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, April 21, 2024

Dennis Ledwon

Contents

Chapter 1

Introduction

Chapter 2

Related Work

Chapter 3

Method

- Method is probably not the right title here, it's more "Foundations" or something like that. Maybe give some introduction to this chapter that your goal is to present the basics necessary for XPBD and PD and maybe a teaser how optimization is need etc., a bit of an outline of the chapter.
- While citing books, put chapters into the citations.

3.1 Time-Integration of Physical Systems

Fabian recommends citing [GSS+15].

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. Maybe list all of them. Newton's first law is definitely going to be mentioned, not sure about the third one. I don't want to spend too much time on contact. 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 in order 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, both XPBD (Section 3.4.4) and PD (Section 3.5.3) are based on a numerical integration technique called implicit Euler integration [MMC16; BML+14]. The equations of motions due to Newton's second law are reviewed

in Section 3.1.1. Common approaches for numerical integration are introduced in Section 3.1.2. An alternative formulation of implicit Euler integration as an unconstrained optimization problem called the variational form of implicit Euler integration is presented in Section 3.1.3. Finally, additional considerations for the numerical integration of the equations of motion in real-time settings are covered in Section 3.1.4.

3.1.1 The Equations of Motion

The motion of a spatially discretized system with m particles evolving in time according to Newton's laws of motion 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 $\boldsymbol{f}(\boldsymbol{q}(t), \boldsymbol{v}(t))$. In the context of projective dynamics (Section 3.5), it is $\boldsymbol{q}(t), \boldsymbol{v}(t), \boldsymbol{f}(\boldsymbol{q}(t)) \in \mathbb{R}^{m\times 3}$ and $\mathbf{M} \in \mathbb{R}^{m\times m}$. In the context of position based dynamics (Section 3.4), it is $\boldsymbol{q}(t), \boldsymbol{v}(t), \boldsymbol{f}(\boldsymbol{q}(t)) \in \mathbb{R}^{3m}$ and $\mathbf{M} \in \mathbb{R}^{3m\times 3m}$. $\dot{\boldsymbol{q}}(t)$ and $\dot{\boldsymbol{v}}(t)$ are short for $\frac{d\boldsymbol{q}}{dt}(t)$ and $\frac{d\boldsymbol{v}}{dt}(t)$, respectively. From now on, 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.

3.1.2 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.

The simplest approach to numerical integration is explict 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.

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].

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 Section 3.3.1). 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 Section 3.3) build on top of the integration schemes introduced here with the goal of alleviating their drawbacks without sacrificing their more advantageous properties.

3.1.3 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

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

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 $\tilde{\boldsymbol{q}} = \boldsymbol{q}_n + h\boldsymbol{v}_n + h^2\mathbf{M}^{-1}\boldsymbol{f}_{\text{ext}}$. The first and second term of the objective function are called the momentum potential and the elastic potential, 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 momentum potential and the elastic potential depends on the particle masses $\mathbf{M}$, the time step h and the material stiffness of the elastic potentials ψ_j. 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 Armijo condition or both Wolfe conditions (Section 3.2.1). In fact, many algorithms for solving non-linear systems of equations construct approximate merit functions if the objective function from an equivalent optimization problem is not available [NW06].

3.1.4 Considerations for Real-Time Simulations

As discussed in Section 3.1.2, 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 Eq. (3.4). Thus, additional measures are required in order 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 (Eq. (3.4)). The strategy of approximating the solution of Equation 3.4 is also at the heart of XPBD (Section 3.4.4), 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 (Section 3.5). The second method is particularly appealing in settings where visual plausibility is prioritized over physical accuracy.

3.2 Unconstrained Optimization

- Don't forget glue text!!!
- Truncate this section!!!
- Mention that the entire section is based on [NW06] and only cite other sources.

The goal of unconstrained optimization is to find the global minimizer of smooth, but generally nonlinear functions of the form $f\colon \mathbb{R}^n \to \mathbb{R}, n \in \mathbb{N}$, or formally

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}).$$

Here, f is called the objective function.

Most algorithms are incapable of finding global minimizers of general nonlinear functions. Instead, these algorithms begin their search at a starting point $\boldsymbol{x}_0$ and then iteratively improve this initial guess until a local minimizer is found. A local minimizer is a point $\boldsymbol{x}^*$ such that there is a neighborhood $\mathcal{N}$ of $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathcal{N}$. If the initial guess $\boldsymbol{x}^*$ is close enough to the global minimizer the local minimizer that the algorithm converges to can often coincide with a global minimizer.

3.2.1 Line Search Methods

It can be shown that $\nabla f(\boldsymbol{x}^*) = 0$ if $\boldsymbol{x}^*$ is a local minimizer and f is continuously differentiable in an open neighborhood of $\boldsymbol{x}^*$. The proof is by contradiction and establishes that if $\nabla f(\boldsymbol{x}^*) \neq 0$, then it is possible to pick a descent direction along which it is possible to decrease the value of the objective function if the step size is picked sufficiently small. This observation gives rise to the idea of a family of optimization algorithms called line search algorithms: Given the current iterate $\boldsymbol{x}_k$, pick a descent direction $\boldsymbol{p}_k$ and search along this direction for a new iterate $\boldsymbol{x}_{k+1}$ with $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$. This process is repeated until $\nabla f(\boldsymbol{x}_k)$ is sufficiently close to zero. It is important to note that $\nabla f(\boldsymbol{x}) = 0$ does not imply that $\boldsymbol{x}$ is a local minimizer. Instead, $\boldsymbol{x}$ is only guaranteed to be a local minimizer if the second-order optimality conditions are satisfied, which additionally require $\nabla^2 f(\boldsymbol{x})$ to be positive semidefinite.

Ideally, α_k is picked such that it is the minimizer of the one-dimensional optimization problem

$$\min_{\alpha_k > 0} f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k).$$

In most cases, it is infeasible to compute α_k exactly. Instead, the idea is to compute an approximation of α_k such that the objective function decreases sufficiently and that α_k is close enough to the true minimizer. Formally, these requirements are captured in the strong Wolfe conditions for step lengths α_k:

$$f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k) \leq f(\boldsymbol{x}_k) + c_1 \alpha_k \nabla f(\boldsymbol{x}_k)^T \boldsymbol{p}_k \tag{3.10}$$

$$\left| \nabla f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)^T \boldsymbol{p}_k \right| \leq c_2 \left| \nabla f(\boldsymbol{x}_k)^T \boldsymbol{p}_k \right| \tag{3.11}$$

for some constants $c_1 \in (0, 1), c_2 \in (c_1, 1)$. Equation 3.10 is called the sufficient decrease or the Armijo condition and states that the reduction in f should be proportional to both the step length α_k and the directional derivative $\nabla f(\boldsymbol{x}_k)^T \boldsymbol{p}_k$. Informally, the second condition (Eq. (3.11)), known as the curvature condition,

ensures that there is no more fast progress to be made along the search direction $\boldsymbol{p}_k$, indicated by the fact that $\left|\nabla f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)^T \boldsymbol{p}_k\right|$ is already rather small.

Step sizes satisfying the strong Wolfe conditions have the following properties under mild assumptions. Firstly, if $\boldsymbol{p}_k$ is a descent direction, it is possible to find a step size that satisfies the strong Wolfe conditions. In particular, the Armijo condition is always satisfied once α_k is sufficiently close to zero. Secondly, it can be shown that line search methods where α_k satisfies the strong Wolfe conditions for all k converge to a stationary point $\boldsymbol{x}^*$ with $\nabla f(\boldsymbol{x}^*) = 0$ if the search direction $\boldsymbol{p}_k$ is sufficiently far from orthogonal to the steepest descent direction $\nabla f(\boldsymbol{x}_k)$ for all k. Such line search algorithms are called globally convergent.

