, an appropriate step size α_k for the current search direction $\mathbf{p}_k$ is determined. Given the step size α_k and the search direction $\mathbf{p}_k$, the next iterate is computed in line 12. Finally, the positions and the velocities for the next time step are updated after the last iteration in line 15.

Algorithm 6 L-BFGS Method For PD Energies

1: **require** $\beta \in (0,1), t \in (0,1)$
2: **procedure** SOLVEPDVIALBFGS($\mathbf{q}_n$, $\mathbf{v}_n$, $\mathbf{f}_{\text{ext}}$, m, h)
3: $\quad\tilde{\mathbf{q}} = \mathbf{q}_n + h\mathbf{v}_n + h^2\mathbf{M}^{-1}\mathbf{f}_{\text{ext}}$
4: $\quad\mathbf{q}_k = \tilde{\mathbf{q}}$
5: $\quad$**for** all iterations **do**
6: $\quad\quad\mathbf{s_k} = \mathbf{q}_k - \mathbf{q}_{k-1}, \mathbf{y}_k = \nabla g(\mathbf{q}_k) - \nabla g(\mathbf{q}_{k-1}), \rho_k = 1/\text{tr}(\mathbf{s}_k^T\mathbf{y}_k)$
7: $\quad\quad\mathbf{p}_k = \text{TWOLOOPRECURSION}(\mathbf{S}, \mathbf{q}_k, \mathbf{s}, \mathbf{y}, \rho, m, k)$ (Algorithm 7)
8: $\quad\quad\alpha_k = 1$
9: $\quad\quad$**if** α_k does not satisfy the strong Wolfe conditions **then**
10: $\quad\quad\quad$compute α_k that satisfies the strong Wolfe conditions
11: $\quad\quad\quad$**or** $\alpha_k = \text{BACKTRACK}(\mathbf{q}_k, \mathbf{p}_k, 1, \beta, t)$ (Algorithm 8)
12: $\quad\quad$**end if**
13: $\quad\quad\mathbf{q}_k = \mathbf{q}_k + \alpha_k\mathbf{p}_k$
14: $\quad$**end for**
15: $\quad$**return with result** $\mathbf{q}_{n+1} = \mathbf{q}_k, \mathbf{v}_{n+1} = (\mathbf{q}_{n+1} - \mathbf{q}_n)/h$
16: **end procedure**

The search direction $\mathbf{p}_k$ is computed using the two-loop recursion shown in Algorithm 7 [NW06; LBK17]. Here, the Hessian approximation used in the L-BFGS method $\mathbf{B}_k$ does not need to be stored or computed explicitly. Instead, the search direction $\mathbf{p}_k$ can be calculated directly at the cost of $\mathcal{O}(l)$ inner products and vector additions (lines 3-5 and 8-10) and solving an LSE with constant system matrix $\mathbf{S}$ (line 7). The quantities $\mathbf{s}_k, \mathbf{y}_k$ and ρ_k are computed from the positions and gradients from previous iterations and are stored over a window of l iterations (see Alg. 6, line 6). This enables capturing curvature information during the computation of the search direction $\mathbf{p}_k$ in the two-loop recursion.

Algorithm 7 L-BFGS Two-Loop Recursion

1: **procedure** TWOLOOPRECURSION($\mathbf{B}_0$, $\mathbf{q}_k$, $\mathbf{s}$, $\mathbf{y}$, ρ, l, k)
2: $l^* = \min(l, k), \mathbf{t} = -\nabla g(\mathbf{q}_k)$
3: **for** $i = k-1, k-2, \ldots, k-l^*$ **do**
4: $\alpha_i = \rho_i \text{tr}(\mathbf{s}_i^T \mathbf{t})$
5: $\mathbf{t} = \mathbf{t} - \alpha_i \mathbf{y}_i$
6: **end for**
7: solve $\mathbf{B}_0 \mathbf{r} = \mathbf{t}$
8: **for** $i = k-l^*, k-l^*+1, \ldots, k-1$ **do**
9: $\beta = \rho_i \text{tr}(\mathbf{y}_i^T \mathbf{r})$
10: $\mathbf{r} = \mathbf{r} + \mathbf{s}_i(\alpha_i - \beta)$
11: **end for**
12: **return with result** -$\mathbf{B}_k^{-1} \nabla g(\mathbf{q}_k) = \mathbf{r}$.
13: **end procedure**

Step sizes α_k for the L-BFGS method are usually chosen such that they satisfy the Wolfe conditions, which ensures that sufficient progress towards a local minimizer is made during each L-BFGS iteration (see [NW06]). Finding such step sizes typically requires the execution of so-called step size selection algorithms. Instead, Liu et al. [LBK17] opt for a simpler strategy called backtracking (see Alg. 8). There, the natural step size associated with Newton's method $\alpha_k = 1$ is iteratively reduced by a factor of β until the first Wolfe condition is satisfied (lines 4-5). This is guaranteed to happen if $\mathbf{p}_k$ is a descent direction and if α_k is sufficiently small. In practice, backtracking is stopped after $\alpha_k < t$ for some user defined threshold t to prevent pathologically small step sizes (line 4).

4.3.2 Quasi-Newton Methods for Valanis-Landel Energies

To achieve satisfactory performance when applying the L-BFGS method to the variational form of implicit Euler integration (see Eq. 3.7) without the need to

Algorithm 8 Backtracking Line Search

require $\tilde{\alpha} > 0, c_1 \in (0,1), \beta \in (0,1), t \in (0, \tilde{\alpha})$
procedure BACKTRACK($\mathbf{q}_k$, $\mathbf{p}_k$, $\tilde{\alpha}$, β, t)
    $\alpha = \tilde{\alpha}$
    while $g(\mathbf{q}_k + \alpha \mathbf{p}_k) > g(\mathbf{q}_k) + c_1 \alpha \nabla g(\mathbf{q}_k)^T \mathbf{p}_k$ and $\alpha > t$ **do**
        $\alpha = \beta \alpha$
    end while
    return with $\alpha_k = \alpha$
end procedure

restrict energy potentials to the PD framework (see Eq. 4.43), a suitable candidate for the initial Hessian approximation $\mathbf{B}_0$ or its inverse $\mathbf{H}_0$ needs to be found. The choice $\mathbf{B}_0 = \mathbf{S} = \frac{1}{h^2}\mathbf{M} + \sum_j w_j \mathbf{G}_j^T \mathbf{G}_j$ from Algorithm 6 cannot be applied to the general setting without modification as the matrices $\mathbf{G}_j$ are taken directly from the definitions of the individual PD energy potentials.

We focus on the setting of tetrahedral meshes with energy potentials that are derived from one of the material models in Section 3.6 for each tetrahedron. Here, the energy of the entire body is the sum of the individual tetrahedral energies (see Sec. 3.5). As shown in Section 4.2.5, the energy of a single tetrahedron can be emulated using PD potentials when the strain material model is used (see Eq. 4.51). Here, $\mathbf{G}_j = \mathbf{A}_j$, where $\mathbf{A}_j$ is the linear operator that maps $\mathbf{q}$ to the tranpose of the deformation gradient $\mathbf{F}_j^T$, $\boldsymbol{C}_j = \mathrm{SO}(3)$ and $w_j = k_j V_j$, where V_j is the volume of the undeformed tetrahedron and k_j is a user-defined stiffness value. Liu et al. suggest approximating more general tetrahedral energies with PD strain potentials and using the global system matrix $\mathbf{S}$ that arises as the initial Hessian approximation $\mathbf{B}_0$ given by

$$\mathbf{B}_0 = \mathbf{S} = \frac{1}{h^2}\mathbf{M} + \sum_j w_j \mathbf{A}_j^T \mathbf{A}_j. \tag{4.59}$$

Instead of using user-defined stiffness values k_j, a procedure for setting k_j from the original, more general energy potential is required. The discussion is restricted to isotropic and world-space rotation invariant material models that can be defined in terms of the singular values $\sigma_1, \sigma_2, \sigma_3$ of $\mathbf{F_j}$ called the principal stretches [SB12]. Liu et al. [LBK17] present a strategy for a subclass of these material models that can be written in Valanis-Landel form

$$\Psi(\sigma_1, \sigma_2, \sigma_3) = a(\sigma_1) + a(\sigma_2) + a(\sigma_3) + b(\sigma_1\sigma_2) + b(\sigma_2\sigma_3) + b(\sigma_1\sigma_3) \tag{4.60}$$

with $a, b, c : \mathbb{R} \to \mathbb{R}$. Many popular material models including the St. Venant-Kirchhoff model and the Neohookean model can be expressed in this form.

The approach uses the insight that for linear materials that follow Hooke's law, the stiffness k_j is given as the second derivative of the energy potential. Computing first and second partial derivatives in the rest configuration $\sigma_1, \sigma_2, \sigma_3 = 1$ yields

$$\left.\frac{d\psi}{d\sigma_i}\right|_{\sigma_j,\sigma_k=1} = a'(\sigma_i) + 2b'(\sigma_i) + c'(\sigma_i) \tag{4.61}$$

$$\left.\frac{d^2\psi}{d\sigma_i}\right|_{\sigma_j,\sigma_k=1} = a''(\sigma_i) + 2b''(\sigma_i) + c''(\sigma_i), \tag{4.62}$$

where $i, j, k \in [1, 3], i \neq j \neq k$. Thus, the first and second partial derivatives at the rest configuration are the same for each of the principal stretches, allowing for convenient representation of the material stiffness in a single scalar value. However, simply picking

$$k_j = \left.\frac{d^2\psi}{d\sigma_i}\right|_{\sigma_i,\sigma_j,\sigma_k=1} = a''(1) + 2b''(1) + c''(1)$$

runs into issues as the expression often evaluates to zero, even in common non-pathological cases. This issue arises because the second derivative in Equation 4.62 is not representative of the stiffness behavior of the material in the entire range of principal stretches $[\sigma_{\text{min}}, \sigma_{\text{max}}]$ that is expected to be encountered during the simulation.

To alleviate this, Liu et al. [LBK17] propose approximating the rate of change of the first derivative Equation 4.61 by computing the slope of its best linear approximation over the interval $[\sigma_{\text{min}}, \sigma_{\text{max}}]$. Formally, k is defined by

$$k \coloneqq \underset{k}{\text{argmin}} \int_{\sigma_{\text{min}}}^{\sigma_{\text{max}}} (k(\sigma - 1) - (a'(\sigma) + 2b'(\sigma) + c'(\sigma))^2 d\sigma \tag{4.63}$$

Using these stiffness values in Equation 4.59 yields a suitable initial Hessian approximation $\mathbf{B}_0 = S$ for the minimization of the variational form of the implicit Euler integration with Valanis-Landel materials via Algorithm 6. According to Liu et al. Algorithm 6 is insensitive to the size of the interval $[\sigma_{\text{min}}, \sigma_{\text{max}}]$. It is important to note that while PD strain energies are used to approximate the original energy potentials when constructing $\mathbf{B}_0$, the function evaluations $g(\mathbf{q}_k)$ and gradients $\nabla g(\mathbf{q}_k)$ are computed from the original potential energies in Algorithm 6.

4.3.3 Properties of the QN Solver

The QN solver described in Sections 4.3.1 and 4.3.2 emerges from combining ideas from PD and the L-BFGS method for unconstrained optimization. As a result, the QN solver has superior convergence properties and is more general than the PD solver. We discuss the properties of the QN solver below.

Relation to the PD Solver. As described in Section 4.2.6, the PD solver can be interpreted as a Quasi-Newton method with constant step size and Hessian approximation applied to the optimization problem in Equation 4.57. The QN solver enhances the PD solver by applying L-BFGS updates that capture local curvature information to the constant Hessian approximation $\mathbf{S}$ and performing backtracking to find a step size that is guaranteed to at least satisfy the first Wolfe condition. Experimental data provided by Liu et al. [LBK17] suggests that this leads to significantly improved convergence properties. This is in line with the fact that the convergence properties of Quasi-Newton methods benefit from Hessian approximations $\mathbf{B}_k$ that are as close as possible to the true Hessian matrix $\nabla^2 g(\mathbf{q}_k)$ [NW06]. As described in Section 4.3.2, the QN solver can be applied to material models that can be written in the Valanis-Landel form. Thus, the QN solver is more general than the PD solver.

Step Size. In our discussion of PD (see Sec. 4.2.4), we pointed out that PD iterations are guaranteed to weakly decrease the objective function of the variational form of implicit Euler integration. As such, using a constant step size of $\alpha_k = 1$ is a viable choice when the constant Hessian approximation $\mathbf{B}_k = \mathbf{S}$ is chosen. In contrast, it is well known that search directions $\mathbf{p}_k$ produced by the L-BFGS method may not be directions of descent if the step sizes α_k are not guaranteed to satisfy the Wolfe conditions [NW06]. In such settings, convergence of the L-BFGS method is not guaranteed. Since the step size α_k determined by backtracking (see Alg. 6) may not satisfy the second Wolfe condition, this applies for the QN solver as well. However, Liu et al. [LBK17] report that skipping more complicated step size selection algorithms in favor of simple backtracking works well in practice. In contrast, leaving out backtracking all together and simply using $\alpha_k = 1$ produces poor results according to the authors. Thus, it is essential that step sizes at the very least satisfy the first Wolfe condition.

Efficient Implementation. The QN solver lends itself to efficient implementation. Firstly, the contributions of each tetrahedral element to the evaluations of the objective function g and its gradient ∇g can be computed independently and are easily parallelizable as a result. Secondly, the two-loop recursion (see Alg. 7)

can be performed at the cost of $\mathcal{O}(l)$ inner products and vector additions (lines 3-5 and 8-10) and solving an LSE with constant system matrix $\mathbf{S}$ (line 7). The total computational cost incurred by the inner products and vector additions is only $\mathcal{O}(lm)$, where m is the number of particles. Solving the LSE can be sped up by precomputing and storing a matrix factorization of $\mathbf{S}$. Lastly, the LSE can be solved independently for x-, y- and z- coordinates since $\mathbf{t} \in \mathbb{R}^{m \times 3}$.

Choice of L-BFGS History Window Size. The user defined L-BFGS window size l needs to be large enough to sufficiently capture local curvature information. On the other hand, L-BFGS performance can be negatively affected by taking into account distant iterates if l is too large. Liu et al. [LBK17] recommend a window size of $l = 5$. Additionally, the authors report that it may be useful to reduce the window size for simulations with frequent collisions.

Changing Constraint Sets. In PD, the system matrix $\mathbf{S}$ and its factorization need to be recomputed each time a constraint is added to or removed from the simulation. This is not the case for the initial Hessian approximation $\mathbf{B}_0$ used in the QN solver [LBK17]. When a new constraint is added, it is accounted for by the contribution of its energy or energy gradient to the evalution of g and ∇g. Ignoring its contribution to the initial matrix approximation $\mathbf{B}_0$ can be interpreted as a more aggressive Hessian approximation that might negatively affect the convergence rate of the solver. However, with an an appropriate step size selection algorithm in place, the solver will still successfully decrease the objective function g. This suggests that it might be helpful to replace backtracking with a step size selection algorithm that ensures that the step size α_k satisfies both Wolfe conditions during simulations with rapidly changing constraint sets. This includes collision heavy simulations.

Hard Constraints. The challenges of implementing hard constraints with the QN solver are identical to those discussed for the PD solver in Section 4.2.4.