The general structure of line search methods is given in Algorithm 1.

Algorithm 1 Line Search Methods

require $\epsilon > 0$
procedure LINESEARCHMETHOD($\boldsymbol{x}_0$, ϵ)
  $\boldsymbol{x}_k = \boldsymbol{x}_0$
  while $\left\|\nabla f(\boldsymbol{x}_k)\right\| > \epsilon$ **do**
    compute a descent direction $\boldsymbol{p}_k$
    compute α_k that satisfies the strong Wolfe conditions
    $\boldsymbol{x}_k = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
  end while
  return with result $\boldsymbol{x}_k$
end procedure

Steepest Descent

The most obvious choice for the search direction $\boldsymbol{p}_k$ at iteration k is the negative gradient at the current iterate given by

$$\boldsymbol{p}_k = -\nabla f(\boldsymbol{x}_k).$$

This search direction is called the steepest descent direction. Using the steepest descent direction in Algorithm 1 is called the steepest descent method. While simple, steepest descent exhibits poor performance, even for simple problems. Its convergence rate is only linear and depends on the eigenvalue distribution of the Hessian $\nabla^2 f(\boldsymbol{x}^*)$ at the local minimizer $\boldsymbol{x}^*$. If the eigenvalue distribution is wide, steepest descent often requires an unacceptably large number of iterations to find a stationary point.

Newton's Method

It can be shown that any search direction $\boldsymbol{p}_k$ that makes an angle of strictly less than $\pi/2$ radians with the steepest descent direction $\nabla f(\boldsymbol{x}_k)$ is a descent direction as well. As long as $\boldsymbol{p}_k$ does not get arbitrarily close to orthogonal to $\nabla f(\boldsymbol{x}_k)$, any such $\boldsymbol{p}_k$ can be used in the line search framework. The so called Newton direction $\boldsymbol{p}_k^N$ is a popular choice. It is derived from the second-order Taylor series approximation to $f(\boldsymbol{x}_k + \boldsymbol{p})$ which is given by

$$f(\boldsymbol{x}_k + \boldsymbol{p}) \approx f(\boldsymbol{x}_k) + \boldsymbol{p}^T \nabla f(\boldsymbol{x}_k) + \frac{1}{2}\boldsymbol{p}^T \nabla^2 f(\boldsymbol{x}_k)\boldsymbol{p} =: m_k(\boldsymbol{p}). \tag{3.12}$$

The model function m_k has a unique minimizer if $\nabla^2 f(\boldsymbol{x}_k)$ is positive definite. In this case, the Newton direction is defined as the unique minimizer $\boldsymbol{p}_k^N$ of m_k, which can be found by setting the derivative of $m_k(\boldsymbol{p})$ to zero:

$$\boldsymbol{p}_k^N = -(\nabla^2 f(\boldsymbol{x}_k))^{-1} \nabla f(\boldsymbol{x_k}). \tag{3.13}$$

The better the quadratic model function $m_k(\boldsymbol{p})$ approximates $f(\boldsymbol{x}_k + \boldsymbol{p})$ around $\boldsymbol{x}_k$, the more reliable is the Newton direction.

It is easy to show that $\boldsymbol{p}_k^N$ is a descent direction, given that $\nabla^2 f(\boldsymbol{x}_k)$ is positive definite. Otherwise, the Newton direction is not guaranteed to exist, or to be a descent direction if it does. In such cases, the Newton direction cannot be used without modification. Thus, in its naive form, the Newton's method is not globally convergent. However, if $\nabla^2 f(\boldsymbol{x}^*)$ is positive definite at a local solution $\boldsymbol{x}^*$ and f is twice differentiable, then $\nabla^2 f(\boldsymbol{x})$ is also positive definite for $\boldsymbol{x} \in \mathcal{N}$ for some neighborhood $\mathcal{N}$ of $\boldsymbol{x}^*$. If we have $\boldsymbol{x}_0 \in \mathcal{N}$ for the starting point of $\boldsymbol{x}_0$ of Newton's method and $\boldsymbol{x}_0$ is sufficiently close to the solution $\boldsymbol{x}^*$ it can be shown that Newton's method with step length $\alpha_k = 1$ converges to $\boldsymbol{x}^*$ with a quadratic rate of convergence under mild conditions. Thus, Newton's method has satisfactory convergence properties close to the solution $\boldsymbol{x}^*$ and the Newton direction $\boldsymbol{p}_k^N$ has a natural step size $\alpha_k = 1$ associated with it. Since $\alpha_k = 1$ often does not satisfy the Wolfe conditions when the current iterate $\boldsymbol{x}_k$ is still far away from the solution $\boldsymbol{x}^*$, line searches are still necessary in Newton's method. However, it is recommended to use $\alpha_k = 1$ as the initial guess as $\alpha_k = 1$ guarantees quadratic convergence once $\boldsymbol{x}_k$ gets sufficiently close to $\boldsymbol{x}^*$.

A general overview over Newton's method is given in Algorithm 2. Note that practical implementations of Newton's method might deviate from the outlined algorithm. As an example, it is possible to apply positive definiteness fixes to the Hessian matrix $\nabla^2 f(\boldsymbol{x}_k)$ while computing its matrix factorization. Point out issues with positive definiteness fixes and cite [LLF+23].

Algorithm 2 Newton's Method

require $\epsilon > 0$
procedure NEWTONSMETHOD($\boldsymbol{x}_0$, ϵ)
  $\boldsymbol{x}_k = \boldsymbol{x}_0$
  while $\left\|\nabla f(\boldsymbol{x}_k)\right\| > \epsilon$ **do**
    compute $\nabla^2 f(\boldsymbol{x}_k)$
    if $\nabla^2 f(\boldsymbol{x}_k)$ is not positive definite **then**
      apply positive definiteness fix to $\nabla^2 f(\boldsymbol{x}_k)$
    end if
    $\boldsymbol{p}_k = -(\nabla^2 f(\boldsymbol{x}_k))^{-1} \nabla f(\boldsymbol{x}_k)$
    $\alpha_k = 1$
    if α_k does not satisfy the strong Wolfe conditions **then**
      compute α_k that satisfies the strong Wolfe conditions
    end if
    $\boldsymbol{x}_k = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
  end while
  return with result $\boldsymbol{x}_k$
end procedure

Despite its favorable convergence properties, Newton's method comes with a couple of disadvantages. Firstly, computing the Hessian matrix $\nabla^2 f(\boldsymbol{x}_k)$ during each iteration is often expensive. Secondly, deriving analytical expressions for the second derivatives of the objective function is a common source of bugs. Lastly, a new system

$$\nabla^2 f(\boldsymbol{x}_k) \boldsymbol{p}_k^N = -\nabla f(\boldsymbol{x_k})$$

needs to be solved at every iteration as the Hessian matrix changes with the current iterate $\boldsymbol{x}_k$. If the Hessian $\nabla^2 f(\boldsymbol{x}_k)$ is sparse, its factorization can be computed via sparse elimination techniques. However, there is no guarantee for the matrix factorization of a sparse matrix to be sparse itself in the general case. For these reasons, while a single Newton iteration often makes quite a lot of progress towards the solution, it takes a significant amount of time to compute. If $\boldsymbol{x}_k \in \mathbb{R}^n$ for some large $n \in \mathbb{N}$, computing the exact Newton iteration can become infeasible, especially for real-time applications. Concomitantly, the memory required to store the Hessian matrix of size $\mathcal{O}(n^2)$ becomes prohibitive.

Quasi-Newton Methods

Due to the shortcomings of Newton's method mentioned in Section 3.2.1, it can be favorable to simply approximate the Newton direction in order to find an ef-

fective search direction while keeping the cost of a single iteration low. Effective Newton approximations can be computed without the need to compute the Hessian $\nabla^2 f(\boldsymbol{x}_k)$ during each iteration. Often, multiple Quasi-Newton iterations fit into the same time budget as a single Newton iteration. As a result, Quasi-Newton methods can sometimes converge to a solution in a shorter amount of time than Newton's method, even though their search directions are not as effective as the exact Newton direction.

In Quasi-Newton methods, search directions of the following form are used

$$\boldsymbol{p}_k = -\mathbf{B}_k^{-1} \nabla f(\boldsymbol{x}_k), \tag{3.14}$$

where $\mathbf{B}_k \in \mathbb{R}^{n \times n}$ is positive definite. Note that the Newton direction is a special case of Equation 3.14 with $\mathbf{B}_k = \nabla^2 f(\boldsymbol{x}_k)$. Just like for the Newton direction $\boldsymbol{p}_k^N$ in Equation 3.12, a model function m_k that attains its minimum at $\boldsymbol{p}_k = -\mathbf{B}_k^{-1} \nabla f(\boldsymbol{x}_k)$ can be defined via

$$m_k(\boldsymbol{p}) = f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T \boldsymbol{p} + \frac{1}{2} \boldsymbol{p}^T \mathbf{B}_k \boldsymbol{p}. \tag{3.15}$$

As $\mathbf{B}_k \neq \nabla^2 f(\boldsymbol{x}_k)$, m_k does not correspond to a second-order Taylor approximation of f around $\boldsymbol{x}_k$ anymore. Instead, $\mathbf{B}_k$ is picked such that the gradient of m_k matches the gradient of f at the last two iterates $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k-1}$. Since $\nabla m_k(\mathbf{0}) = \nabla f(\boldsymbol{x}_k)$, the first condition is true independent of $\mathbf{B}_k$. The second condition yields

$$\nabla m_k(-\alpha_{k-1} \boldsymbol{p}_{k-1}) = \nabla f(\boldsymbol{x}_k) - \alpha_{k-1} \mathbf{B}_k \boldsymbol{p}_{k-1} = \nabla f(\boldsymbol{x}_{k-1}).$$

Rearranging gives

$$\mathbf{B}_k \boldsymbol{s}_{k-1} = \boldsymbol{y}_{k-1}, \tag{3.16}$$

where $\boldsymbol{s}_{k-1} = \boldsymbol{x}_k - \boldsymbol{x}_{k-1} = \alpha_{k-1} \boldsymbol{p}_{k-1}$. This is called the secant equation. Multiplying both sides from the left with $\boldsymbol{s}_{k-1}^T$ yields the curvature condition given by

$$\boldsymbol{s}_{k-1}^T \boldsymbol{y}_{k-1} > 0, \tag{3.17}$$

since $\mathbf{B}_k$ is positive definite. Note that the curvature condition is not satisfied for arbitrary $\boldsymbol{x}_k, \boldsymbol{x}_{k-1}$ if f is not convex. However, it can be shown that the curvature condition always holds when the step size α_{k-1} satisfies the strong Wolfe conditions. Thus, a proper line search strategy is vital for the viability of Quasi-Newton methods.

Since $\mathbf{B}_k$ is positive definite, the secant equation can be written in terms of the inverse $\mathbf{H}_k \coloneqq \mathbf{B}_k^{-1}$ as

$$\mathbf{H}_k \boldsymbol{y}_{k-1} = \boldsymbol{s}_{k-1}$$

and the formula for the new search direction becomes $-\mathbf{H}_k \nabla f(\boldsymbol{x}_k)$. The secant equation is not enough to uniquely determine the entries of $\mathbf{H}_k$, even if $\mathbf{H}_k$ is required to be symmetric positive definite. Thus, the additional requirement that $\mathbf{H}_k$ is closest to $\mathbf{H}_{k-1}$ according to some norm is imposed. In summary, $\mathbf{H}_k$ is picked such that it solves the following constrained minimization problem

$$\min_H \|\mathbf{H} - \mathbf{H}_{k-1}\|, \text{ subject to } \mathbf{H} = \mathbf{H}^T \text{ and } \mathbf{H}\boldsymbol{y}_{k-1} = \boldsymbol{s}_{k-1}.$$

Using a scale-invariant version of the weighted Frobenius norm gives rise to the popular Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm. It is defined via the following update formula for $\mathbf{H}_k$

$$\mathbf{H}_k = (I - \rho_{k-1}\boldsymbol{s}_{k-1}\boldsymbol{y}_{k-1}^T)\mathbf{H}_{k-1}(I - \rho_{k-1}\boldsymbol{s}_{k-1}\boldsymbol{y}_{k-1}^T) + \rho_{k-1}\boldsymbol{s}_{k-1}\boldsymbol{s}_{k-1}^T, \quad (3.18)$$

where $\rho_{k-1} = 1/(\boldsymbol{s}_{k-1}^T \boldsymbol{y}_{k-1})$. Is is possible to give a similar update formula in terms of $\mathbf{B}_k$. Generally, using the formulation in terms of the inverse matrices $\mathbf{H}_k$ is preferrable since the computation of the new descent direction can be achieved by simple matrix-vector multiplication instead of solving a linear system if $\mathbf{B}_k$ is maintained instead. An overview over the algorithm is given in Algorithm 3.

While global convergence of the BFGS method cannot be established for general nonlinear smooth functions, it is possible to show that it converges superlinearly if the initial guess $\boldsymbol{x}_0$ is close to the solution $\boldsymbol{x}^*$ and $\alpha_k = 1$ for sufficiently large k under mild conditions. Thus, just like Newton's method (Section 3.2.1), the BFGS method has a natural step length $\alpha = 1$, which should be the initial guess for all line search algorithms. Typically, the BFGS method dramatically outperforms steepest descent and performs comparably to Newton's method on many practical problems.

The behavior of the BFGS method depends on the choice of the initial inverse matrix $\mathbf{H}_0$. One obvious choice is $\mathbf{H}_0 = \nabla^2 f(\boldsymbol{x}_0)$. However, there is no guarantee that $\nabla^2 f(\boldsymbol{x}_0)$ is positive definite. Additionally, computing even a single inverse matrix can be prohibitively expensive for large problems. Thus, scaled versions of the identity matrix $\gamma I, \gamma \in \mathbb{R}^+$ are often used instead. There is no good general strategy for choosing γ, even though heuristic appraoches are popular. Maybe explain one heuristic, if necessary down the line.