4.4 Alternating Direction Method of Multipliers

Both the PD solver covered in Section 4.2 and the QN solver introduced in Section 4.3 perform implicit Euler integration by minimizing the objective function of the variational form of implicit Euler integration

$$\min_{\boldsymbol{q}_{n+1}} \frac{1}{2h^2} \left\| \mathbf{M}^{\frac{1}{2}} (\boldsymbol{q}_{n+1} - \tilde{\boldsymbol{q}}) \right\|_F^2 + \sum_j \psi_j(\boldsymbol{q}_{n+1}). \tag{4.64}$$

Since PD can be interpreted as a Quasi-Newton method with constant step size and Hessian approximation (see Sec. 4.2.6), both of these solvers approach the minimization problem in Equation 4.64 through the lense of unconstrained optimization. It turns out that Equation 4.64 can be expressed as an equivalent constrained optimization problem of the form

$$\min_{\boldsymbol{q},\boldsymbol{z}} f(\boldsymbol{q}) + g(\boldsymbol{z}) \tag{4.65}$$

$$\text{subject to } \mathbf{A}\boldsymbol{q} + \mathbf{B}\boldsymbol{z} = \boldsymbol{c} \tag{4.66}$$

for appropriate functions f, g, matrices $\mathbf{A}, \mathbf{B}$ and vectors $\boldsymbol{q}, \boldsymbol{z}, \boldsymbol{c}$. This allows approaching implicit Euler integration using tools from the constrained optimization literature. Overby et al. [OBLN17] propose performing implicit Euler integration by solving the corresponding constrained optimization problem by a popular algorithm for constrained optimization called the Alternating Direction Method of Multipliers (ADMM). We refer to the resulting solver as the ADMM solver. The ADMM solver is highly parallelizable, can be applied to arbitrary conservative forces and supports hard constraints. Overby et al. [OBLN17] show that the ADMM solver is equivalent to PD in the special case of linear forces. Thus, similar to the QN solver, ADMM can be considered an extension of PD. We describe how to apply ADMM to the variational form of implicit Euler integration in Section 4.4.1. In Section 4.4.2, we outline how hard constraints can be implemented with the ADMM solver. In Section 4.4.3, the properties of the ADMM solver are discussed.

4.4.1 ADMM Solver

ADMM is an algorithm for solving constrained optimization problems of the form given in Equation 4.65. The algorithm takes initial values $\boldsymbol{q}_0$, $\boldsymbol{z}_0$ and $\boldsymbol{u}_0$ and iteratively updates them according to the following formulas [GQ20]

$$\begin{aligned}
\boldsymbol{q}_{k+1} &= \underset{\boldsymbol{q}}{\operatorname{argmin}} \left(f(\boldsymbol{q}) + \frac{\rho}{2} \|\mathbf{A}\boldsymbol{q} + \mathbf{B}z_k - \boldsymbol{c} + \boldsymbol{u}_k\|^2 \right) \\
\boldsymbol{z}_{k+1} &= \underset{\boldsymbol{z}}{\operatorname{argmin}} \left(g(\boldsymbol{z}) + \frac{\rho}{2} \|\mathbf{A}\boldsymbol{q}_{k+1} + \mathbf{B}\boldsymbol{z} - \boldsymbol{c} + \boldsymbol{u}_k\|^2 \right) \\
\boldsymbol{u}_{k+1} &= \boldsymbol{u}_k + (\mathbf{A}\boldsymbol{q}_{k+1} + \mathbf{B}\boldsymbol{z}_{k+1} - \boldsymbol{c}).
\end{aligned} \tag{4.67}$$

Here, $\boldsymbol{u}$ is an auxiliary scaled dual variable, $\rho \in \mathbb{R}^+$ is a step length parameter and the subscripts denote the current iteration.

To perform implicit Euler integration using ADMM, we express the variational form of implicit Euler integration (see Eq. 4.64) in terms of a constrained optimization problem of the form in Equation 4.65 [OBLN17]. To achieve this, we rewrite the individual energy terms ψ_j in terms of local variables $\boldsymbol{z}_j = \mathbf{D}_j\boldsymbol{q}$, where $\mathbf{D}_j$ is a so-called reduction matrix. Then, the energy potential Ψ can be rewritten as follows

$$\Psi(\boldsymbol{q}) = \sum_j \psi_j(\mathbf{D}_j\boldsymbol{q}) = \sum_j \psi_j(\boldsymbol{z}_j).$$

As an example, if $\psi_j(\mathbf{D}_j\boldsymbol{q})$ measures the elastic energy of tetrahedron j at configuration $\boldsymbol{q}$ according to one of the material models introduced in Section 3.6, then $\mathbf{D}_j$ can be chosen such that $\mathbf{D}_j\boldsymbol{q} = \text{flatten}(\mathbf{F}_j)$, where $\mathbf{F}_j$ is the deformation gradient of tetrahedron j. If the individual reduction matrices $\mathbf{D}_j$ are stacked into a single matrix $\mathbf{D} = [\mathbf{D}_1^T, \ldots, \mathbf{D}_r^T]^T$ the concatenation of the local coordinates $\boldsymbol{z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_r]$ is given by $\boldsymbol{z} = \mathbf{D}\boldsymbol{q}$. This allows introducing the function Ψ_* with

$$\Psi_*(\mathbf{D}\boldsymbol{q}) = \sum_j \psi_j(\mathbf{D}_j\boldsymbol{q}) = \Psi(\boldsymbol{q}).$$

Then, the variational form of implicit Euler integration can be rewritten as follows

$$\min_{\boldsymbol{q}_{n+1}} \frac{1}{2h^2}\left\|\mathbf{M}^{\frac{1}{2}}(\boldsymbol{q}_{n+1} - \tilde{\boldsymbol{q}})\right\|_F^2 + \Psi_*(\mathbf{D}\boldsymbol{q}_{n+1}). \tag{4.68}$$

In the context of the ADMM solver, $\boldsymbol{q} \in \mathbb{R}^{3m}$ and $\mathbf{M} \in \mathbb{R}^{3m\times 3m}$, where $m \in \mathbb{N}^+$ is the number of particles. By setting

$$\begin{aligned} f(\boldsymbol{q}) &= \frac{1}{2h^2}\left\|\mathbf{M}^{\frac{1}{2}}(\boldsymbol{q} - \tilde{\boldsymbol{q}})\right\|^2, \\ g(\boldsymbol{z}) &= \Psi_*(\boldsymbol{z}), \\ \mathbf{A} &= \mathbf{W}\mathbf{D}, \\ \mathbf{B} &= -\mathbf{W}, \\ \boldsymbol{c} &= \mathbf{0}, \end{aligned} \tag{4.69}$$

for any invertible weight matrix $\mathbf{W}$, the variational form of implicit Euler integration can be expressed in terms of the constrained optimization problem in Equation 4.65. Typically, $\mathbf{W}$ is chosen to be block diagonal with blocks $\mathbf{W}_j$ of the same size as the corresponding local vectors $\boldsymbol{z}_j$. In this thesis, we use $\mathbf{W}_j = w_j\mathbf{I}$ for user defined weights $w_j \in \mathbb{R}^+$.

Plugging the identities from Equation 4.69 into the ADMM update formulas in Equation 4.67 yields

$$\boldsymbol{q}_{k+1} = (\mathbf{M} + \rho h^2 \mathbf{D}^T \mathbf{W}^T \mathbf{W} \mathbf{D})^{-1} \left(\mathbf{M}\tilde{\boldsymbol{q}} + \rho h^2 \mathbf{D}^T \mathbf{W}^T \mathbf{W} (\boldsymbol{z}_k - \bar{\boldsymbol{u}}_k) \right), \quad (4.70)$$

$$\boldsymbol{z}_{k+1} = \underset{\boldsymbol{z}}{\operatorname{argmin}} \left(\Psi_*(\boldsymbol{z}) + \frac{\rho}{2} \left\| \mathbf{W}(\mathbf{D}\boldsymbol{q}_{k+1} - \boldsymbol{z} + \bar{\boldsymbol{u}}_k) \right\|^2 \right), \quad (4.71)$$

$$\bar{\boldsymbol{u}}_{k+1} = \bar{\boldsymbol{u}}_k + \mathbf{D}\boldsymbol{q}_{k+1} - \boldsymbol{z}_{k+1}, \quad (4.72)$$

where $\bar{\boldsymbol{u}} = \mathbf{W}^{-1}\boldsymbol{u}$. We point out that the matrix $\mathbf{M} + \rho h^2 \mathbf{D}^T \mathbf{W}^T \mathbf{W} \mathbf{D}$ in the $\boldsymbol{q}$-update is constant across the entire simulation if the set of energy terms remains unchanged. Finally, note that ρ can be absorbed into $\mathbf{W}$ via $\mathbf{W} \leftarrow \sqrt{\rho}\mathbf{W}$. Thus, ρ is ommitted in the following for convenience.

Due to the separable structure of Ψ_* and $\mathbf{W}$, the $\boldsymbol{z}$- and $\bar{\boldsymbol{u}}$-updates can be performed independently for each energy term. For each ψ_j, the corresponding subvectors $\boldsymbol{z}_j$ and $\bar{\boldsymbol{u}}_j$ can be updated via

$$\boldsymbol{z}_j^{k+1} = \underset{\boldsymbol{z}_j}{\operatorname{argmin}} \left(\psi_j(\boldsymbol{z}_j) + \frac{1}{2} \left\| \mathbf{W}_j(\mathbf{D}_j \boldsymbol{q}^{k+1} - \boldsymbol{z}_j + \bar{\boldsymbol{u}}_j^k) \right\|^2 \right), \quad (4.73)$$

$$\bar{\boldsymbol{u}}_j^{k+1} = \bar{\boldsymbol{u}}_j^k + \mathbf{D}_j \boldsymbol{q}^{k+1} - \boldsymbol{z}_j^{k+1}. \quad (4.74)$$

While the $\boldsymbol{q}$- and $\bar{\boldsymbol{u}}$-updates are trivial to perform, a solver for the optimization problem in Equation 4.73 needs to be provided for each energy term ψ_j. This optimization problem is non-linear and does not have an analytical solution in the general case. Still, since the optimization problem is low dimensional for typical energy terms it can usually be solved at low computational cost via numerical optimization. An overview over implicit Euler integration via ADMM is given in Algorithm 9.

4.4.2 Hard Constraints

Hard constraints can be incorporated into the ADMM solver in a straight-forward manner. Given a constraint manifold $\boldsymbol{C}_j$, we construct the energy term

$$\psi_j(\boldsymbol{z}_j) = \delta_{\boldsymbol{C}_j}(\boldsymbol{z}_j) = \begin{cases} 0, & \text{if } \mathbf{z}_j \text{ lies on the constraint manifold } \boldsymbol{C}_j \\ \infty, & \text{otherwise,} \end{cases} \quad (4.75)$$

where $\delta_{\boldsymbol{C}_j}$ is the indicator function for $\boldsymbol{C}_j$ (see Eq. 4.37). Then, the $\boldsymbol{z}_j$-update is as simple as projecting $\mathbf{D}_j \boldsymbol{q}^{k+1} + \bar{\boldsymbol{u}}_j^k$ onto the constraint manifold $\boldsymbol{C}_j$, i.e.

$$\boldsymbol{z}_j^{k+1} = \text{proj}_{\boldsymbol{C}_j}(\mathbf{D}_j \boldsymbol{q}^{k+1} + \boldsymbol{u}_j^k).$$

Algorithm 9 Implicit Euler Integration via ADMM

procedure SOLVEADMM($\boldsymbol{q}_n$, $\boldsymbol{v}_n$, $\boldsymbol{f}_{\text{ext}}$, h)
  $\tilde{\boldsymbol{q}} = \boldsymbol{q}_n + h\boldsymbol{v}_n + h^2\mathbf{M}^{-1}\boldsymbol{f}_{\text{ext}}$
  $\boldsymbol{q} = \tilde{\boldsymbol{q}}$, $\bar{\boldsymbol{u}} = \mathbf{0}$
  for all iterations **do**
    for energy terms j **do**
      Compute $\mathbf{D}_j\boldsymbol{q} + \bar{\boldsymbol{u}}_j$
      Update $\boldsymbol{z}_j$ using Equation 4.73
      Update $\bar{\boldsymbol{u}}_j$ using Equation 4.74
    end for
    Update $\boldsymbol{q}$ using Equation 4.70
  end for
  return with $\mathbf{q}_{n+1} = \mathbf{q}, \mathbf{v}_{n+1} = (\mathbf{q}_{n+1} - \mathbf{q}_n)/h$
end procedure

4.4.3 Properties of ADMM

The ADMM solver approaches implicit Euler integration of the equations of motion through the lense of constrained optimization. In the following, we provide a detailed discussion of its properties.

Relation to PD. It can be shown that ADMM with weights $w_j = \sqrt{k_j}$ applied to PD energy potentials ψ_j (see Eq. 4.43) with stiffness k_j and constraint manifold $\boldsymbol{C}_j$ is equivalent to PD if $\boldsymbol{C}_j$ is affine. For a detailed proof of this statement, we refer the reader to the paper by Overby et al. [OBLN17]. Note that the proof does not apply for the strain material model (see Sec. 3.6.1), since the constraint manifolds used in the corresponding PD energy potentials are not affine (see Sec. 4.2.5). Still, the authors provide experimental data that shows that ADMM and PD exhibit almost identical behavior in this setting. As such, the ADMM solver can be interpreted as an extension of the PD solver that allows handling hard constraints and modelling arbitrary conservative energy terms.

Efficient Implementation. As pointed out in Section 4.4.1, the system matrix in the $\boldsymbol{q}$-update (see Eq. 4.70) is constant across the entire simulation if the constraint set remains unchanged. Thus, its matrix factorization can be reused during all $\boldsymbol{q}$-updates, allowing for the LSE in Equation 4.70 to be solved efficiently. Additionally, the $\boldsymbol{z}_j$- and $\bar{\boldsymbol{u}}$-updates can be computed independently for each energy term, enabling performance gains via parallelization. In many cases, the $\boldsymbol{z}_j$-updates do not have an analytical solution and may need to be determined numerically. However, the corresponding optimization problem is low dimensional

and can usually be computed at moderate computational cost. Either way, the impact of a single $\boldsymbol{z}_j$-update on the runtime of the ADMM solver becomes negligible with increasing hardware parallelism [OBLN17].

Generality. The ADMM solver can be applied to hyperelastic energy terms $\psi_j(\boldsymbol{q})$ for which the optimization problem in the $\boldsymbol{z}_j$-update can be solved. While Overby et al. [OBLN17] provide a closed-form solution for the $\boldsymbol{z}_j$-update for PD energy potentials, the $\boldsymbol{z}_j$-update for (simplified) Neohookean energy terms needs to be computed numerically. The ADMM solver is capable of modelling arbitrary isotropic and anisotropic hyperelastic materials. This is in contrast to the QN solver, where anisotropy can only be modelled by adding an anisotropic stiffness term to Valanis-Landel energies [LBK17]. Additionally, we demonstrate that the ADMM solver is compatible with energy terms that are not differentiable in Section 4.4.2. This enables implementing hard constraints with the ADMM solver.

Convergence. ADMM is known to converge to the optimal solution of the constrained minimization problem in Equation 4.67 with a convergence rate of $\mathcal{O}(1/n)$ for convex functions f and g under mild assumptions [OBLN17]. However, since the energy potentials of the material models used in physical simulations are generally not convex and $g(\boldsymbol{z}) = \Psi_*(\boldsymbol{z})$, these results may not apply directly to the ADMM solver. Thus, the performance of the ADMM solver is expected to deteriorate for energy terms that are characterized by large degrees of non-convexity. Note that even for convex f, g the convergence rate of $\mathcal{O}(1/n)$ is much lower than the superlinear convergence rate observed for many Quasi-Newton methods [NW06].

Weights. The convergence rate of the ADMM solver depends heavily on the choice of appropriate weights w_j for energy terms ψ_j [OBLN17]. To the best of our knowledge, no method for determining the choice of w_j for reliably fast convergence is available. Overby et al. [OBLN17] provide experimental data that suggests that reducing w_j can have beneficial effects on the convergence rate of ADMM, but that the algorithm may fail to converge if the weights are reduced too much. The first observation can be explained by taking a closer look at the $\boldsymbol{z}_j$-update, which is restated here again for the convenience of the reader

$$\boldsymbol{z}_j^{k+1} = \underset{\boldsymbol{z}_j}{\operatorname{argmin}} \left(\psi_j(\boldsymbol{z}_j) + \frac{1}{2} \left\| \mathbf{W}_j (\mathbf{D}_j \boldsymbol{q}^{k+1} - \boldsymbol{z}_j + \bar{\boldsymbol{u}}_j^k) \right\|^2 \right). \tag{4.76}$$

According to Overby et al. [OBLN17], the $\boldsymbol{z}_j$-update can be interpreted as a proximal operator for ψ_j and intuitively amounts to minimizing ψ_j with a quadratic

penalty for moving away from $\mathbf{D}_j \boldsymbol{x}^{n+1} + \bar{\boldsymbol{u}}_n^i$. This penalty increases with the weights that make up the entries of $\mathbf{W}_j$. Thus, reducing the weights allows making faster progress towards a solution that minimizes the constraint during the $\boldsymbol{z}_j$-update. Keeping in mind that $\mathbf{D}_j \boldsymbol{q}^* = \boldsymbol{z}_j^*$ for all constraints C_j once ADMM converges, it is natural to assume that using smaller weights is particularly helpful when the converged positions $\boldsymbol{q}^*$ reduce the total constraint energy Ψ significantly in comparison to the initial positions $\boldsymbol{q}_0$. However, if $\Psi(\boldsymbol{q}^*) \approx \Psi(\boldsymbol{q}_0)$, it might be counterproductive to take large steps towards a solution that minimizes the constraint energies during the $\boldsymbol{z}_j$-update. This effect might also be responsible for ADMM's failure to converge if inappropriately small weights are used.

Changing Constraint Sets. Similar to the PD and QN solvers, the ADMM solver is not suited for handling frequently changing constraint sets. This is because efficient implementation of the ADMM solver exploits the fact that the system matrix of the $\boldsymbol{q}$-update is constant, allowing for its factorization to be reused. However, Overby et al. [OBLN17] point out that collisions with static geometry can be handled efficiently using hard constraints. There, a single collision energy term ψ_{coll} with reduction matrix $\mathbf{D}_{\text{coll}} = \mathbf{I} \in \mathbb{R}^{3m \times 3m}$ is added to the system. During the $\boldsymbol{z}$-update, particles are simply projected to their closest non-penetrating positions.

Hard Constraints. In Section 4.4.2, we demonstrated how to model hard constraints using the ADMM solver by introducing suitable energy terms ψ_j. Overby et al. [OBLN17] point out that while the $\boldsymbol{z}_j$ satisfy their constraints at all times, the same may not be true for the particle positions $\boldsymbol{q}$ before convergence is achieved. This is particularly important in the context of real-time applications, where the number of solver iterations is often capped to ensure that the physics simulations fit into the time budget of a single frame.

Chapter 5

Evaluation

The theoretical analysis of the solvers provided in Chapter 4 shows that XPBD and PD-style solvers pursue fundamentally different approaches to performing implicit Euler integration on the equations of motion. XPBD employs a local Gauss-Seidel-type solver that independently minimizes energies one-by-one by performing corresponding constraint projections. On the other hand, PD-style solvers treat implicit integration as an optimization problem and attempt to minimize the objective function of the variational form of implicit Euler integration (see Sec. 3.3). In contrast to XPBD, all PD-style solvers contain a global optimization step that enables making compromises during the minimization of multiple competing energies or constraints. The goal of this section is to provide experimental data that highlights how the differences between the solvers manifest themselves in practice during the simulation of deformable three-dimensional bodies. Starting from experiments using the simple strain material model, we extend our analysis to the more complex Neohookean material model and incompatible constraint sets. Here, we say that a set of constraints is incompatible if there are no particle positions $\boldsymbol{q}$ that simultaneously minimize all constraints from the set.

The main tool for comparing solvers used in this report consists of setting up identical simulation states and analyzing the positions computed by the solvers during each of the iterations required for the simulation of a single time step. We provide a detailed description of the simulated scenarios in Section 5.1 and discuss how insights on various solver properties can be gained from the positions achieved after each iteration in Section 5.2. During our comparison of the solvers, we exclude PBD due to the fact that it is not derived from the equations of motion and does not produce physically accurate results (see Sec. 4.1.3). We provide experimental data to justify this decision in Section 5.3. Additionally, before a meaningful comparison of ADMM with other solvers is possible, suitable ADMM weights need to be determined (see Sec. 4.4). We provide insights

on choosing ADMM weights in Section 5.4. We start our comparison between XPBD and PD-style solvers on simulations using only tetrahedral constraints for the strain material model (see Sec. 3.6.1) in Section 5.5. The effects of adding position constraints that are incompatible with the material constraints are analyzed in Section 5.6. The experiments in Section 5.5 are repeated for the simplified Neohookean material model (see Sec. 3.6.3) in Section 5.7 to gain insights on the compatibility of the solvers with more complicated non-linear material models. Finally, we summarize our findings in Section 5.8.

5.1 Experimental Setup

Experiments start from the deformed configuration of a cuboid beam that has been twisted in opposite directions at both ends (see Fig. 5.1, left). This initial deformed configuration is determined by simulating the twisting motion of the beam using the QN solver with a large number of iterations. Twisting is achieved by in-plane rotation of the reference positions of position constraints acting on particles at the ends of the beam.

Starting from this configuration, we investigate the behavior of the solvers during the next time step in two settings: In the first, the position constraints at the ends of the beam are released, causing the beam to untwist (see Fig. 5.1, top right). In the second, the beam is twisted further in accordance with a fixed angular velocity as described above (see Fig. 5.1, bottom right). In both settings, the initial positions of the twisted beam need to be updated to adapt to the changes to the position constraints at the ends of the beam. In the untwisting beam experiments, all constraints are satisfied at the undeformed reference configuration of the beam $\boldsymbol{q}_{\text{ref}}$. However, the position constraints causing the twisting motion in the twisting beam experiments are incompatible with the tetrahedral constraints derived from the material model of interest.

Position constraints are modelled via springs with rest-length zero and finite stiffness for all solvers. In other words, only soft position constraints are used. This has two main advantages: Firstly, soft position constraints can easily be modelled using each of the solvers, while it is challenging (or impossible) to implement hard position constraints with PD and QN. Mixing soft and hard position constraints makes it impossible to compare solvers, since they would converge towards distinct solutions. Secondly, soft position constraints have corresponding spring energies that are finite and continuously differentiable. This allows for the computation of common metrics of deformed configurations such as the constraint energy.

The elastic properties of the beam are modelled via the strain material model and the simplified Neohookean material model (see Sec. 3.6). The strain model

is supported by all solvers, whereas the simplified Neohookean model is not supported by PD. The simplified Neohookean model is chosen over the original Neohookean model because the log term in the original Neohookean energy potential makes it incompatible with XPBD (see Sec. 4.1.7). The experiments are run for time steps $1 \cdot 10^{-1}$ s, $1 \cdot 10^{-2}$ s, $1 \cdot 10^{-3}$ s and $1 \cdot 10^{-4}$ s. For the strain and Neohookean constraints, both stiffness and Youngs modulus, respectively, are set to powers of ten ranging from $1 \cdot 10^{5}$ to $1 \cdot 10^{12}$. The Poisson ratio is fixed at 0.45 for all experiments, since we put the focus of our evalution on material stiffness over incompressibility. A representative subset of both twisting and untwisting beam experiments using the strain material model are discussed in Sections 5.5 and 5.6, respectively. For the Neohookean material model, we focus on the untwisting beam experiment in order to isolate the effects caused by applying solvers to a more complicated non-linear material model from artifacts caused by incompatible constraints. A representative set of experiments is discussed in Section 5.7.

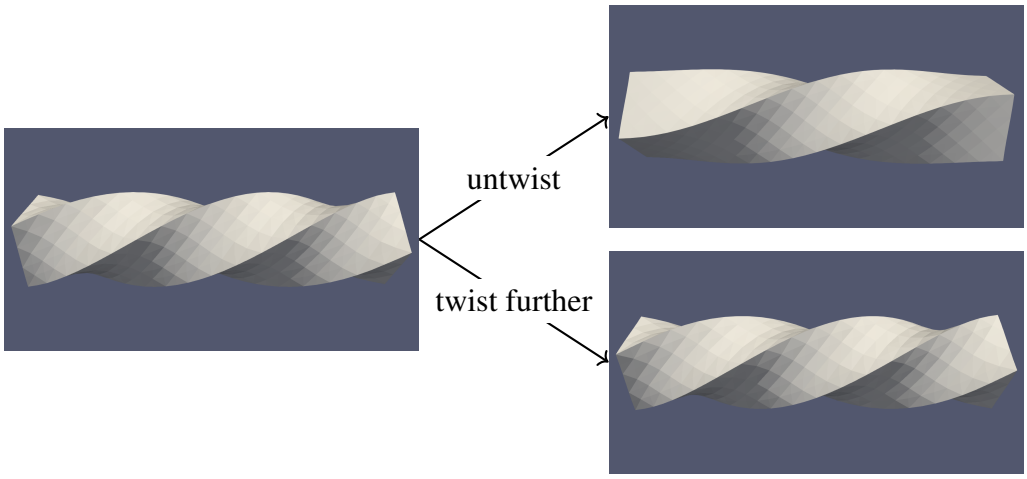


Figure 5.1: Visualization of the experimental settings starting from a twisted beam. The beam is either untwisted by removing position constraints at the ends of the beam (top) or twisted further by rotating the anchor positions of the position constraints in-plane (bottom).

5.2 Analyzing Solver Properties

In many cases, poor solver performance is immediately obvious from closer inspection of the computed geometry. However, sometimes solvers can produce geometries that pass the eye test, but are in violation of the equations of motion. This is particularly applicable to small time step sizes. Thus, additional metrics

are required to reliably evaluate solver performance across a variety of different time step sizes and stiffnesses.

Objective Function of the Variational Form of Implicit Euler Integration. With the exception of PBD, all solvers discussed in the context of this thesis are based on implicit Euler integration of the equations of motion (see Sec. 3.2) and aim to either approximate (XPBD) or compute exactly (PD, ADMM and QN) the implicit positions at the next time step. As discussed in Section 3.2, the implicit positions can be interpreted as the positions that minimize the objective function of the variational form of implicit Euler integration given in Equation 3.7. To gain initial insights into the convergence behavior of the solvers, we compute the objective function value, or simply objective value for short, for the positions obtained after each iteration for each solver. The objective is restated here for the convenience of the reader

$$f_{\text{obj}}(\boldsymbol{q}) \coloneqq \min_{\boldsymbol{q}} \frac{1}{2h^2} \left\| \mathbf{M}^{\frac{1}{2}}(\boldsymbol{q} - \tilde{\boldsymbol{q}}) \right\|_F^2 + \sum_j \psi_j(\boldsymbol{q}). \tag{5.1}$$

The solvers are expected to decrease the objective with each iteration until a value close to the minimum obtained by the implicit positions is achieved. By comparing the objective values after the final iteration, the quality of the configurations different solvers converge to can be analyzed. Further, by plotting the objective values over the iteration number and inspecting the slope of the graphs, information about the convergence rates of the solvers can be inferred. During the discussion of the resulting figures, we consider a solver converged once the graph of its objective function value over the iterations is visually indistinguishable from a horizontal line. While not entirely rigorous, this is particularly appropriate for comparing solvers in the context of real-time applications where solver iterations that only make minor progress towards the true implicit solutions often cannot be afforded. Additionally, this allows setting the focus on the most important qualitative differences between solvers without getting lost in the details of when a solver can formally be considered converged.

As discussed in Section 3.3, the constraint term of the objective function corresponds to the sum of the constraint energies, while the inertial term introduces a penalty for moving particles away from their inertial positions $\tilde{\boldsymbol{q}}$. However, in the context of the untwisting and twisting beam experiments described in Section 5.1, the inertial term can also be interpreted as the kinetic energy of the beam. To see this, note that both the initial particle velocities $\boldsymbol{v}_0$ and the external forces $\boldsymbol{f}_{\text{ext}}$ acting on the simulated body are zero. Consequently, the inertial positions $\tilde{\boldsymbol{q}}$ are equivalent to the initial positions $\boldsymbol{q}_0$, as shown below

$$\tilde{\boldsymbol{q}} = \boldsymbol{q}_0 + h\boldsymbol{v}_0 + h^2\mathbf{M}^{-1}\boldsymbol{f}_{\text{ext}} = \boldsymbol{q}_0.$$

It follows that the inertial term at the positions achieved after k solver iterations $\boldsymbol{q}_k$ can be rewritten as

$$\frac{1}{2h^2}\left\|\mathbf{M}^{\frac{1}{2}}(\boldsymbol{q}_k - \boldsymbol{q}_0)\right\|_F^2 = \frac{1}{2}\left\|\mathbf{M}^{\frac{1}{2}}\frac{\boldsymbol{q}_k - \boldsymbol{q}_0}{h}\right\|_F^2 = \frac{1}{2}\left\|\mathbf{M}^{\frac{1}{2}}\boldsymbol{v}_k\right\|_F^2,$$

where $\boldsymbol{v}_k$ are the particle velocities after k iterations as computed via Verlet integration. The last term in the equation above is exactly the kinetic energy of the simulated body. As the sum of the elastic and kinetic energy, the objective function can be interpreted as the energy of the beam. According to Newton's laws, this energy should be conserved in the absence of external forces. However, implicit Euler integration typically introduces numerical damping which is expected to reduce the energy of the beam over time (see Sec. 3.2).