Even though BFGS iterations are typically faster to compute than Newton iterations, the BFGS method is still not suitable for large problems in its naive form. Just like in Newton's method, either $\mathbf{H}_k$ or $\mathbf{B}_k$ needs to be stored explicitly, which can be infeasible for large-scale problems. While the BFGS update

Algorithm 3 BFGS method

require $\mathbf{H}_0$ symmetric positive definite, $\epsilon > 0$
procedure BFGS($\boldsymbol{x}_0$, $\mathbf{H}_0$, ϵ)
  $\boldsymbol{x}_k, \boldsymbol{x}_{k-1} = \boldsymbol{x}_0, \mathbf{H}_k = \mathbf{H}_0$
  while $\left\|\nabla f(\boldsymbol{x}_k)\right\| > \epsilon$ **do**
    $\boldsymbol{s} = \boldsymbol{x}_k - \boldsymbol{x}_{k-1}, \boldsymbol{y} = \nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_{k-1}), \rho = 1/(\boldsymbol{s}^T\boldsymbol{y})$
    $\mathbf{H}_k = (I - \rho \boldsymbol{s}\boldsymbol{y}^T)\mathbf{H}(I - \rho \boldsymbol{s}\boldsymbol{y}^T) + \rho \boldsymbol{s}\boldsymbol{s}^T$
    $\boldsymbol{p}_k = -\mathbf{H}_k \nabla f(\boldsymbol{x}_k)$
    $\alpha_k = 1$
    if α_k does not satisfy the strong Wolfe conditions **then**
      compute α_k that satisfies the strong Wolfe conditions
    end if
    $\boldsymbol{x}_{k-1} = \boldsymbol{x}_{\boldsymbol{k}}$
    $\boldsymbol{x}_k = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
  end while
  return with result $\boldsymbol{x}_k$
end procedure

formula using the inverse matrices $\mathbf{H}_k$ replaces the need for a matrix factorization with a simple matrix-vector multiplication, $\mathbf{H}_k$ and $\mathbf{B}_k$ are generally dense, even if $\nabla^2 f(\boldsymbol{x}_k)$ is sparse. This removes the possibility of alleviating storage requirements and speeding up computations via sparse matrix techniques when using the naive BFGS method.

Limited-Memory Quasi-Newton Methods

Fabian's Feedback: Although this is used to some extent by the PD papers, I think this section (possibly also the one before) goes into a bit too much detail. If you want this written down for your own understanding maybe move it to an Appendix? But in general I would recommend for the foundations to focus a bit more on what properties are relevant for what you are doing later. What you need to know about performance, complexity, convergence etc. Not so much how each step of the (L)BFGS algorithm works. In the end, it's just a building block like a matrix factorization. Without knowing all the details I would say that something between a half and full page should suffice on LBFGS (with references to the details of course)

As discussed in Section 3.2.1, the BFGS method is unsuitable for large-scale problems due to the storage requirements of the typically dense inverse Hessian approximation $\mathbf{H}_k$. This highlights the need for effective Hessian approximations that are not only cheap to compute, but also cheap to store. Just like Quasi-

Newton methods use approximations of the Newton direction in order to keep the computational cost of a single iteration low, limited-memory Quasi-Newton methods approximate Quasi-Newton directions with the goal of reducing the memory footprint of a single iteration. This comes at the prize of inferior convergence properties. On the upside, limited-memory Quasi-Newton directions can sometimes be computed by using only a couple of vectors of size n, without the need to explicitly form the inverse Hessian approximation $\mathbf{H}_k$. This can drop the space complexity of a single iteration to $\mathcal{O}(n)$ compared to $\mathcal{O}(n^2)$ for the BFGS method.

A popular limited-memory method called L-BFGS can be derived from the BFGS update formula for the inverse Hessian approximation $\mathbf{H}_k$ (Eq. (3.18)). Note that the BFGS update in iteration k is specified entirely by the vector pair $(\boldsymbol{s}_n, \boldsymbol{y}_n) \in \mathbb{R}^n$. Consequently, $\mathbf{H}_k$ can be constructed from the initial matrix $\mathbf{H}_0$ and the familiy of vector pairs $((\boldsymbol{s}_i, \boldsymbol{y}_i))_{i\in[0,k-1]}$ by simply performing k update steps according to $Equation\ 3.18$. The idea of L-BFGS is to only keep track of the most recent m vector pairs and generate a modified version of the inverse Hessian approximation from the BFGS method by applying the m updates defined by $((\boldsymbol{s}_i, \boldsymbol{y}_i))_{i\in[k-m,k-1]}$ to the initial matrix $\mathbf{H}_0$ at each iteration k.

It is important to note that its not the L-BFGS Hessian approximation $\mathbf{H}_k$ itself, but the search direction $\boldsymbol{p}_k = -\mathbf{H}_k \nabla f(\boldsymbol{x}_k)$ that is of interest. It turns out that the L-BFGS search direction $\boldsymbol{p}_k$ can be computed without explicitly constructing $\mathbf{H}_k$ using an algorithm called the L-BFGS two-loop recursion (Algorithm 4). This algorithm can be specified in terms of the initial Hessian approximation $\mathbf{B}_0$ with minor changes, as indicated by line 6. To simplify the notation, the entire history of $\boldsymbol{s} = (\boldsymbol{s}_i)_{i\in[0,k]}, \boldsymbol{y} = (\boldsymbol{y}_i)_{i\in[0,k]}, \rho = (\rho_i)_{i\in[0,k]}$ is passed to TWOLOOPRECURSION, even though at most the m most recent values are needed.

Excluding the matrix-vector multiplication $\mathbf{H}_0\boldsymbol{t}$, the two-loop recursion scheme has time complexity $\mathcal{O}(nm) = \mathcal{O}(n)$ since $m << n$. Thus, if $\mathbf{H}_0$ is chosen to be diagonal, the entire L-BFGS iteration can be computed in $\mathcal{O}(n)$. Similarly, the space complexity of the L-BFGS iteration is $\mathcal{O}(n)$ if $\mathbf{H}_0$ is diagonal. Even if $\mathbf{H}_0$ is not diagonal, but sparse, the two-loop recursion can be significantly faster and more space efficient than a BFGS update where matrix-vector multiplication with a dense matrix is required in general. Note that the same is not necessarily true if $\mathbf{B}_0$ is sparse, but not diagonal. In this case, a factorization of $\mathbf{B}_k$ needs to be computed which is not guaranteed to stay sparse.

An overview over the entire L-BFGS algorithm is given in Algorithm 5, where the details of maintaining the history of $\boldsymbol{s}, \boldsymbol{y}, \rho$ are omitted for the sake of clarity.

L-BFGS shares many properties with the BFGS method discussed in Section 3.2.1. The performance of the L-BFGS method depends on the choice of the initial matrix $\mathbf{H}_0$, with scaled diagonal matrices being popular choices. Again, there is no generally viable strategy for picking the scaling factor $\gamma \in \mathbb{R}$. Similarly, the initial guess for the step size $\alpha_k = 1$ should be used. The window

Algorithm 4 L-BFGS two-loop Recursion

require $\mathbf{H}_0$ or $\mathbf{B}_0$ symmetric positive definite
procedure TWOLOOPRECURSION($\mathbf{H}_0$ or $\mathbf{B}_0$, $\boldsymbol{x}_k$, $\boldsymbol{s}$, $\boldsymbol{y}$, ρ, m, k)
    $m^* = \min(m, k), \boldsymbol{t} = -\nabla f(\boldsymbol{x}_k)$
    for $i = k-1, k-2, \ldots, k-m^*$ **do**
        $\alpha_i = \rho_i \boldsymbol{s}_i^T \boldsymbol{t}$
        $\boldsymbol{t} = \boldsymbol{t} - \alpha_i \boldsymbol{y}_i$
    end for
    $\boldsymbol{r} = \mathbf{H}_0 \boldsymbol{t}$ or solve $\mathbf{B}_0 \boldsymbol{r} = \boldsymbol{t}$
    for $i = k-m^*, k-m^*+1, \ldots, k-1$ **do**
        $\beta = \rho_i \boldsymbol{y}_i^T \boldsymbol{r}$
        $\boldsymbol{r} = \boldsymbol{r} + \boldsymbol{s}_i(\alpha_i - \beta)$
    end for
    return with result -$\mathbf{H}_k \nabla f(\boldsymbol{x}_k) = \boldsymbol{r}$.
end procedure