Linear and Angular Momentum. Since adding artificial linear and angular momentum to simulations manifests itself in ghost forces which act like external forces dragging and rotating the object, conserving global momenta is a hard requirement for achieving visually plausible simulations [MHHR06]. If the elastic forces in the equations of motion are derived from rigid motion invariant energy potentials, the true positions resulting from implicit Euler integration conserve linear and angular momentum [BML+14]. However, there are many situations where we cannot expect solvers to converge to the true implicit positions. As an example, we have shown that XPBD cannot be said to solve the equations of motions exactly due to simplifying assumptions made during the derivation of the XPBD constraint projection (see Sec. 4.1.6). Additionally, in the context of real-time applications, the number of solver iterations is often fixed to ensure that the physics simulation fits into the allotted time budget. In such situations, there is no guarantee that solvers converge in the available number of iterations. Thus, it is important to analyze whether solvers conserve global momenta at both the converged configuration and the intermediate configurations achieved after each solver iteration. The computation of the linear and angular momentum after solver iteration k requires the computation of velocities $\boldsymbol{v}_k$ from the current positions $\boldsymbol{q}_k$. In our experiments, we compute velocities $\boldsymbol{v}_k$ by performing a Verlet integration step given by

$$\boldsymbol{v}_k = \frac{\boldsymbol{q}_k - \boldsymbol{q}_0}{h}, \tag{5.2}$$

where h is the time step size. Now, let i denote a particle index and let $\boldsymbol{q}$ and $\boldsymbol{v}$ denote the current particle positions and velocities, respectively. Then, the global linear momentum is equal to

$$\sum_i m_i \boldsymbol{v}_i, \tag{5.3}$$

and the angular momentum with respect to some point $\boldsymbol{p} \in$ is given by

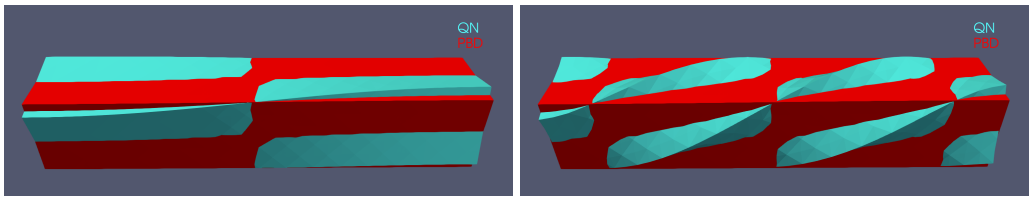
$$\sum_i (\boldsymbol{q}_i - \boldsymbol{p}) \times (m_i \boldsymbol{v}_i). \tag{5.4}$$

5.3 Physically-Based PBD

In contrast to XPBD and the PD-style solvers, PBD is not derived from the implicit integration of the equations of motion. As a result, the stiffness of materials simulated using PBD depends on the number of iterations and the chosen time step (see Sec. 4.1.3). Accordingly, in the limit of infinite iterations and zero time step size, the material becomes infinitely stiff. We demonstrate this effect by example of a single solve over time steps $1 \cdot 10^{-2}\,\mathrm{s}$ and $1 \cdot 10^{-3}\,\mathrm{s}$ in the untwisting beam setting (see Fig. 5.1) using strain constraints with stiffness $1 \cdot 10^{8}$ in Figure 5.2. The experiment shows that the beam is untwisted entirely to its undeformed reference configuration when using the PBD solver, regardless of the chosen time step. On the other hand, the degree to which the beam is untwisted depends on the time step size when using the QN solver. In particular, the beam untwists further when the larger time step of $1 \cdot 10^{-2}\,\mathrm{s}$ is used. Of course, the behavior of the PBD solver is not physically plausible. This observation and the fact that PBD can seamlessly be extended to XPBD at the cost of storing a single additional variable per constraint leads us to the decision to exclude PBD from the remaining experiments in this thesis and to focus on analyzing the properties of XPBD, ADMM, PD and the QN solver instead.

5.4 ADMM Weights

As stated in Section 4.4.3, there is no generally valid method for determining ADMM weights that ensures acceptable convergence properties. Thus, we need to manually fine-tune ADMM weights before comparing different solvers using the method described above. The convergence of the ADMM solver is evaluated via the objective function of the beam (see Sec. 5.2). We focus on untwisting beam experiments using strain constraints. For each combination of time steps from $1 \cdot 10^{-1}$ to $1 \cdot 10^{-4}$ and stiffnesses k from $1 \cdot 10^{5}$ to $1 \cdot 10^{12}$, candidate weights



(a) time step $1 \cdot 10^{-2}$ s **(b)** time step $1 \cdot 10^{-3}$ s

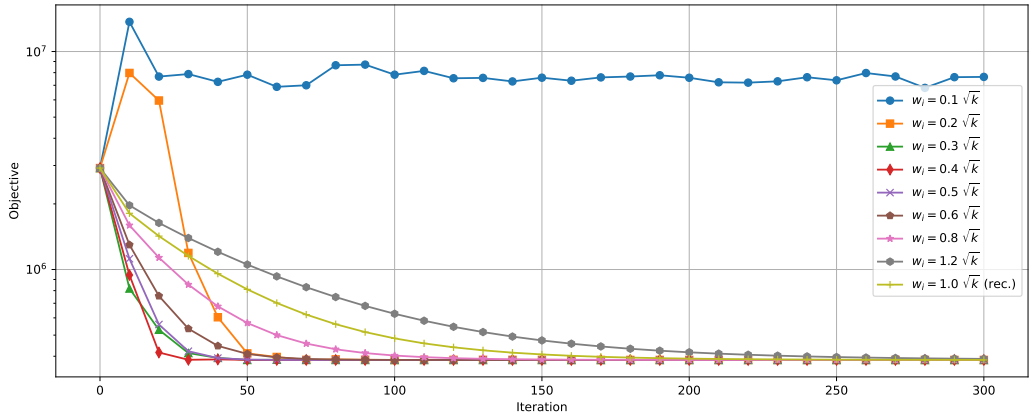
Figure 5.2: Geometries of an untwisting beam with strain constraints (stiffness $1 \cdot 10^8$) after 1000 iterations over a single time step of size $1 \cdot 10^{-2}$ s (left) and $1 \cdot 10^{-3}$ s (right) using PBD (red) and QN (cyan).

around the recommended value $w_i = \sqrt{k}$ are generated and the weight that reduces the objective function the fastest is used as the ADMM weight for all following experiments. This process reveals that ADMM weights are in fact time step dependent.

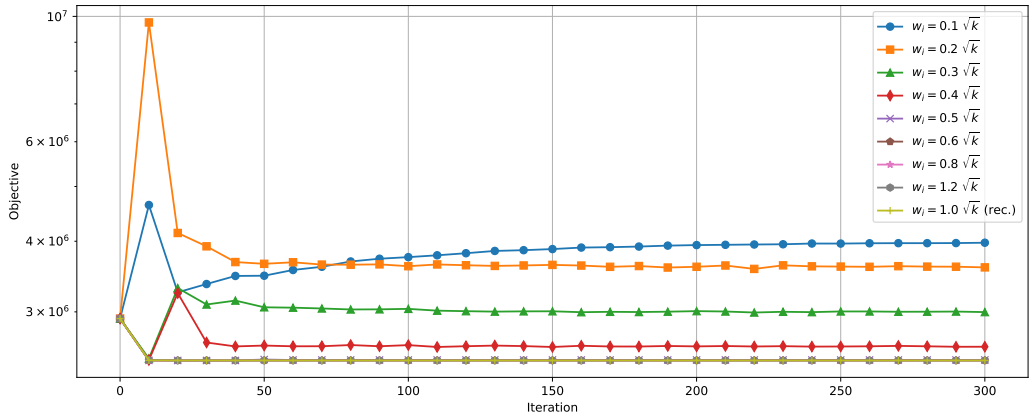
To see this, consider the plots of the objective values over the number of ADMM iterations with different weights for time steps $1 \cdot 10^{-2}$ s and $1 \cdot 10^{-3}$ s with constraint stiffness $1 \cdot 10^8$ shown in Figure 5.3. When a time step of $1 \cdot 10^{-2}$ s is used (see Fig. 5.3 a), the ADMM solver converges to the same objective value for all of the weights, except for the lowest weight $w_i = 0.1\sqrt{k}$, where the value is increased compared to the initial objective value. The fastest convergence is achieved by setting $w_i = 0.3\sqrt{k}$ and $w_i = 0.4\sqrt{k}$. When picking smaller weights, the objective value is larger than the initial objective value during the first 20 iterations. Setting the weights to higher values than $w_i = 0.4\sqrt{k}$ leads to a slower convergence rate. However, if the time step is decreased to $1 \cdot 10^{-3}$ s (see Fig. 5.3 b), weights higher than $w_i = 0.4\sqrt{k}$ achieve a better convergence rate and converge to configurations with a lower objective value than weights lower than or equal to $w_i = 0.4\sqrt{k}$. In particular, ADMM converges to a solution whose objective value is higher than the initial objective value when using $w_i = 0.2\sqrt{k}$ and $w_i = 0.3\sqrt{k}$. Thus, $w_i = 0.3\sqrt{k}$ is an excellent weight for time step $1 \cdot 10^{-2}$ s, while it is unsuitable for time step $1 \cdot 10^{-3}$ s. Overall, it can be seen that more aggressive reductions of the ADMM weights without jeopardizing convergence are possible when a larger time step is used. Qualitatively equivalent results can be observed for other stiffness values.

In Section 4.4.3, we hypothesized that it might be preferrable to use smaller ADMM weights if the constraint energy Ψ is significantly smaller at the true implicit positions $\boldsymbol{q}^*$ than at the initial positions $\boldsymbol{q}_0$, whereas larger weights might be more appropriate when $\Psi(\boldsymbol{q}^*) \approx \Psi(\boldsymbol{q}_0)$. The time step dependence of ADMM weights can be explained by combining this insight with the fact that the extent to which a single application of implicit Euler integration reduces the constraint en-

ergy of a deformed body generally grows with increasing time step size. Thus, for smaller time steps it is $\Psi(\boldsymbol{q}^*) \approx \Psi(\boldsymbol{q}_0)$, favoring the use of larger ADMM weights. Note that this time step dependence makes finding suitable ADMM weights incredibly tedious: For each material model, weights need to be manually fine-tuned for each combination of time step size and material stiffness. To save time, we reuse the ADMM weights determined for the untwisting beam experiments (see Sec. 5.5) in the twisting beam experiments (see Sec. 5.6). However, it is not clear whether ADMM weights can be reused across different simulation scenarios, even if the time step size and the material stiffness remain unchanged.



(a) Time step $1 \cdot 10^{-2}$ s



(b) Time step $1 \cdot 10^{-3}$ s

Figure 5.3: Values of the objective function of the variational form of implicit Euler integration (see Eq. 3.7) over the iterations of a single time step of size $1 \cdot 10^{-2}$ s (a) and $1 \cdot 10^{-3}$ s (b) for an untwisting beam with strain constraints with stiffness $1 \cdot 10^{8}$ using ADMM with different weights.

5.5 Untwisting Beam Simulations - Strain Material

In the untwisting beam experiments using the strain material model, challenges due to incompatible constraint sets are avoided and the tetrahedral strain constraints are easy to minimize due to the fact that corresponding restorative elastic forces are proportional to the distance between the deformation and its closest rigid-body transform (see Sec. 3.6.1). Thus, we use them as a starting point for highlighting differences between the different solvers. We start by comparing the convergence properties of different solvers in Section 5.5.1. Experimental data that suggests that solvers convert elastic energy stored in the deformed beam into kinetic energy is provided in Section 5.5.2. The effects of terminating different solvers after a fixed number of solver iterations are analyzed in Section 5.5.3. In Section 5.5.4, the computational costs of the solvers are compared. Finally, we investigate the behavior of the solvers when larger time step sizes and stiffness values are used in Sections 5.5.5 and 5.5.6.

5.5.1 Convergence Properties

We start by analyzing the convergence properties of different solvers with time step size $1 \cdot 10^2$ s and stiffness $1 \cdot 10^8$ by plotting the objective values over the iteration number in Figure 5.4. Note that the results are representative for a large range of time step sizes and stiffness values. The graph shows that each of the solvers indeed mostly decreases the objective from iteration to iteration until it converges towards a final configuration after a sufficient number of iterations. ADMM, PD and QN converge to configurations with roughly the same objective value whereas the objective value obtained after 1000 XPBD iterations is still slightly larger. QN and ADMM converge after roughly 40 and 60 iterations, whereas PD and XPBD converge after upwards of 260 and 1000 iterations, respectively. To quantify the progress that other solvers achieve during the number of iterations it takes each of the solvers to converge, we compute relative errors after 40 (QN), 60 (ADMM), 260 (PD) and 1000 (XPBD) iterations. The relative error is given by

$$\frac{f_{\text{obj}}(\boldsymbol{q}_k) - f_{\text{obj}}(\boldsymbol{q}^*)}{f_{\text{obj}}(\boldsymbol{q}_0) - f_{\text{obj}}(\boldsymbol{q}^*)}, \tag{5.5}$$

where $\boldsymbol{q}_k$ are the positions after iteration k and $\boldsymbol{q}^*$ are the positions with minimal objective value achieved by any of the solvers. The relative errors are shown in Table 5.1. Note that the relative error of ADMM is only 0.4 % by the time QN has converged, while the errors achieved by PD and XPBD are still substantial. Additionally, even after a 1000 iterations, XPBD's relative error is still at 1.8 %.

The data suggests that ADMM, PD and QN converge towards a solution that is closer to the true implicit positions at the next time step than XPBD. This is ex-

	QN	ADMM	PD	XPBD
40 iterations (QN)	0.0	0.4	22.7	67.8
60 iterations (ADMM)	0.0	0.0	12.6	47.8
260 iterations (PD)	0.0	0.0	0.0	4.0
1000 iterations (XPBD)	0.0	0.0	0.0	1.8

Table 5.1: Relative errors (see Eq. 5.5) in % achieved by solvers after 40, 60, 260 and 1000 iterations for the untwisting beam experiment from Figure 5.4.

pected, as XPBD only approximates a solution to the implicit equations of motion due to the simplifying assumptions made during its derivation (see Sec. 4.1.6). Due to the second assumption (see Eq. 4.21) in particular, there is no penalty for moving vertex positions away from their inertial positions during the XPBD constraint projections (see Sec. 4.1.6), even though it increases the objective function. Additionally, it is not clear whether the Gauss-Seidel solver at the core of XPBD is able to minimize the constraint energies sufficiently via local constraint projections only. Since constraints are projected independently one-by-one, later constraint projections can undo some of the work achieved by previous projections. Bouaziz et al. [BML+14] describe this phenomenon as an oscillation between different constraints. On the other hand ADMM, PD and QN aim to solve the implicit equations of motion exactly. The difference in the rate of convergence between ADMM and PD goes in line with experimental data provided by Overby et al. [OBLN17]. There, the authors show that ADMM with a weight of $\sqrt{k}$, where k is the stiffness of the strain constraints, is almost equivalent to PD and that the convergence rate can often be improved by reducing the weight. As discussed in Section 4.4.3, this is due to the fact that lower weights allow faster progress towards positions that reduce the constraint energy, which is particularly useful for simulations with larger timesteps. Similarly, the faster convergence rate of QN compared to PD is expected in light of the observation that PD is a quasi-Newton method with constant Hessian approximation (see [LBK17], Sec. 4.2.6). The QN solver augments the PD solver with a line-search algorithm and LBFGS-updates to the constant Hessian approximation from PD that take into account curvature information from previous iterations to better approximate the true Hessian matrix. Both of these measures help improve the convergence rate of QN in comparison to PD. Again, this is in line with results presented by Liu et al. [LBK17]. The comparably slow convergence rate of XPBD could again be explained by the simplifying assumptions made during the XPBD derivation and the aforementioned oscillations of Gauss-Seidel solvers.

To better understand the main factor for XPBD's unfavorable convergence properties, it is helpful to plot the inertial term and the constraint term of the ob-

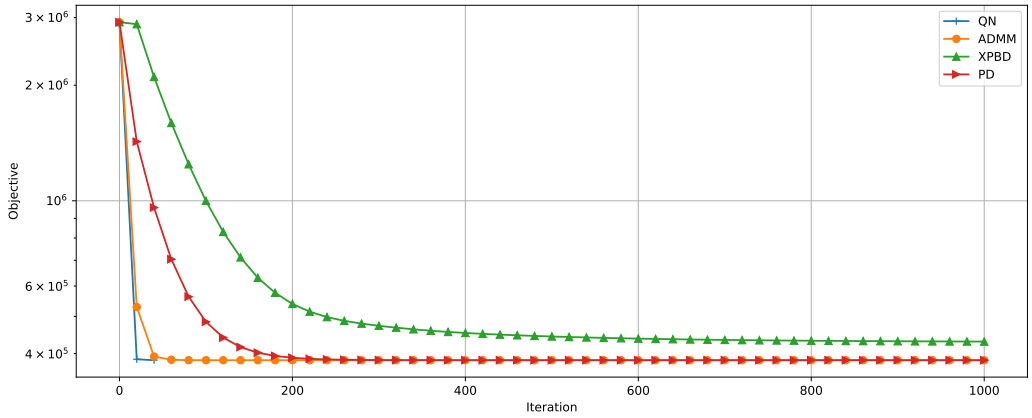


Figure 5.4: Values of the objective function of the variational form of implicit Euler integration (see Eq. 3.7) over the iterations of a single time step of size $1 \cdot 10^{-2}\,\mathrm{s}$ for an untwisting beam using strain constraints with stiffness $1 \cdot 10^{8}$.

jective function separately. The resulting plots are shown in Figure 5.5. We focus on the first 500 iterations to gain clearer insights into the behavior of the solver during earlier iterations, since these are more relevant for real-time applications. First, it is evident that the constraint energy dominates the objective value for each solver during most of the iterations before convergence is achieved. While each solver mostly decreases the constraint energy with an increasing number of iterations, the inertial term is mostly increased. The only exception is the ADMM solver between iterations 30 to 60, where the inertial term is decreased from its peak and the constraint energy is increased from its low point. ADMM, PD and QN converge to configurations with roughly the same inertial and constraint terms. However, at its final state, the constraint energy of the XPBD solver is higher than its counterparts from the PD-style solvers, whereas the opposite is observed for the inertial term.