Algorithm 5 L-BFGS method

require $\mathbf{H}_0$ symmetric positive definite, $\epsilon > 0$
procedure LBFGS($\boldsymbol{x}_0$, $\mathbf{H}_0$, m, ϵ)
    $\boldsymbol{x}_k = \boldsymbol{x}_0, \mathbf{H}_k = \mathbf{H}_0, k = 0$
    while $\left\|\nabla f(\boldsymbol{x}_k)\right\| > \epsilon$ **do**
        $\boldsymbol{s_k} = \boldsymbol{x}_k - \boldsymbol{x}_{k-1}, \boldsymbol{y}_k = \nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_{k-1}), \rho_k = 1/(\boldsymbol{s}_k^T \boldsymbol{y}_k)$
        $\boldsymbol{p}_k =$ TWOLOOPRECURSION($\mathbf{H}_0, \boldsymbol{x_k}, \boldsymbol{s}, \boldsymbol{y}, \boldsymbol{\rho}, m, k$) (Algorithm 4)
        $\alpha_k = 1$
        if α_k does not satisfy the strong Wolfe conditions **then**
            compute α_k that satisfies the strong Wolfe conditions
        end if
        $\boldsymbol{x}_k = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k, k = k+1$
    end while
    return with result $\boldsymbol{x}_k$
end procedure

size m is a parameter that needs to be tuned on a per-problem basis. While the L-BFGS algorithm is generally less robust if m is small, making m arbitrarily large increases the amount of time required to perform the two-loop recursion. If the matrix-vector multiplication in Algorithm 4 is expensive to compute, the additional computational cost incurred by increasing m is usually overshadowed by the matrix-vector multiplication. Still, larger values of m do not necessarily lead to better performance. [LBK17] suggest that curvature information from vector pairs $(\boldsymbol{s}_i, \boldsymbol{y}_i)$ from iterations i with $i << k$ can become out of date, making moderately large values of m more beneficial. The main weakness of the L-BFGS method is its slow convergence on problems where the true Hessian matrices $\nabla^2 f(\boldsymbol{x})_k$ are ill-conditioned.

Step Length Selection Algorithms

In Section 3.2.1, the need for step lengths α_k that satisfy the strong Wolfe conditions (Eq. (3.10), Eq. (3.11)) for the convergence of line search methods was discussed. Appropriate step lengths are determined via step length selection algorithms. These algorithms are typically split into two phases. The bracketing phase determines an interval $[\alpha_{\text{min}}, \alpha_{\text{min}}]$ that is guaranteed to contain suitable step lengths. The selection phase is an iterative process that interpolates function values and gradients from previous iterations in order to shrink the interval and eventually pick the final step length. For more details, the reader is referred to Chapter 3 of. As step length algorithms are a common source of bugs, Nocedal and Wright recommend using publically available implementations.

To avoid the complexities of correct step length algorithms, the insight that the Armijo condition (Eq. (3.10)) is always satisfied once α is sufficiently close to zero (Section 3.2.1) can be exploited: If a good first estimate $\alpha = \tilde{\alpha}$ is available, we check whether it satisfies the Armijo condition. Otherwise, α is gradually decreased until sufficient decrease is satisfied or until it falls below a predefined threshold. The idea is that the resulting step length will often satisfy the second Wolfe condition automatically as long as the initial estimate $\tilde{\alpha}$ is a well-informed guess and step lengths are not decreased too rapidly. For Newton's method and Quasi-Newton methods, usually $\tilde{\alpha} = 1$ is used for the best results. This approach is known as backtracking and is outlined in Algorithm 6. Here, c_1 is the constant factor from the Armijo condition.

Backtracking is much simpler to implement than correct step length algorithms. Additionally, each iteration of the backtracking algorithm only requires the computation of a single function evaluation. Thus, if function evaluations are cheaper than gradient evaluations backtracking is also more efficient. However, backtracking does not provide a guarantee that the final step length satisfies the curvature condition. If the search direction $\boldsymbol{p}_k$ is effective, e.g. when Newton's

Algorithm 6 Backtracking Line Search

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

method is used, this tradeoff is often justifiable. For less effective search directions, including search directions from most Quasi-Newton methods, backtracking might be less suitable.

3.3 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 bodies 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 3.1.2 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 in order to fit into the equations of motion in Eq. (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 the linear implicit Euler stepper introduced in Section 3.1.4. While both XPBD (Section 3.4.4) and PD (Section 3.5) differ significantly from the linear implicit Euler stepper, the challenges encountered in modelling hard constraints are largely transferrable. Handling hard constraints via stiff spring

forces as outlined above is covered in more detail in Section 3.3.1. Combining the system of ODEs from the equations of motion with hard constraints into a Differential Algebraic Equation (DAE) without introducing stiff constraint forces is explored in Section 3.3.3. Finally, adding regularization to the resulting DAE in a physcially meaningful manner yields a method called compliant constraints. Compliant constraints are introduced in Section 3.3.4.

- Probably best to skip this here. Make sure this is mentioned in the discussion of XPBD, PD, etc.
- generality. it is convenient to handle different effects in a unified manner. for example, it should be possible to handle the elasticity of the simulated body due to its material model, constraints between bodies and contact in roughly the same way.
- simplicity. prevents bugs, easy to model new effects, etc. this is mostly a practical consideration, but still relevant.
- Use $\boldsymbol{q}$ instead of $\boldsymbol{x}$ everywhere
- Rewrite this entire section from scratch. Make sure that the notation is unified. At the end of each method, point out what its issues are. When the next method is introduced that solves some of these issues, highlight how with a couple of sentences.

3.3.1 Stiff Springs

Common effects in elasticity-based simulations such as attaching one body to another or maintaining fixed distance between two particles can be approximated via stiff springs. Consider a spring between

High stiffness values lead to large forces which in turn cause numerical issues in the solver.

We demonstrate these issues based on the example of maintaining a desired distance between two points using a stiff spring [TNGF15]. Let $\boldsymbol{x_1}, \boldsymbol{x_2}$ be the positions, $\boldsymbol{v}_1, \boldsymbol{v}_2$ the velocities and $\boldsymbol{a}_1, \boldsymbol{a}_2$ be the accelerations of the two particles. Let $\bar{l}$ be the rest length and $l = \|\boldsymbol{x_1} - \boldsymbol{x_2}\|$ be the current length of the spring with stiffness k. It can be shown that the force that the spring applies at each particle is equal to $\boldsymbol{f_1} = -\boldsymbol{f_2} = \lambda \boldsymbol{u}$, where $\boldsymbol{u} = (\boldsymbol{x_1} - \boldsymbol{x_2})/l$ and $\lambda = -\frac{\delta V}{\delta l} = k(\bar{l} - l)$.

Once the forces, accelerations, velocities and positions are combined into vectors $\boldsymbol{f}, \boldsymbol{a}, \boldsymbol{v}, \boldsymbol{x}$, respectively, the motions of the system can be modeled via Newton's Ordinary Differential Equation (ODE) $\boldsymbol{f} = \mathbf{M}\boldsymbol{a}$, where $\mathbf{M}$ is a $n_d \times n_d$

diagonal matrix and n_d is the total number of independent degrees of freedom for the particles.

This system can be integrated via the symplectic Euler method as follows (I believe this should be moved into the section on numerical integration...):

$$\boldsymbol{v}_{n+1} = \boldsymbol{v}_n + h\boldsymbol{a}_n$$
$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + h\boldsymbol{v}_{n+1}$$

As the stiffness k of the spring increases, so does the magnitude of the acceleration $\boldsymbol{a}$. Consequently, the integration diverges unless the time step is prohibitively small. The stability issues are often addressed by switching to an implicit integration scheme, such as the backward Euler method [BW98] (refer to the section on numerical integration here). Replacing current accelerations with future accelerations requires the solution of the following linear system of equations (LSE):

$$(\mathbf{M} - h^2\mathbf{K})\boldsymbol{v_{n+1}} = \boldsymbol{p} + h\boldsymbol{f}$$

where $\boldsymbol{p} = \mathbf{M}\boldsymbol{v}_n$ is the momentum, and $\mathbf{K} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$ is the stiffness matrix. Note that $\mathbf{K}$ is typically non-singular since elastic forces are invariant under rigid body transforms. When using large stiffness k for springs, the entries of $\mathbf{K}$ are large (due to large restorative forces for stiff springs) and dominate the entries of the system matrix

$$\mathbf{H} = \mathbf{M} - h^2\mathbf{K}. \tag{3.19}$$

In these cases, $\mathbf{H}$ will be almost non-singular as well, leading to numerical issues and poor convergence for many solvers. Additionally, implicit integration introduces noticable numerical damping [SLM06].

This system results from performing the implicit integration and solving the non-linear system via linearization using the Taylor expansion. Positions can be expressed in terms of velocities and eliminated from the system.

3.3.2 Penalty Forces

In Section 3.3.1, the energy was derived from Hooke's Law for springs. However, it is also possible to derive energies from geometric displacement functions $\boldsymbol{\phi}(\boldsymbol{x})$ which vanish in the rest configuration. From the displacement functions, quadratic potential energies of the form $U(\boldsymbol{x}) = \Sigma_i(k/2)\boldsymbol{\phi}^2(x)$, where k is a positive stiffness parameter, are constructed [TPBF87]. The potential energy $U(\boldsymbol{x})$ is zero if the displacement function is satisfied, and greater than zero otherwise. The

resulting forces are called penalty forces. Make sure to be consistent with naming of potentials across the thesis.

Using the geometric displacement function $\phi_{\text{spring}}(\boldsymbol{x}) = (\|\boldsymbol{x_i} - \boldsymbol{x_j}\|) - l$ with k_{spring} recovers the behavior of a spring with stiffness k_{spring} (Section 3.3.1). Its displacement function $\phi_{\text{spring}}(\boldsymbol{x})$ is satisfied when the distance of the particles $\boldsymbol{x_i}, \boldsymbol{x_j}$ is equal to a desired rest length l. 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]. Additionally, the generated penalty forces do not converge in the limit of infinite stiffess, leading to oscillations unless the time step is reduced significantly [RU57].

Maybe explain the challenges with penalty forces a bit better! Also read [TPBF87; NMK+06; RU57]. I just skimmed over [TPBF87] for now, but want to make sure that I am citing this correctly. The term penalty forces is not used in the paper, I am just following the trail from [SLM06]. [NMK+06] is a review that might be intersting to read. [RU57] would be really interesting to read for once, just to understand why strong penalty forces oscillate. Is this a general problem with penalty forces, or is it an issue with the solver?

3.3.3 Hard Constraints

The problem of maintaining hard distance constraints between particles can be formulated as a Differential Algebraic Equation (DAE) [UR95; Bar96]. In this framework, the equations of motion (reference somewhere) are handled together with algebraic equations that model the constraints on the positions of the system. Distance constraints are typically implemented using holonomic constraints of the form $\phi(\boldsymbol{x}) = 0$. Note that the distance constraint $\phi(x)$ is formulated in terms of the particle positions, whereas the equations of motion work on velocities and accelerations. Consequently, the constraints need to be differentiated with respect to time once or twice so that they can be combined with the ODE in terms of velocties or accelerations, respectively. In XPBD, we go the other way! The ODE is tanslated so that it is in terms of positions, so that it can be handled together with the constraints. Is there a reason nobody bothered to do this before? Fabian notes that the translation in terms of positions in XPBD is in fact just rewriting the implicit Euler integration in terms of positions. Thus, we have baked the integration method into the formulation in XPBD. In the DAE framework, there

is more flexibility w.r.t. the integration scheme of choice. Using $\mathbf{J} = \frac{\delta \phi}{\delta \boldsymbol{x}}$, where $\mathbf{J}$ is a $n_c \times n_d$ matrix and n_c is the number of scalar constraints, this leads to the following constraint formulations:

$$\mathbf{J}\boldsymbol{v} = 0$$
$$\mathbf{J}\boldsymbol{a} = \boldsymbol{c}(\boldsymbol{v})$$

for some $\boldsymbol{c}(\boldsymbol{v})$. If you check [UR95], see that $c(v)$ also depends on the positions q. That should be indicated! Additionally, constraint forces (use internal forces, more general and will be used throughout the thesis) are required in order to link the algebraic constraint equations with the ODE describing the motion of the system. It can be shown that the constraint forces $\boldsymbol{f_c}$ applied to the particles have to be in the following form in order to avoid adding linear and angular momentum to the system [Bar96]:

$$\boldsymbol{f_c} = \mathbf{J}^T \boldsymbol{\lambda} \tag{3.20}$$

where the λ are the Lagrange multipliers of the constraints. With external forces $\boldsymbol{f}_{\text{ext}}$, the DAE can now be expressed as follows [UR95]:

$$\begin{pmatrix} \mathbf{M} & -\mathbf{J}^T \\ \mathbf{J} & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{a} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f_e} \\ \boldsymbol{c}(\boldsymbol{v}) \end{pmatrix}$$

Note that the lower block-row of the system drives towards accelerations that satisfy the constraints imposed by $\boldsymbol{\phi}(\boldsymbol{x})$ (or, stricly speaking, the differentiations thereof) exactly. This is indicated by the lower-right zero block in the system matrix in either formulation. Thus, the system does not have a solution if constraints are contradictory. Aren't $\dot{q} = v$ and $\dot{v} = a$ also part of the differential equation? Because $c(v)$ and f_e also depend on q!

In [UR95], the DAE is approached by eliminating the λ from the system entirely and constructing an ODE in terms of positions and velocities. In [TNGF15], the authors suggest applying implicit integration schemes to the system directly by constructing the following Karush-Kuhn-Tucker (KKT) equation system:

$$\begin{pmatrix} \mathbf{M} & -\mathbf{J}^T \\ \mathbf{J} & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{v}_{n+1} \\ \boldsymbol{\mu}_{n+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{p} + h\boldsymbol{f_e} \\ 0 \end{pmatrix}$$

Here, the external forces $\boldsymbol{f}_{\text{ext}}$ and the constraint gradients $\mathbf{J}$ are considered constant across the time step and $\mathbf{J}(x_{n+1})$ is not approximated using the 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 3.19.