As discussed in Section 3.3, the weighting between the inertial term and the constraint energy depends on the time step, the particle masses and the material stiffnesses. The results suggest that with a time step of $1 \cdot 10^{-2}\,\mathrm{s}$ and a constraint stiffness of $1 \cdot 10^{8}$, the constraint energy can be decreased significantly until the cost of moving particles away from their inertial positions becomes too high. Since the inertial term at the converged XPBD state is in fact lower than the inertial term achieved by PD-style solvers, it is evident that the reason for the higher objective value of XPBD is that the constraint energy is not reduced sufficiently. It is natural to assume that this is in part a consequence of the oscillations resulting from the iterative Gauss-Seidel solver used in XPBD. However, it is worth pointing out again that XPBD does not solve the implicit equations of motion exactly due to the simplifying XPBD assumptions. Even if Equation 4.22

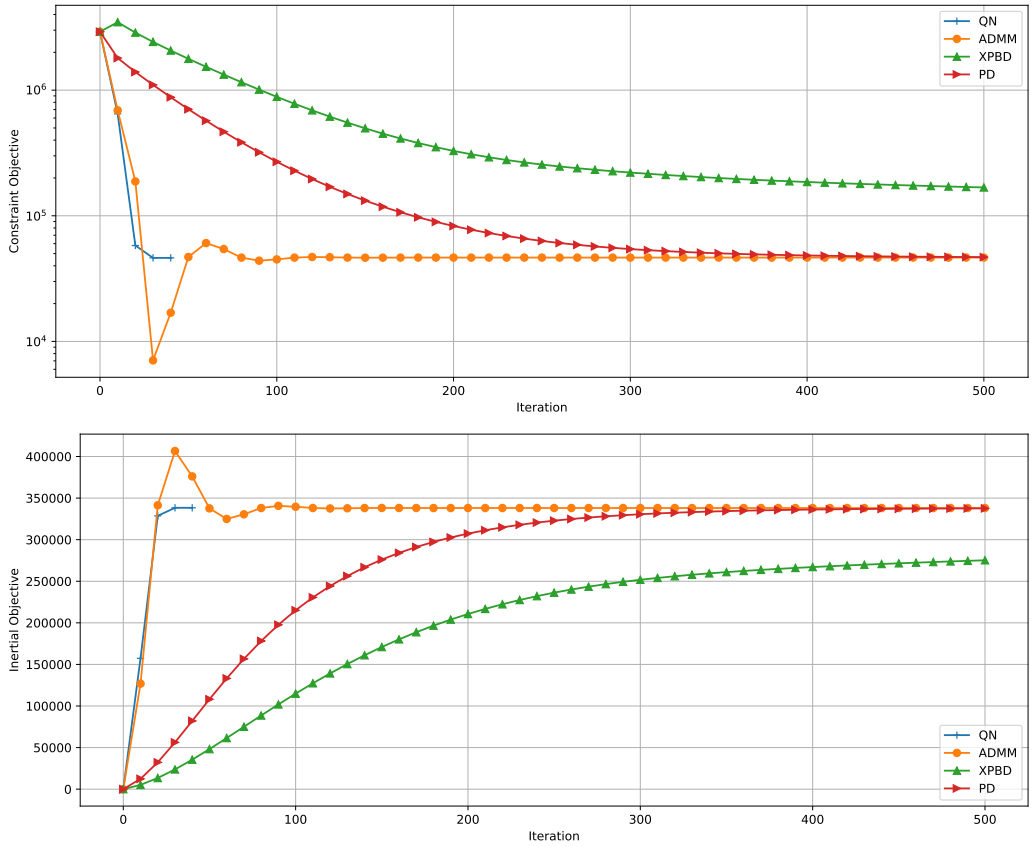


Figure 5.5: Values of the constraint (top) and inertial (bottom) terms of the objective function of the variational form of implicit Euler integration (see Eq. 3.7) over the iterations of single time step of size $1 \cdot 10^{-2}$ s for an untwisting beam with strain constraints with stiffness $1 \cdot 10^{8}$.

were to be solved with a global solver, it is not clear whether the constraint energy would be decreased further than it is using the Gauss-Seidel solver.

The fact that the constraint energy is higher for the final state of the XPBD solver is also perceptible when comparing its geometry with the final geometries computed by PD-style solvers. The geometries achieved after 1000 XPBD and QN iterations are shown in Figure 5.6 a. The corresponding geometries for PD and ADMM are ommitted since they are visually indistinguishable from the geometry computed by the QN solver. The image shows that the QN solver untwists the beam further towards its undeformed rest configuration than XPBD. Since the constraint energies obtain a minimum at the reference configuration, this is in line with the observation that QN is more successful than XPBD at reducing the constraint term of the objective function. We point out that the surface of the

XPBD geometry is smooth. In the discussion of Figure 5.6 b and Figure 5.13, we suggest that oscillations of the Gauss-Seidel-type solver manifest themselves in noticable geometric artifacts on the beam surface. Thus, it is likely that the comparably large constraint term after 1000 XPBD iterations is caused by the simplifying assumptions during the XPBD derivation instead of the local nature of the XPBD solver.

Considering that the inertial positions do not appear in the update equations of XPBD (see Sec. 4.1.6), it is surprising that the inertial term achieved by the converged state of the XPBD solver is smaller than the final inertial term of the other solvers. On the other hand, the only factor driving particles away from their inertial positions in the untwisting beam experiment are the elastic energies due to the deformation of the beam. One possible explanation for the low inertial term for XPBD might be that the particles do not move far away from their inertial positions simply because the XPBD constraint solver is not as successful at minimizing the constraint energies. Lastly, the penalty for moving particles from their inertial positions is quite low for PD-style solvers if a time step as large as $1 \cdot 10^{-2}\,\mathrm{s}$ is used. However, the inertial term achieved by XPBD remains smaller than its counterparts, even if the timestep is decreased.

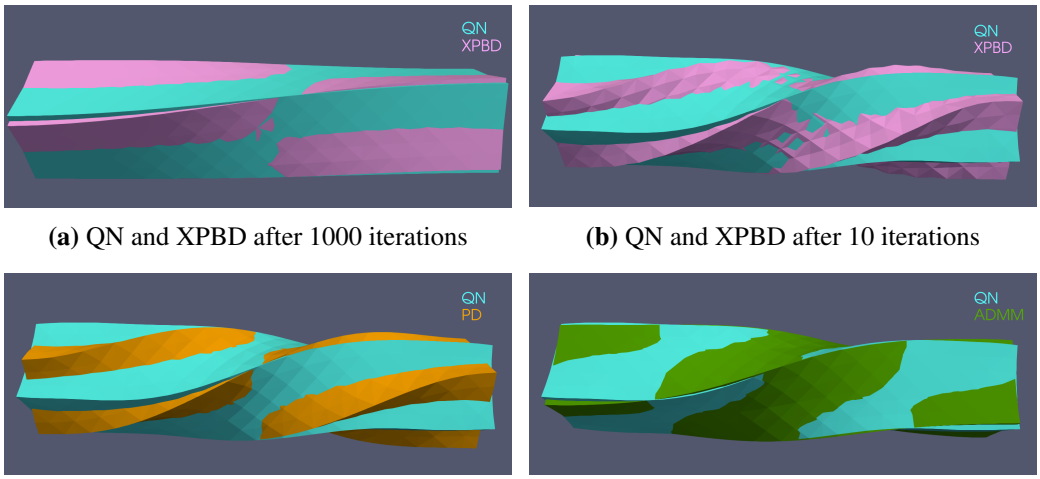
5.5.2 Conversion of Constraint Energy Into Kinetic Energy

In the light of the observation that the inertial term of the objective function of the variational form of implicit Euler integration can be interpreted as the kinetic energy of the beam (see Sec. 5.2), the results in Figure 5.5 suggest that some of the constraint energy stored in the deformed beam is converted into kinetic energy during most solver iterations. Additionally, solvers with a faster convergence rate appear to be able to convert a larger share of the elastic energy into kinetic energy during a single iteration.

5.5.3 Effects of Early Termination

In the context of real-time applications, the number of solver iterations is often capped to ensure that the computations fit into the time budget allocated for the simulation of the next frame. A closer look at Figure 5.5 shows that all of the solvers are still in the process of converting the constraint energy stored in the twisted beam into kinetic energy after the first 10 iterations. Accordingly, none of the solvers have reduced the objective function to the minimum achieved by the PD-style solvers yet (see Fig. 5.4). After 10 iterations, the QN solver has reduced the objective function the most, followed by ADMM, PD and XPBD.

To gain a better understanding of the effects caused by terminating solvers before the objective function is minimized, we take a closer look at the geome-



(a) QN and XPBD after 1000 iterations

(b) QN and XPBD after 10 iterations

(c) QN and PD after 10 iterations

(d) QN and ADMM after 10 iterations

Figure 5.6: Geometries of an untwisting beam with strain constraints (stiffness $1 \cdot 10^8$) after 10 and 1000 iterations over a single time step of size $1 \cdot 10^{-2}$ s using XPBD (pink), PD (orange), ADMM (green) and QN (cyan).

tries that the XPBD, PD and ADMM solvers arrive at after 10 solver iterations in Figure 5.6 b-d. The corresponding QN geometry is given as a reference in each panel. Figure 5.6 b-d shows that the solvers untwist the beam to different extents after the first 10 solver iterations. The QN geometry gets closest to the beam's reference configuration, again followed by ADMM, PD and, lastly, XPBD. Note that the degree to which solvers untwist the beam goes hand-in-hand with the extent to which they minimize the objective function and particularly its constraint energy. In summary, terminating the solvers after a small number of iterations can cause the motion of simulated bodies to appear slowed down. This effect is more noticable for solvers that have inferior convergence properties, such as PD and XPBD. Note that Bouaziz et al. [BML+14] report the same observation in the discussion of the PD solver and state that this is due to the fact that forces may not be able to fully propagate through the mesh if the optimization is not run long enough.

Closer inspection of Figure 5.6 b shows that the XPBD geometry is noticably rugged compared to the geometries of the PD-type solvers, which appear mostly smooth at the surface. Thus, in contrast to the PD-style solvers, terminating the simulation early does not only manifest itself in slowing down the motion of the simulated body but also in visible geometric artifacts. This matches the observation that the constraint objective value achieved after 10 XPBD iterations is larger than the initial constraint objective value in Figure 5.5. While it is difficult to verify, it is natural to assume that this effect is caused by oscillations of

the Gauss-Seidel solver at the core of XPBD, since it does not appear for solvers that have a global optimization step. Then, the fact that the XPBD geometry does appear smooth after 1000 iterations (see Fig. 5.6) would suggest that oscillations are particularly severe during earlier iterations where the constraint energies and their gradients are large.

5.5.4 Computational Cost

To take into account the computational cost of the iterations for different solvers, the objective values are plotted again over the accumulated iteration durations in Figure 5.7. However, the provided iteration durations should be treated as a rough indication of the computational costs of different solvers since no effort has been put into profiling or speeding up the computations by providing a multithreaded implementation. The computational costs of XPBD, ADMM and PD seem to very similar on average. 1000 iterations of XPBD, ADMM and PD take roughly 4 seconds each. On the other hand QN iterations appear to be almost twice as expensive as the iterations of the other solvers. Still, the conclusions that can be drawn from Figure 5.7 are similar to what was discussed for Figure 5.4: ADMM and QN converge in a shorter amount of time than XPBD and PD, but ADMM and PD appear in a more favorable light compared to QN due to faster iterations.

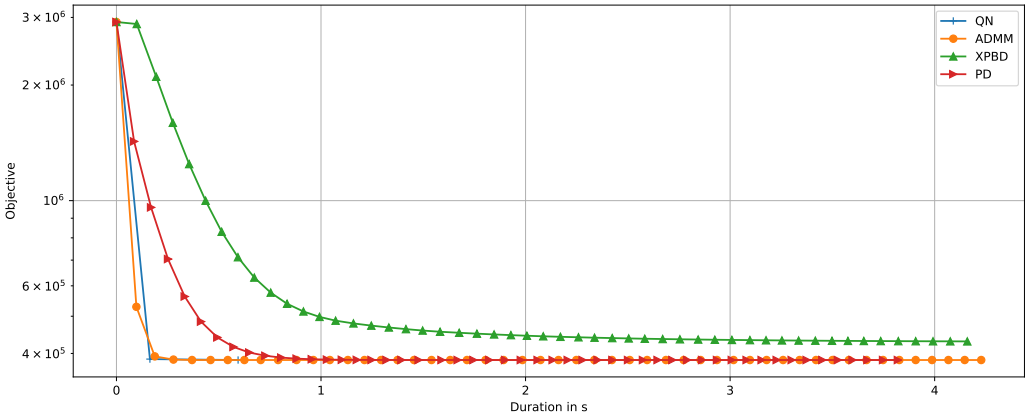


Figure 5.7: Values of the objective function of the variational form of implicit Euler integration (see Eq. 3.7) over the accumulated durations of the iterations over a single time step of size $1 \cdot 10^{-2}$ s for an untwisting beam using strain constraints with stiffness $1 \cdot 10^{8}$.

5.5.5 Convergence Properties With Large Time Steps and Stiffnesses

Figure 5.4 shows data for time step $1 \cdot 10^{-2}$ s and constraint stiffness $1 \cdot 10^{8}$. While the results are qualitatively the same for a wide range of time steps and stiffness values, noticable differences arise when a combination of large time steps and large stiffness values is used. Consider the corresponding plot with time step $1 \cdot 10^{-1}$ s and constraint stiffness $1 \cdot 10^{9}$ in Figure 5.8. Again, QN makes the fastest progress towards decreasing the objective, followed by ADMM, PD and XPBD. However, from 680 iterations onwards the objective value achieved by the XPBD solver is lower than the objective value achieved by the PD solver. In this setting, QN, ADMM and XPBD converge to configurations that achieve roughly the same objective function value, while it is PD whose final objective is larger than the other solvers'. After 1000 iterations, PD has a relative error (see Eq. 5.5) of approx. 0.02 %. While this appears negligible, it is worth noting that the minimal objective value that PD arrives at is still approx. 2.3 times larger than the minimal value achieved by the other solvers.

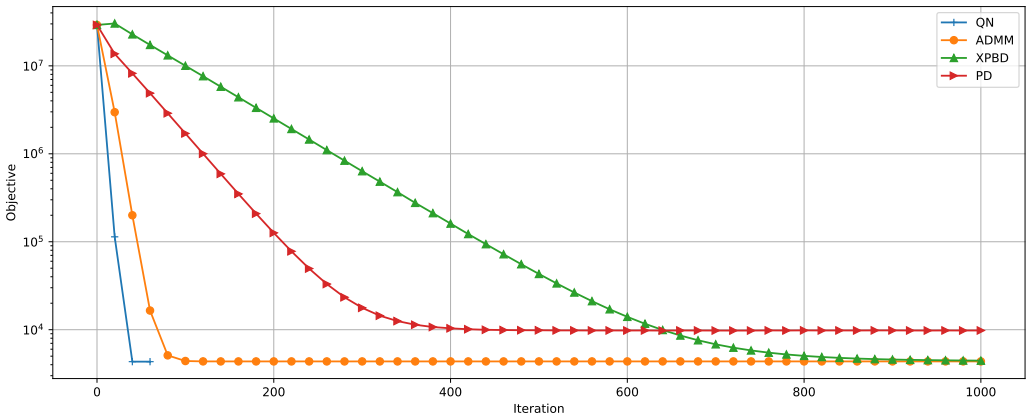


Figure 5.8: Values of the objective function of the variational form of implicit Euler integration (see Eq. 3.7) over the iterations of a single time step of size $1 \cdot 10^{-1}$ s for an untwisting beam using strain constraints with stiffness $1 \cdot 10^{9}$.

As discussed in Section 4.2.4, each PD iteration is guaranteed to weakly reduce the objective function. However, this only ensures that the objective function at the configuration PD converges to is never higher than the initial objective value. There is no guarantee that PD converges to the true implicit positions. Thus, the observation that PD can fail at reducing the objective function all the way is not surprising. To gain a better understanding of why the convergence properties of the PD solver worsen when combining larger time steps and stiffness values, we again create separate plots for the inertial and constraint terms. This is analogous

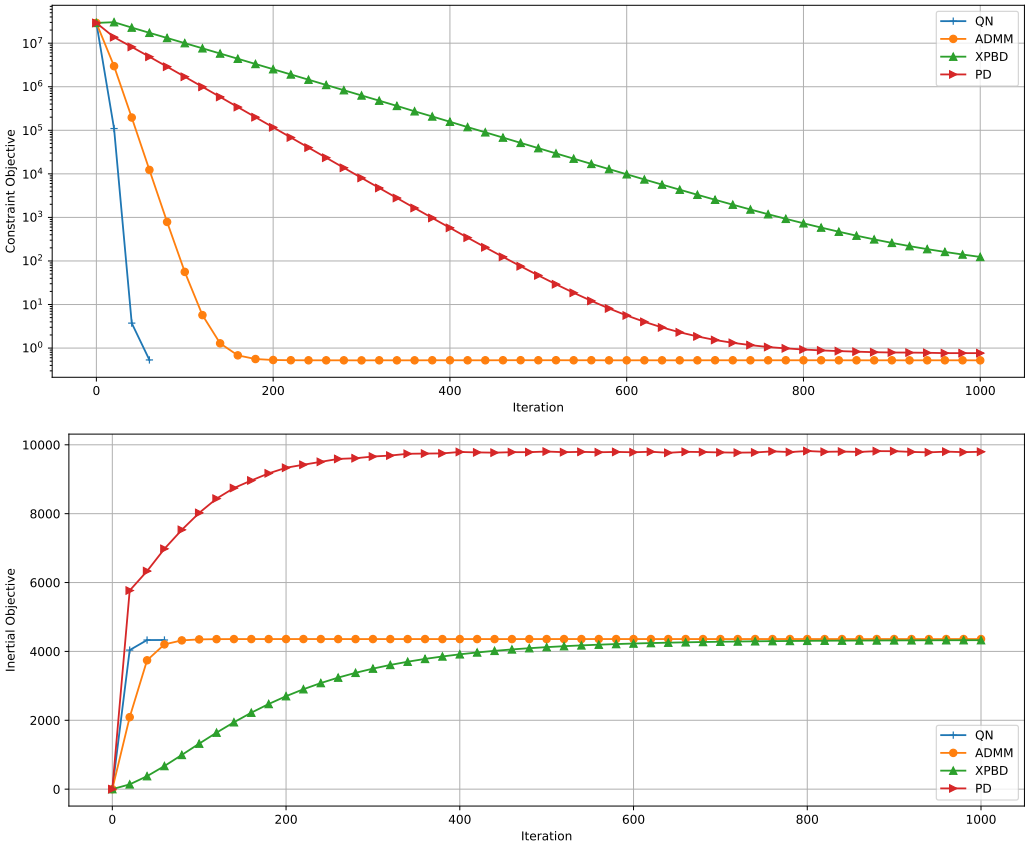


Figure 5.9: Values of the constraint (top) and inertial (bottom) terms of the objective function of the variational form of implicit Euler integration (see Eq. 3.7) over the iterations of a single time step of size $1 \cdot 10^{-1}$ s for an untwisting beam using strain constraints with stiffness $1 \cdot 10^{9}$.

to the experiments conducted in Figure 5.5 to investigate the unfavorable convergence properties of XPBD observed in Figure 5.4.

Again, Figure 5.9 shows that the solvers mostly decrease the constraint term and increase the inertial term during each iteration. QN, ADMM and PD are more successful at reducing the constraint energy than XPBD, even though PD converges to a constraint energy that is slightly larger than the one achieved by QN and ADMM this time. Each of the PD-style solvers manages to bring the constraint energy down to close to zero. This is significantly lower than the minimum achieved for a time step of $1 \cdot 10^{-2}$ s and stiffness $1 \cdot 10^{8}$ (see Fig. 5.5). However, it is evident that XPBD has not fully converged yet as it still continues to decrease the constraint energy values. All in all, the results for the constraint energy are qualitatively the same as the ones observed for time step $1 \cdot 10^{-2}$ s and stiffness

$1 \cdot 10^8$. On the other hand, the plots for the inertial term differ quite significantly. With larger time step and stiffness, QN, ADMM and XPBD converge to roughly the same inertial term, whereas the PD solver converges to a value that is more than twice as large. Already after 20 iterations, the inertial term achieved by PD is larger than the value that the other solvers converge to. Still, PD continues to increase the inertial term during the first 400 iterations. Finally, each of the solvers converge to an inertial term that is significantly lower than the corresponding inertial term obtained with time step $1 \cdot 10^{-2}\,\mathrm{s}$ and stiffness $1 \cdot 10^8$.

Since the weight of the inertial term decreases with increasing time step size and the weight of the constraint term increases with increasing stiffness values, moving particles away from their inertial positions to reduce the constraint energy is encouraged in Figure 5.9 compared to Figure 5.5. This is reflected in the fact that the solvers reduce the constraint energy to a lower value, even though a larger stiffness value is used. The results suggest that the PD solver is too aggressive with moving particles away from their inertial positions if the associated penalty is largely outweighed by the incentives for minimizing the constraint energies. This issue appears to be particularly serious during the first solver iterations, where the derivatives of the quadratic inertial penalty are small. Without further experiments, it is difficult to understand why this effect occurs when using the PD solver while the other solvers are unaffected.

5.5.6 Conservation of Global Linear and Angular Momentum With Large Time Steps and Stiffnesses

To gain further insights into the cause of the large inertial term at the converged configuration from PD, we repeat parts of the experiments in Figure 5.6 and take a closer look at the geometries that the solvers arrive at after 1000 iterations. Figure 5.10 a shows a comparison between the XPBD and QN geometries. Similar to the results in Figure 5.6 a, the XPBD solver does not untwist the beam as far as the QN solver. This is in line with the observation that the constraint term for XPBD is larger than the constraint term of PD-style solvers after 1000 iterations (see Figure 5.9). More interestingly, the comparison of the QN, ADMM and PD geometries in Figure 5.10 b shows that all PD-style solvers succeed at untwisting the beam to its reference configuration. However, while the orientations of the beams appear the same between solvers, the positions of their centers of gravity are shifted with respect to each other. This shift is very apparent for PD, whereas the QN and ADMM geometries are almost overlapping.

The results in Figure 5.10 suggest that the large inertial term observed for the PD solver in Figure 5.9 is caused by undesired rigid-body motion introduced during the solver iterations. The shifted center of gravity of the beam is unexpected

(a) QN and XPBD after 1000 iterations

(b) QN, PD and ADMM after 1000 iterations

Figure 5.10: Geometries of an untwisting beam with strain constraints (stiffness $1 \cdot 10^9$) after 1000 iterations over a single time step of size $1 \cdot 10^{-1}$ s using XPBD (pink), PD (orange), ADMM (green) and QN (cyan).

in light of the fact that the beam has zero global linear and angular momentum in its initial twisted configuration and no external forces are applied. At the true implicit positions that the PD solver attempts to converge to, global linear and angular momentum are preserved [BML+14]. However, the shifted center of gravity indicates that the global linear momentum of the beam is non-zero after 1000 PD iterations.

To dig deeper, we plot the norms of the global linear and angular momentum of the beam over the solver iterations in Figure 5.11. First, the plots confirm that the initial linear and angular momenta are zero at the initial configuration. However, the QN solver is the only solver that succeeds at preserving both types of momentum. XPBD preserves linear momentum, but increases the angular momentum. PD and ADMM fail to preserve either type of momentum. The bulk of the increase in linear momentum observed for PD and ADMM happens during the first 20 iterations. The final linear momentum observed for the PD solver is significantly larger than the one observed for the other solvers. During the rest of the iterations, the linear momentum remains almost constant for all solvers. It is worth pointing out that the plot for the linear momentum is representative for the corresponding plots for almost all combinations of time step size and constraint stiffness. On the other hand, the plots for the angular momentum are more varied. The main commonality is that the angular momentum norms are much smaller than the linear momentum norms observed for PD and ADMM solvers. For these reasons, we focus our attention on linear momentum.

First, the data provides insight into why the spike of the inertial term of the objective function after 20 PD iterations in Figure 5.9 does not go along with a large decrease in the constraint term: The spike is caused by rigid-body motion, to which elastic energy potentials are invariant. Secondly, we discussed in Section 4.2.6 that PD can be considered a quasi-Newton method with constant Hessian approximation. The QN solver enhances the PD solver by integrating local curvature information into the Hessian approximation via an LBFGS update and adding

a line search algorithm. Since the QN solver does not introduce any global linear momentum, it follows that the reason for the PD solver's inability to preserve the linear momentum is the lack of one or both of these features of the QN solver. To find out which of these factors is more important, it might be interesting to remove the line search or the LBFGS-update from the QN solver and take another look at the momenta. Lastly, recall that ADMM and PD are almost identical for strain constraints if the ADMM weights w_i are set to $\sqrt{k}$. For the experiments in Figure 5.11, $w_i \approx \frac{1}{2}\sqrt{k}$ is used. Thus, the smaller linear momentum observed for ADMM invites the hypothesis that lower ADMM weights can be beneficial for the preservation of linear momentum. However, more effort is required to verify this claim and to establish the causal justification.

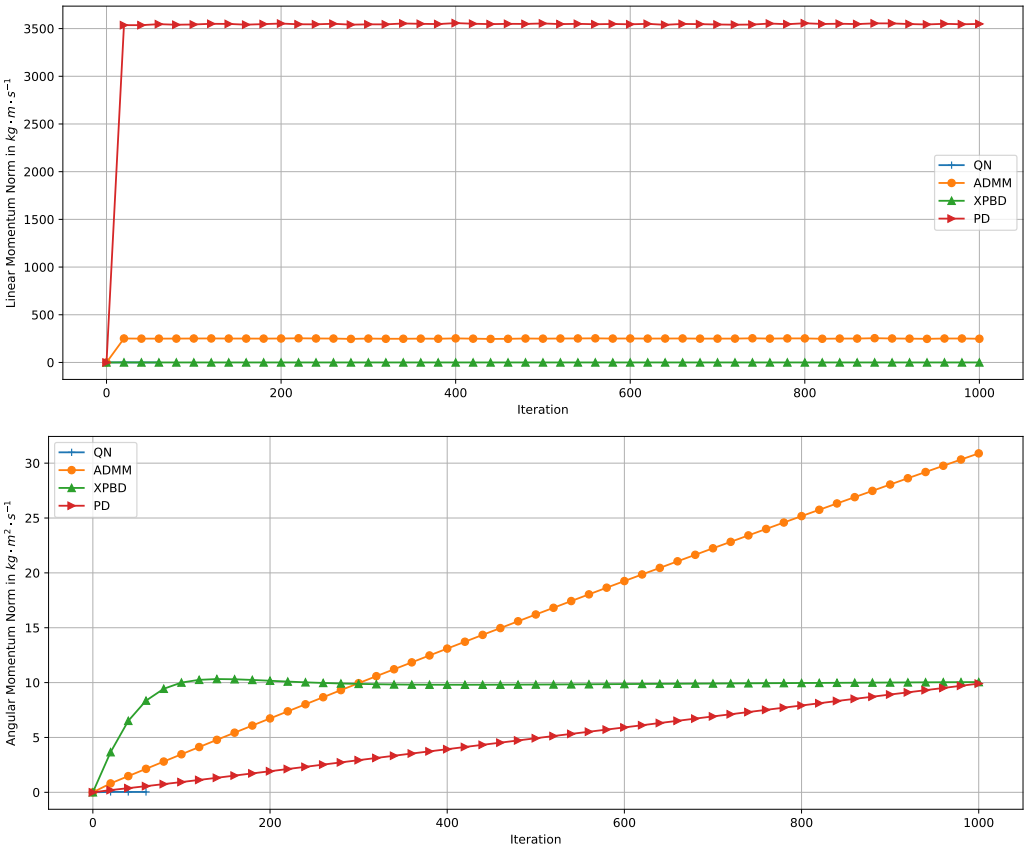


Figure 5.11: Norms of the global linear (top) and global angular (bottom) momenta in $\mathrm{kg} \cdot \mathrm{m}^2 \cdot \mathrm{s}^{-1}$ over the iterations of a single time step of size $1 \cdot 10^{-1}$ s for an untwisting beam with strain constraints with stiffness $1 \cdot 10^9$. The required velocities $\boldsymbol{v}_k$ at iteration k are computed via Verlet integration (see Sec. 5.2).

5.6 Twisting Beam Simulations - Strain Material

In the twisting beam experiment, the twisting motion of the beam is achieved by position constraints whose reference positions are rotated in-plane at the ends of the beam. These position constraints and the tetrahedral constraints for the strain material model are incompatible in the sense that there are no positions $\boldsymbol{q}$ for which all constraint energies are minimized at the same time. Thus, the twisting beam experiments put the the solvers' abilities to make compromises between competing constraints to the test. The ADMM weights for the twisting beam experiments are reused from the parameter screens for the untwisting beam experiments. We start by giving an overview of the convergence properties of different solvers in the twisting beam experiments in Section 5.6.1. Relevant details for the XPBD and ADMM solvers are discussed in Sections 5.6.2 and 5.6.3.

5.6.1 Overview Over Convergence Properties

To gain insights into the convergence properties of the solvers in the twisting beam examples, we again start by plotting the objective values over the solver iterations. The resulting graphs for a time step size of $1 \cdot 10^{-1}\,\mathrm{s}$ and a stiffness value of $1 \cdot 10^{9}$ are shown in Figure 5.12. While the results obtained for the QN and PD solvers are qualitatively the same for all time step sizes and stiffness values, the behavior of XPBD and ADMM can vary quite dramatically across different parameter settings. The plots in Figure 5.12 show that both QN and PD converge to a configuration with a lower objective value than the initial objective value within the first 20 iterations. Note that it takes a larger number of iterations to achieve convergence in the corresponding untwisting beam experiments (see Fig. 5.8). This is particularly noticable for the PD solver. Figure 5.12 shows that ADMM exhibits the same behavior as QN and PD for time step $1 \cdot 10^{-1}\,\mathrm{s}$ and stiffness $1 \cdot 10^{9}$. While this can be observed for a variety of different parameter settings, there are also many cases where ADMM either does not converge at all or converges to a solution whose objective value is significantly lower than the objective value achieved by the other PD-style solvers. Lastly, XPBD converges to a configuration with an objective value that is significantly larger than the initial objective value in Figure 5.12. Again, this observation is repeated for a variety of different parameters. However, there are also combinations of time step sizes and stiffness values for which XPBD is successful at reducing the objective to below its initial value. Even in such settings, the minimal objective value achieved by XPBD is almost always higher than the one achieved by QN and PD.