Reverse-engineering how the authors arrived at this system is quite enlightening. Start out from the equations of motion [UR95]

$$\dot{\boldsymbol{v}} = \mathbf{M}^{-1}(\boldsymbol{f} - \mathbf{J}^T\boldsymbol{\lambda})$$

and perform implicit integration:

$$\begin{aligned}
\boldsymbol{v}_{n+1} &= \boldsymbol{v}_n + h\mathbf{M}^{-1}(\boldsymbol{f}_e(\boldsymbol{X}_{n+1}) - \mathbf{J}^T(\boldsymbol{x}_{n+1})\boldsymbol{\lambda}(\boldsymbol{x}_{n+1})) \\
\mathbf{M}\boldsymbol{v}_{n+1} &= \boldsymbol{p} + h\boldsymbol{f}_e(\boldsymbol{x}_{n+1}) - h\mathbf{J}^T(\boldsymbol{x}_{n+1})\boldsymbol{\lambda}(\boldsymbol{x}_{n+1}) \\
\mathbf{M}\boldsymbol{v}_{n+1} + h\mathbf{J}^T(\boldsymbol{x}_{n+1})\boldsymbol{\lambda}(\boldsymbol{x}_{n+1}) &= \boldsymbol{p} + h\boldsymbol{f}_e(\boldsymbol{x}_{n+1}) \\
\mathbf{M}\boldsymbol{v}_{n+1} + \mathbf{J}^T(\boldsymbol{x}_{n+1})\boldsymbol{\mu}(\boldsymbol{x}_{n+1}) &= \boldsymbol{p} + h\boldsymbol{f}_e(\boldsymbol{x}_{n+1})
\end{aligned}$$

If we assume that $\boldsymbol{f}_e$ and the constraint gradients $\mathbf{J}$ are constant across the time step, we arrive at the formulation from the paper. For the external forces, which are usually only comprised of gravitational forces, this is not a big deal. For the constraint gradients, I am not sure what the ramifications are. In [BW98], the Taylor expansion is performed which requires the compution of second derivatives over the constraint functions. This is not happening here at all! Is this what authors mean when they say that the constraints are effectively linearized during one solve, e.g. second page of [MMC+20]? Technically, speaking, even if the Taylor expansion is performed, the constraints are linearized, if I understand correctly.

Note that the system matrix is sparse, which can be exploited by sparse-matrix solvers in order to solve the system efficiently [Bar96]. Alternatively, the Schur complement can be constructed since the mass matrix in the upper left block is invertible. This leads to a smaller, albeit less sparse system [TNGF15]:

$$\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\mu} = -\mathbf{J}\mathbf{M}^{-1}(\boldsymbol{p}) + h\boldsymbol{f}_e)$$

If the constraints are not redundant, $\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T$ is non-singular and symmetric positive definite [Bar96], which are desirable properties for many solvers. According to [SLM06], the common approaches for linearizing the constraint forces and stabilizing the constraints $\phi(\boldsymbol{x}) = 0$ are notoriously unstable (I need to look this up again. I do not understand what exactly this means anymore). 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].

3.3.4 Compliant Constraints

By combining ideas from hard constraints (Section 3.3.3) and penalty forces (Section 3.3.2), 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 (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:

$$\psi(\boldsymbol{q}) = \frac{1}{2}\boldsymbol{C}(\boldsymbol{q})^T\alpha^{-1}\boldsymbol{C}(\boldsymbol{q}) \tag{3.21}$$

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

$$\boldsymbol{f_c} = \nabla\psi(\boldsymbol{q}) = -\nabla\boldsymbol{C}(\boldsymbol{q})^T\alpha^{-1}\boldsymbol{C}(\boldsymbol{q}). \tag{3.22}$$

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

$$\boldsymbol{f}_c = \nabla\boldsymbol{C}(\boldsymbol{q})^T\boldsymbol{\lambda} \tag{3.23}$$

This leads to the following DAE:

$$\begin{aligned} \dot{\boldsymbol{q}} &= \boldsymbol{v} \\ \mathrm{M}\dot{\boldsymbol{v}} &= \boldsymbol{f}_e + \nabla\boldsymbol{C}(\boldsymbol{q})^T\boldsymbol{\lambda} \\ \alpha\boldsymbol{\lambda}(\boldsymbol{q}) &= -\boldsymbol{C}(\boldsymbol{q}) \end{aligned} \tag{3.24}$$

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

Usually, the DAE is solved by employing some numerical integration scheme (Section 3.1.2) 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

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

leading to the following LSE [TNGF15]

$$\begin{pmatrix} \mathbf{M} & -\nabla\boldsymbol{C}(\boldsymbol{q}_n)^T \\ \nabla\boldsymbol{C}(\boldsymbol{q}_n) & \frac{1}{h^2}\alpha \end{pmatrix} \begin{pmatrix} \boldsymbol{v}_{n+1} \\ \boldsymbol{\mu}_{n+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{p} + h\boldsymbol{f}_e \\ -\frac{1}{h}\boldsymbol{C}(\boldsymbol{q}_n) \end{pmatrix}, \tag{3.25}$$

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 (reference), the lower-right zero block is replaced by the compliance matrix α in Equation 3.25 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 ((reference to section)) [TNGF15]. Secondly, setting $\alpha = \mathbf{0}$ allows modelling infinite stiffness since α^{-1} does not occur in Equation 3.25. Additionally, Tournier et al. [TNGF15] claim that, in comparison to penalty forces where high stiffness makes the system matrix in (reference equation) almost singular, decreasing the compliance terms in the lower-right block of the system matrix in Equation 3.25 makes working with high stiffness numerically tractable. However, close inspection of the system matrix of Equation 3.25 shows that it quickly becomes singular for high stiffnesses as well when the lower-right block containing the corresponding compliances approaches zero.

Fabian: Does it? Is it singular if a = 0? Isn't it possibly non-singular depending on grad C? Or am I missing something? Maybe explain more "your close inspection".

3.4 Position Based Dynamics

As discussed in Section 3.1.1, 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 (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 3.4.1. The details of individual constraint projection in PBD and XPBD are covered in Section 3.4.2 and Section 3.4.4, respectively.

3.4.1 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 $\nabla_{\boldsymbol{p}_l} C_j(\boldsymbol{q}) = \mathbf{0}$.

An overview over the PBD framework is given in Algorithm 7 [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, non-linear constraints, moving the initial estimates from the symplectic Euler integration to positions that satisfy the constraints requires solving a non-linear 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 non-linear 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.

Due to the fact that PBD is a geometrical method that is not derived from Newton's laws of motion (Section 3.4.2) 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 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 (Section 3.4.4). Thus PBD and XPBD projections are performed along

Algorithm 7 Position Based Dynamics Framework

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

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 3.4.2 and Section 3.4.4 and the resulting algorithms for projecting all constraints acting on simulated bodies are given in Algorithm 8 and Algorithm 9, respectively.

3.4.2 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

$$C(\boldsymbol{p} + \Delta\boldsymbol{p}) = 0. \tag{3.26}$$

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

$$\Delta\boldsymbol{p} = \lambda \mathbf{W} \nabla C(\boldsymbol{p}) \tag{3.27}$$

for some $\lambda \in \mathbb{R}$ and $\mathbf{W} = diag(w_1, w_1, w_1, \ldots, w_m, w_m, w_m)$. Plugging into $Equation$ 3.26 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

$$\lambda = -\frac{C(\boldsymbol{p})}{\sum_{i \in \{i_1, \ldots, i_{n_c}\}} w_i |\nabla_{\boldsymbol{p}_i} C(\boldsymbol{p})|^2}. \tag{3.28}$$

Plugging λ into Equation 3.27 results in the final position update

$$\Delta \boldsymbol{p} = -\frac{C(\boldsymbol{p})}{\sum_{i\in\{i_1,\dots,i_{n_c}\}} w_i |\nabla_{\boldsymbol{p}_i} C(\boldsymbol{p})|^2} \mathbf{W} \nabla C(\boldsymbol{p}). \tag{3.29}$$

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

$$\Delta \boldsymbol{p}_i = -\frac{C(\boldsymbol{p})}{\sum_{i\in\{i_1,\dots,i_{n_c}\}} w_i |\nabla_{\boldsymbol{p}_i} C(\boldsymbol{p})|^2} w_i \nabla_{\boldsymbol{p}_i} C(\boldsymbol{p}) \tag{3.30}$$

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 non-linear. Consider a distance constraint with rest length 0 acting on predictions $\boldsymbol{p}_{i_1}, \boldsymbol{p}_{i_2}$ given by

$$C_{\text{dist}}(\boldsymbol{p}) = |\boldsymbol{p}_{i_1} - \boldsymbol{p}_{i_2}|. \tag{3.31}$$

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 8.