During the discussion of the untwisting beam experiments using large time steps and stiffness values (see Fig. 5.8), we pointed out that the PD solver can fail to decrease the objective value to the value achieved by the QN solver. This

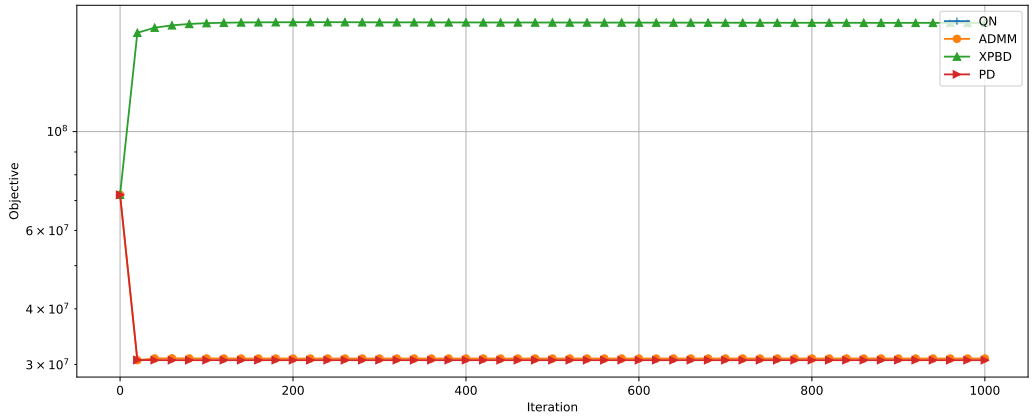


Figure 5.12: Values of the objective function of the variational form of implicit Euler integration (see Eq. 3.7) over the iterations of a single time step of size $1 \cdot 10^{-1}$ s for a twisting beam using strain constraints with stiffness $1 \cdot 10^9$.

is caused by the introduction of undesired rigid-body motion during early PD iterations when the penalty for moving particles away from their inertial positions is largely outweighed by the incentives for minimizing constraint energies. Since the position constraints at the ends of the beam in the twisting beam experiments further disincentivize moving the beam's center of gravity, it is not surprising that this failure mode of the PD solver is not encountered. The fact that convergence is achieved after fewer iterations during the twisting beam experiments than during the unwisting beam experiments is expected considering that the initial particle positions are much closer to the true implicit positions that the solvers aim to converge. To see this, observe that the QN solver untwists the beam from its initial deformed configuration all the way to its undeformed reference configuration over a single time step in Figure 5.10 b. In contrast, the beam is only twisted by a couple of degrees during the twisting beam experiments. As mentioned earlier, the results for XPBD and ADMM are more complex and merit a more detailed discussion below.

5.6.2 Convergence Properties of XPBD

During the untwisting beam experiments using the strain material model discussed earlier, each solver reduces the objective value compared to the initial objective value during the first 1000 iterations, irrespective of the chosen time step size and stiffness value. As indicated by the data in Figure 5.12, this is not always the case for the twisting beam experiments. For various combinations of time step sizes and stiffness values, the positions computed by XPBD after 1000 iterations yield a higher objective function value than the initial objective value. We say

that XPBD fails for such parameters. For each time step, the stiffness values for which XPBD fails are listed in Table 5.2. Note that the results are identical when restricting to the constraint term of the objective function. In other words, the settings in which XPBD fails to decrease the objective are exactly those for which the solver iterations cause the constraint term to increase. Table 5.2 shows that XPBD fails in settings where large stiffness values are used. In particular, XPBD always fails once the stiffness is larger than some time step dependent threshold. This threshold increases with increasing time step size.

time step in s	$1 \cdot 10^{-1}$	$1 \cdot 10^{-2}$	$1 \cdot 10^{-3}$	$1 \cdot 10^{-4}$
stiffness	$\geq 1 \cdot 10^{7}$	$\geq 1 \cdot 10^{8}$	$\geq 1 \cdot 10^{10}$	$\geq 1 \cdot 10^{12}$

Table 5.2: Combinations of time step sizes and stiffness values for which XPBD fails. For a stiffness value k, we write $\geq k$ to indicate that XPBD fails for all tested stiffnesses k' such that $k' > k$. Identical results are obtained when restricting to the constraint term of the objective function.

As discussed before, the large constraint terms that are observed when XPBD fails are most likely caused by a combination of the simplifying assumptions during the XPBD derivation and oscillations of the Gauss-Seidel-type solver at its core. We assess that the latter contributes more to XPBD failures in the twisting beam experiments for a variety of reasons. First, it is easy to see how mixing incompatible position and strain constraints could lead to more severe oscillations. Since there are no particle positions $\boldsymbol{q}$ for which both the position constraints and the strain constraints are satisfied at the same time, individual constraint projections will continue to move particles back and forth over and over again. On the other hand, all strain constraints are satisfied in the reference configuration of the beam in the untwisting beam scenario. It is natural to assume that the former scenario is more challenging for Gauss-Seidel-type solvers. Secondly, the severity of the oscillations is expected to increase with both time step size and stiffness values since both factors enable larger position updates during individual constraint projections. This is in line with the data presented in Table 5.2.

To get a feeling for the visual effects resulting from the large (constraint) objective value achieved when XPBD fails, we take a closer look at the geometry of a twisting strain beam after 1000 XPBD iterations over a time step of $1 \cdot 10^{2}\,\mathrm{s}$ with constraint stiffness $1 \cdot 10^{12}$ in Figure 5.13. The left panel shows that the surface of the XPBD geometry is noticably rugged. Recall that the same observation is made after 10 XPBD iterations in the untwisting beam setting (see Fig. 5.6). The effect is particularly noticable at the ends of the beam, where individual tetrahedra protrude from the volume of the body. To highlight this, we zoom in on the geometry at one end of the beam in the right panel of Figure 5.13. The geometry that the QN solver converges to is given as a reference. While the QN geometry is flat

at the end of the beam, the XPBD geometry is irregular. Additionally, it appears that the end of the beam is twisted further in the counterclockwise direction when using the XPBD solver.

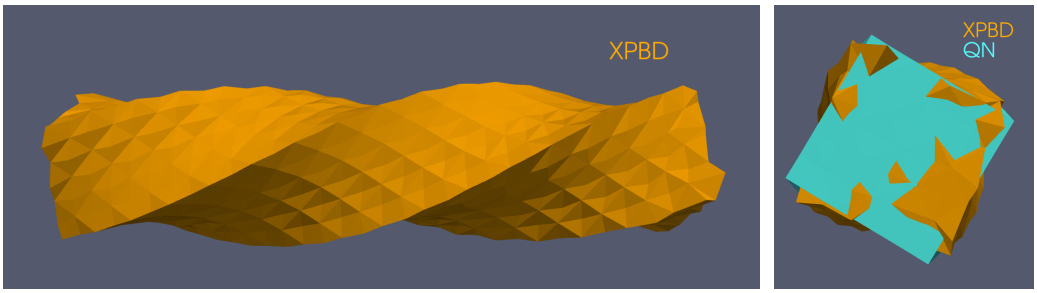


Figure 5.13: Geometries of a twisting beam with strain constraints (stiffness $1 \cdot 10^{12}$) after 1000 XPBD iterations over a time step of size $1 \cdot 10^{-2}$ s from different angles. In the right panel, the converged QN geometry is shown as a reference.

Again, the observed rugged surface fits with our hypothesis that the failures of the XPBD solver in the twisting beam scenario are due to oscillations of the Gauss-Seidel-type solver at the core of XPBD. The protrusions at the end of the beam are most likely caused by oscillations between the incompatible position constraints and local strain constraints. Together, our results from Figure 5.6 and Figure 5.13 suggest that oscillations can cause noticable geometric artifacts if an insufficiently low number of XPBD iterations is used or if subsets of constraints are fundamentally incompatible. The observation that the beam is twisted further from its initial configuration for XPBD than for QN is expected considering that the XPBD solver has no explicit penalty for moving particles away from their inertial positions (see Sec. 4.1.6). Thus, XPBD has more freedom to update particle positions in such a way that the position constraints that cause the beam to twist are satisfied. On the other hand, moving particles away from their inertial positions is associated with a penalty that is quadratic in the inverse of the time step size when using the QN solver. As a result, the final QN positions at the end of the beam are further away from the reference positions of the position constraints. While the behavior of the QN solver is physically accurate, it also highlights one of its weaknesses: Due to the penalty incurred for moving particles away from their inertial positions and the fact that position constraints can only be modelled via stiff springs, exerting exact control over a set of particle positions is challenging. Note that even the large stiffness value of $1 \cdot 10^{12}$ used for the position constraints in Figure 5.13 is insufficient for moving particles to their reference positions when using the QN solver.

The fact that the XPBD solver is more liberal with moving particles at the ends of the beam towards the reference positions of the position constraints should be

time step in s	$1 \cdot 10^{-1}$	$1 \cdot 10^{-2}$	$1 \cdot 10^{-3}$	$1 \cdot 10^{-4}$
stiffness	–	$\geq 1 \cdot 10^{5}$	$\geq 1 \cdot 10^{5}$	$\geq 1 \cdot 10^{7}$

Table 5.3: Combinations of time step sizes and stiffness values for which the inertial term of XPBD is larger than the inertial term of the QN solver after 1000 iterations.

reflected in a larger inertial term compared to the QN solver. In Table 5.3, we list combinations of time step sizes and stiffness values for which the inertial term after 1000 XPBD iterations is larger than the inertial term achieved at the converged QN configuration. The results show that XPBD's inertial term is always smaller when a time step of $1 \cdot 10^{-1}\,\mathrm{s}$ is used, but almost always larger for time step sizes larger than or equal to $1 \cdot 10^{-2}\,\mathrm{s}$. The only exception occurs for small stiffness values when a time step of $1 \cdot 10^{-4}\,\mathrm{s}$ is used.

The results are in line with the observations from the previous experiments. In particular, Table 5.3 shows that the inertial term achieved by XPBD is larger than the inertial term achieved by QN for the parameters used in Figure 5.13 (time step $1 \cdot 10^{-2}\,\mathrm{s}$, stiffness $1 \cdot 10^{12}$). The observation that the inertial term achieved by the QN solver is larger for the large time step $1 \cdot 10^{-1}\,\mathrm{s}$ matches that the inertial penalty taken into account by the QN solver is quadratic in the inverse of the time step size. As this penalty is not reflected in the XPBD update equations, it is expected that XPBD's inertial term becomes larger than QN's inertial term when the time step is reduced.

In our discussion of Figure 5.5, we mentioned that the inertial term achieved by XPBD is almost always lower than the inertial term achieved by PD-style solvers in the untwisting beam experiment. We suggested that this is due to the fact that the tetrahedral strain constraints simply do not provide enough incentive to move particles away from their inertial positions and that XPBD is less successful at minimizing the constraint energies of the tetrahedral elements than PD-style solvers. On the other hand, the position constraints at the end of the twisting beam are associated with quadratic spring energies that are simple to optimize and do encourage moving particles away from their inertial positions. In this setting, the lack of an explicit penalty term in the XPBD update equations becomes evident.

5.6.3 Convergence Properties of ADMM

For some parameters, we observe that ADMM converges to a configuration for which the objective value is significantly lower than the objective values achieved by all other solvers during the twisting beam experiments. As an example, we plot the objective values over the iterations for a time step of $1 \cdot 10^{-2}\,\mathrm{s}$ and stiffness $1 \cdot 10^{10}$ in Figure 5.14. After increasing the objective value during the first 40 iterations, the ADMM solver arrives at a lower objective value than the initial

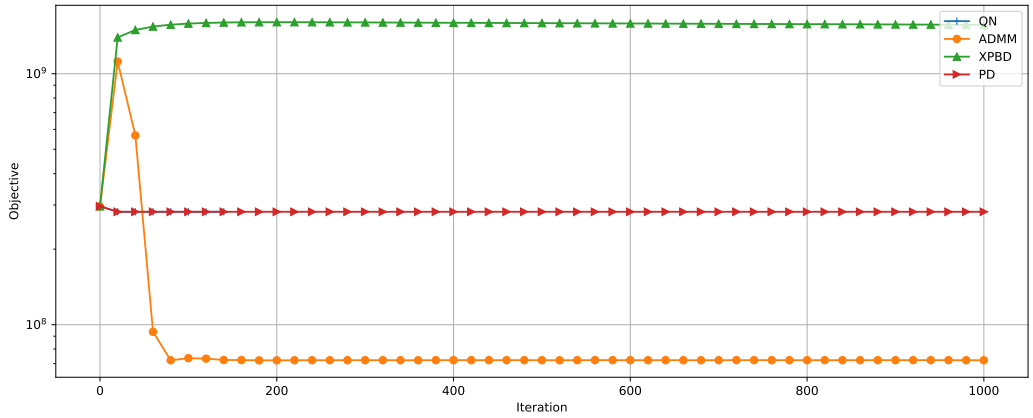


Figure 5.14: Values of the objective function of the variational form of implicit Euler integration (see Eq. 3.7) over the iterations of a single time step of size $1 \cdot 10^{-2}$ s for a twisting beam using strain constraints with stiffness $1 \cdot 10^{10}$.

objective value after 60 iterations. Finally, ADMM converges to a configuration where the objective value is almost an order of magnitude lower than the minimal objective value achieved by PD and QN after 80 iterations.

Again, we create separate plots for the constraint and inertial terms of the objective function in Figure 5.15. The graph for the constraint term looks almost identical to the graph for the entire objective function in Figure 5.14. During the first 40 iterations, the constraint term achieved by ADMM is larger than the constraint term that QN and PD converge to. After 80 ADMM iterations, the constraint term converges to a value that is significantly lower than the one achieved by the other PD-style solvers. The most dramatic decrease in the constraint term occurs between 40 and 60 ADMM iterations. At the same time, ADMM's inertial term increases dramatically. The final inertial term achieved by ADMM is significantly higher than the inertial terms achieved by all other solvers, including XPBD.

The results suggest that ADMM is able to find particle positions that yield much lower constraint energies than the other solvers. The fact that the decrease in the constraint term between 40 and 60 iterations goes along with a sharp increase in the inertial term indicates that these particle positions differ significantly from the initial twisted configuration. This is surprising, since the reference positions of the position constraints at the ends of the beam that initiate the twisting motion are only rotated by a couple of degrees. Thus, we would expect the optimal configuration to have a small inertial term, as is observed for the configurations that QN and PD converge to. One possible explanation is that QN and PD converge to a local minimum of the objective function, whereas ADMM converges to a global solution. This hypothesis is supported by the observation that ADMM

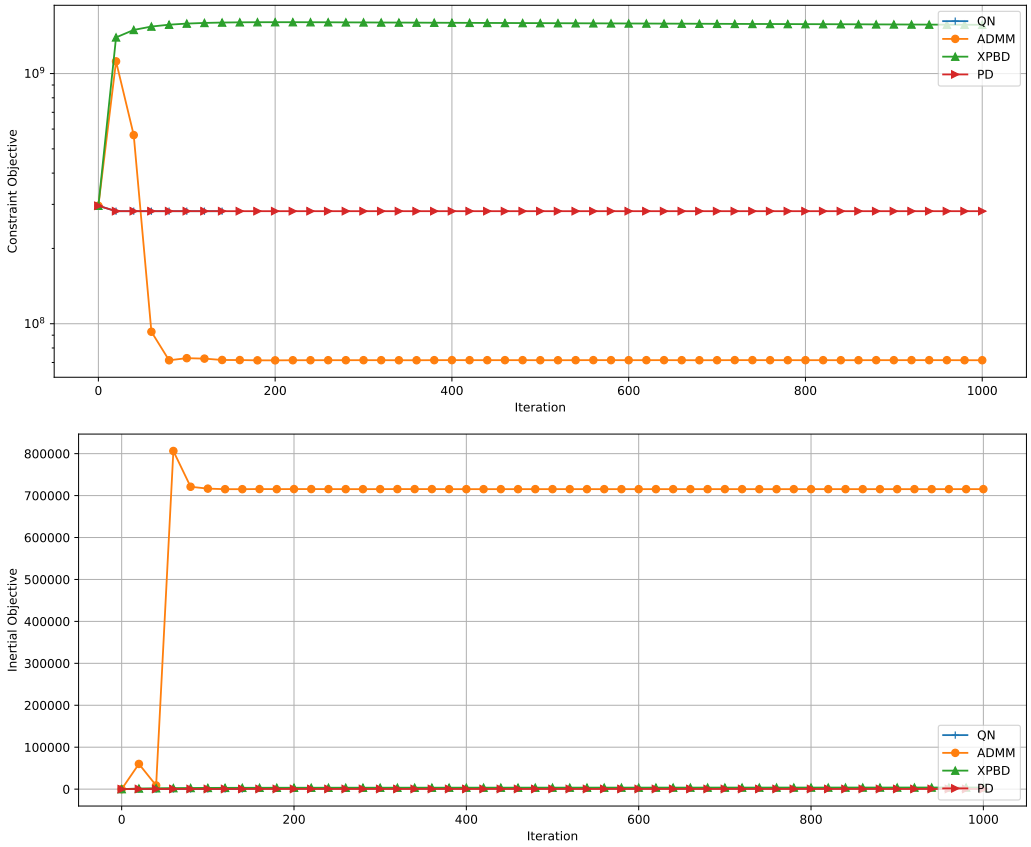
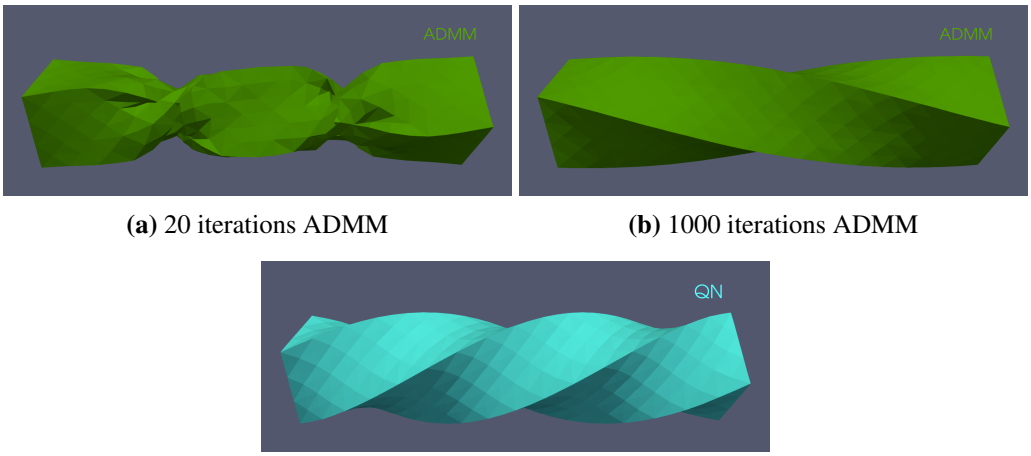


Figure 5.15: Values of the constraint (top) and inertial (bottom) terms of the objective function of the variational form of implicit Euler integration (see Eq. 3.7) over the iterations of a single time step of size $1 \cdot 10^{-1}$ s for an untwisting beam with strain constraints with stiffness $1 \cdot 10^{9}$.

increases the objective value during the first 20 iterations (see Fig. 5.14). Informally, this suggests that the ADMM solver spends the early solver iterations escaping the valley containing the local minimum that QN and PD converge to so that the global minimum can be achieved during later iterations.

To gain more insights, we take a closer look at the ADMM geometries after 20 and 1000 iterations in Figure 5.16. After 20 ADMM iterations, the beam appears overtwisted at two locations at roughly one third and two thirds of the beam width. In the vicinity of these overtwisted locations, the surface of the beam is highly irregular. In contrast, the surface of the beam appears smooth at the converged ADMM geometry. At the configuration ADMM converges to, the beam is significantly closer to its reference configuration than at the geometry computed



(a) 20 iterations ADMM **(b)** 1000 iterations ADMM

(c) QN after convergence

Figure 5.16: Geometries of a twisting beam with strain constraints (stiffness $1 \cdot 10^{10}$) after 20 (a) and 1000 (b) ADMM iterations over a time step of size $1 \cdot 10^{-2}$ s. The converged QN geometry (c) is given as a reference.

by the QN solver. Additionally, the converged ADMM and QN geometries are twisted in opposite directions.

The results in Figure 5.16 support the hypothesis that ADMM converges to a global minimum of the objective function, whereas QN and PD converge to a local minimum. By overtwisting the beam at two locations, the ADMM solver is eventually able to arrive at a configuration where most of the twist from the initial deformed configuration is removed without violating the position constraints at the end of the beam. Recall that the value of the objective function can be interpreted as the energy of the beam at a configuration (see Sec. 5.2). Then, it is obvious that the overtwisting motion required to achieve this global minimum is not phyiscally plausible, since the energy of the beam is increased to a larger value than the energy at the initial deformed state in the process. However, without external forces, the energy of the beam should be conserved according to Newton's laws of motion. For the experiments above, ADMM weights of $w_i = 0.3\sqrt{k}$ were used, where k is the strain constraint stiffness. The fact that PD does not converge to the global minimum even though it is almost identical to ADMM with weights $w_i = \sqrt{k}$ is in line with our observation that lower ADMM weights encourage more aggressive optimization of the constraint energies (see Sec. 5.4). Finally, Figure 5.14 and Figure 5.16 highlight the ramifications of terminating the ADMM solver before convergence is achieved: Early termination can increase the beam energy and cause visual geometric artifacts on the surface of the beam.

For each time step size, we list the stiffness values for which ADMM converges to the global minimum in the twisting beam experiments in Table 5.4. For

time step in s	$1 \cdot 10^{-1}$	$1 \cdot 10^{-2}$	$1 \cdot 10^{-3}$	$1 \cdot 10^{-4}$
stiffness	$1 \cdot 10^{6}$	$1 \cdot 10^{10}$	$1 \cdot 10^{10}$	$1 \cdot 10^{12}$
	$1 \cdot 10^{8}$		$1 \cdot 10^{12}$	
	$1 \cdot 10^{11}$			

Table 5.4: Combinations of time step sizes and stiffness values for which the ADMM solver converges to the global minimum during the twisting beam experiments using the strain material model.

the large time step $1 \cdot 10^{-1}$ s, the ADMM solver converges to the global minimum for stiffness values as low as $1 \cdot 10^{6}$. As the time step size is decreased, larger stiffness values are required. Since achieving the global minimum requires moving particles far away from their inertial positions, it is expected that such aggressive minimizations of the constraint energies are favored by large stiffness values and large time steps (see Sec. 3.3). Still, it is likely that the choice of the ADMM weights has a strong impact on whether the ADMM solver converges to the the global minimum or not.

Lastly, we show that there are time step sizes and stiffness values for which ADMM does not converge at all in the twisting beam scenario. Conside the plot of the objective values over the iterations for a time step of $1 \cdot 10^{-2}$ s and a stiffness value of $1 \cdot 10^{8}$ in Figure 5.17. The graphs show that ADMM oscillates between configurations with objective values that are larger than the initial objective value and objective values that are smaller than the minimal objective values achieved by QN and PD. Similar results are observed for time step size $1 \cdot 10^{-1}$ s and stiffness $1 \cdot 10^{10}$ and timestep size $1 \cdot 10^{-2}$ and stiffness values $1 \cdot 10^{8}$ and $1 \cdot 10^{10}$.

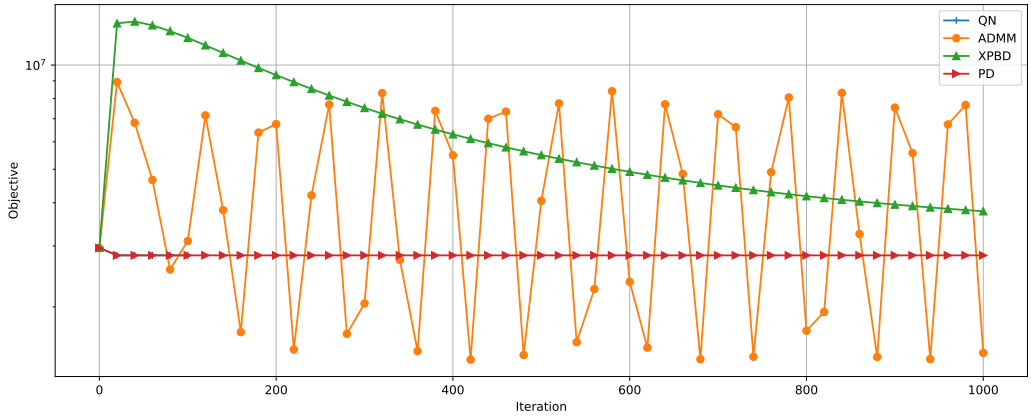


Figure 5.17: Values of the objective function of the variational form of implicit Euler integration (see Eq. 3.7) over the iterations of a single time step of size $1 \cdot 10^{-2}$ s for a twisting beam using strain constraints with stiffness $1 \cdot 10^{8}$.

Together with the results discussed above, the data from Figure 5.17 suggests that ADMM drives particle positions towards the global minimum of the objective function, but eventually undoes its progress once the particle positions get too close. It is not clear why this failure to converge occurs for the parameters listed above, but not for the parameters in Table 5.4. Again, the choice of the ADMM weights most likely plays an important role here. Recall that ADMM weights used during the twisting beam experiments were adopted from the corresponding parameter screens for the untwisting beam experiments (see Sec. 5.4). In light of the similarities between PD and ADMM with weights $w_i = \sqrt{k}$, it is reasonable to assume that convergence could be achieved with more appropriate ADMM weights. As a result, it might be necessary to fine-tune ADMM weights for individual simulation scenarios. Of course, this is highly impractical.

5.7 Untwisting Beam Simulations - Neohookean Material

5.8 Discussion

Chapter 6

Conclusion

Bibliography

[Bar96] David Baraff. “Linear-Time Dynamics Using Lagrange Multipliers”. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’96. New York, NY, USA: Association for Computing Machinery, 1996, pp. 137–146 (cited on page 19).

[BW98] David Baraff and Andrew Witkin. “Large Steps in Cloth Simulation”. In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’98. New York, NY, USA: Association for Computing Machinery, 1998, pp. 43–54 (cited on pages 11, 18, 20).

[BKCW14] Jan Bender, Dan Koschier, Patrick Charrier, and Daniel Weber. “Position-based simulation of continuous materials”. In: *Computers Graphics* 44 (2014), pp. 1–10 (cited on pages 15, 30, 31).

[BML+14] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. “Projective Dynamics: Fusing Constraint Projections for Fast Simulation”. In: *ACM Trans. Graph.* 33.4 (July 2014) (cited on pages 5, 9, 10, 14, 28, 30, 42–45, 48, 69, 74, 78, 83).

[CC05] Steven C. Chapra and Raymond Canale. *Numerical Methods for Engineers*. 5th ed. USA: McGraw-Hill, Inc., 2005 (cited on pages 7, 8).

[GQ20] J.H. Gallier and J. Quaintance. *Linear Algebra And Optimization With Applications To Machine Learning - Volume Ii: Fundamentals Of Optimization Theory With Applications To Machine Learning*. World Scientific Publishing Company, 2020 (cited on page 59).

[LBK17] Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. “Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials”. In: *ACM Trans. Graph.* 36.4 (July 2017) (cited on pages 15, 48, 50–52, 54–58, 63, 74).

[MM21] Miles Macklin and Matthias Muller. “A Constraint-based Formulation of Stable Neo-Hookean Materials”. In: *Proceedings of the 14th ACM SIGGRAPH Conference on Motion, Interaction and Games*. MIG ’21. Virtual Event, Switzerland: Association for Computing Machinery, 2021 (cited on pages 15, 40).

[MMC16] 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*. MIG ’16. Burlingame, California: Association for Computing Machinery, 2016, pp. 49–54 (cited on pages 5, 23, 25, 31, 33, 34, 36, 37).

[MHHR06] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. “Position Based Dynamics”. In: *Vriphys: 3rd Workshop in Virtual Realitiy, Interactions, and Physical Simulation*. Ed. by Cesar Mendoza and Isabel Navazo. The Eurographics Association, 2006 (cited on pages 23–29, 69).

[NMK+06] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. “Physically Based Deformable Models in Computer Graphics”. In: *Computer Graphics Forum* 25.4 (2006), pp. 809–836 (cited on page 18).

[NW06] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. 2e. New York, NY, USA: Springer, 2006 (cited on pages 10, 37, 43, 46, 50, 52–54, 57, 63).

[OBLN17] Matthew Overby, George E. Brown, Jie Li, and Rahul Narain. “ADMM $\supset$ Projective Dynamics: Fast Simulation of Hyperelastic Models with Dynamic Constraints”. In: *IEEE Transactions on Visualization and Computer Graphics* 23.10 (2017), pp. 2222–2234 (cited on pages 15, 48, 59, 60, 62–64, 74).

[SLM06] Martin Servin, Claude Lacoursière, and Niklas Melin. “Interactive Simulation of Elastic Deformable Materials”. In: *Proc. SIGRAD* (Jan. 2006) (cited on pages 7, 8, 11, 17, 18, 20, 35).

[SB12] 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*. SIGGRAPH ’12. Los Angeles, California: Association for Computing Machinery, 2012 (cited on pages 12–16, 55).

[SGK18] Breannan Smith, Fernando De Goes, and Theodore Kim. “Stable Neo-Hookean Flesh Simulation”. In: *ACM Trans. Graph.* 37.2 (Mar. 2018) (cited on pages 14, 15, 41).

[SD06] Ari Stern and Mathieu Desbrun. “Discrete geometric mechanics for variational time integrators”. In: *ACM SIGGRAPH 2006 Courses*. SIGGRAPH ’06. Boston, Massachusetts: Association for Computing Machinery, 2006, pp. 75–80 (cited on pages 8, 9).

[TNGF15] Maxime Tournier, Matthieu Nesme, Benjamin Gilles, and François Faure. “Stable Constrained Dynamics”. In: *ACM Trans. Graph.* 34.4 (July 2015) (cited on pages 11, 17, 20, 21, 32, 37).

[UR95] Linda R. Petzold Uri M. Ascher Hongsheng Chin and Sebastian Reich. “Stabilization of Constrained Mechanical Systems with DAEs and Invariant Manifolds”. In: *Mechanics of Structures and Machines* 23.2 (1995), pp. 135–157 (cited on page 19).

[WRO11] Huamin Wang, Ravi Ramamoorthi, and James F. O’Brien. “Data-Driven Elastic Models for Cloth: Modeling and Measurement”. In: *ACM Transactions on Graphics* 30.4 (July 2011). Proceedings of ACM SIGGRAPH 2011, Vancouver, BC Canada, 71:1–11 (cited on page 43).

Index