Algorithm 8 PBD Constraint Solver

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

3.4.3 Properties of PBD

Due to its simplicity and controllability, PBD is a popular choice for real-time simulations where visually plausible results are sufficient. At the core of the PBD algorithm is the non-linear 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 8 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 different colors are guaranteed to work on separate sets of particles can be employed (citation needed!). 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.

The position update due to a single constraint C in Equation 3.29 is related to the Newton-Raphson method for finding roots of non-linear 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 (Citation needed? Falls into the curriculum of a B.Sc.)

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_0)}. \tag{3.32}$$

Indeed, applying the Newton-Raphson update to

$$f\colon \mathbb{R} \to \mathbb{R}, \lambda \mapsto C(\boldsymbol{p} + \lambda \mathbf{W} \nabla C(\boldsymbol{p})) \tag{3.33}$$

yields

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

and with $\lambda_0 = 0$ we arrive at

$$\lambda_1 = -\frac{C(\boldsymbol{p})}{\nabla^T C(\boldsymbol{p}) \mathbf{W} \nabla C(\boldsymbol{p})} = -\frac{C(\boldsymbol{p})}{\sum_{i \in \{i_1, \dots, i_{n_c}\}} w_i |\nabla_{\boldsymbol{p}_i} C(\boldsymbol{p})|^2}.$$

Note that this is exactly the same as the λ used in the constraint solver given in Equation 3.28. Thus, a single constraint projection corresponds to the first iteration of the Newton-Raphson method applied to Equation 3.33 with $\lambda_0 = 0$. The correspondence breaks down if $\lambda_0 \neq 0$ or if multiple position updates are performed consecutively for the same constraint.

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 7. 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 3.30 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 and constraint projections that are performed towards the end might undo progress achieved by previous projections. Additionally, we have shown above that a single constraint projection corresponds to the first iteration of a Newton-Raphson method with initial guess $\lambda_0 = 0$. For highly non-linear constraints, it cannot be expected that the positions are moved onto or even close to the constraint manifold of interest after a single linearization. Lastly, for general non-linear constraints, it is the shape of the constraint at the current configuration that matters for the stability of the position update. Whether a Newton-Raphson iteration is effective or not cannot be answered for a function – or in this case for a constraint – in its entirety, but in the proximity of specific values.

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 (Eq. (3.31)). As discussed in Section 3.4.2, 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.

While the simulated object's global linear and angular momentum are preserved, the linear momentums 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 3.30 it is only the ratio of the particle masses that matters. This can be seen by multiplying all inverse masses w_i in Equation 3.30 with a constant factor $a \in \mathbb{R}^+$:

$$\begin{aligned}\Delta \boldsymbol{p}_i &= -\frac{C(\boldsymbol{p})}{\sum_{j\in\{i_1,\dots,i_{n_c}\}} aw_j|\nabla_{\boldsymbol{p}_j}C(\boldsymbol{p})|^2} aw_i\nabla_{\boldsymbol{p}_i}C(\boldsymbol{p}) \\ &= -\frac{C(\boldsymbol{p})}{\sum_{j\in\{i_1,\dots,i_{n_c}\}} \frac{aw_j}{aw_i}|\nabla_{\boldsymbol{p}_j}C(\boldsymbol{p})|^2}\nabla_{\boldsymbol{p}_i}C(\boldsymbol{p}) \\ &= -\frac{C(\boldsymbol{p})}{\sum_{j\in\{i_1,\dots,i_{n_c}\}} \frac{w_j}{w_i}|\nabla_{\boldsymbol{p}_j}C(\boldsymbol{p})|^2}\nabla_{\boldsymbol{p}_i}C(\boldsymbol{p})\end{aligned} \tag{3.34}$$

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 momentums 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 momentums of individual particles are not necessarily preserved.

3.4.4 XPBD Constraint Projection

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

$$\mathbf{M}(\boldsymbol{q}_{n+1} - \boldsymbol{q}_n - h\boldsymbol{v}_n) = h^2(\boldsymbol{f}_{\text{ext}} - \sum_j \nabla\psi_j(\boldsymbol{q}_{n+1})). \tag{3.35}$$

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 3.35 yields

$$\mathbf{M}\,(\boldsymbol{q}_{n+1} - \tilde{\boldsymbol{q}}) = -h^2 \sum_j \nabla\psi_j(\boldsymbol{q}_{n+1}), \tag{3.36}$$

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 3.3.4. In the compliant constraint framework, each ψ_j can be written in terms of some positional constraint function C_j

$$\psi_j(\boldsymbol{q}) = \frac{1}{2}\alpha_j^{-1}C_j(\boldsymbol{q})^2, \tag{3.37}$$

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

$$\psi(\boldsymbol{q}) :- \sum_j \psi_j(\boldsymbol{q}) = \frac{1}{2}\boldsymbol{C}(\boldsymbol{q})^T \alpha^{-1} \boldsymbol{C}(\boldsymbol{q}). \tag{3.38}$$

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

$$\boldsymbol{f}_{\text{int}} = -\nabla\psi(\boldsymbol{q}) = -\nabla\boldsymbol{C}(\boldsymbol{q})^T \alpha^{-1} \boldsymbol{C}(\boldsymbol{q}). \tag{3.39}$$

Plugging the internal force $\boldsymbol{f}_{\text{int}}$ into Equation 3.36 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

$$\boldsymbol{\lambda} = -\tilde{\alpha}^{-1} \boldsymbol{C}(\boldsymbol{q}). \tag{3.40}$$

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

$$\mathbf{M}(\boldsymbol{q}_{n+1} - \tilde{\boldsymbol{q}}) - \nabla(\boldsymbol{q}_{n+1})^T \boldsymbol{\lambda}_{n+1} = \mathbf{0} \tag{3.41}$$

$$\boldsymbol{C}(\boldsymbol{q}_{n+1}) + \tilde{\alpha}\boldsymbol{\lambda}_{n+1} = \mathbf{0}. \tag{3.42}$$

The left-hand side of Equation 3.41 and Equation 3.42 are referred to as $\boldsymbol{g}$ and $\boldsymbol{h}$, respectively. The non-linear 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 non-linear 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 3.41 and Equation 3.42:

$$\begin{pmatrix} \mathbf{K} & -\nabla\boldsymbol{C}^T(\boldsymbol{q}_i) \\ \nabla\boldsymbol{C}(\boldsymbol{q}_i) & \tilde{\alpha} \end{pmatrix} \begin{pmatrix} \Delta\boldsymbol{q} \\ \Delta\boldsymbol{\lambda} \end{pmatrix} = - \begin{pmatrix} \boldsymbol{g}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) \\ \boldsymbol{h}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) \end{pmatrix}, \tag{3.43}$$

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

$$\mathbf{K} = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{q}}(\boldsymbol{q}_i) = \mathbf{M} - \frac{\partial \nabla \boldsymbol{C}(\boldsymbol{q}_i)^T \boldsymbol{\lambda}_i}{\partial \boldsymbol{q}}. \tag{3.44}$$

Note how the second term corresponds to the geometric stiffness (see [TNGF15], reference appropriate section). We refer to the system matrix of Equation 3.43 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. In order to avoid the challenges of computing second derivatives and to re-establish a connection to PBD (Section 3.4.2), Macklin et al. [MMC16] drop the geometric stiffness by approximating

$$\mathbf{K} \approx \mathbf{M}. \tag{3.45}$$

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 non-linear systems of equations.

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

$$\boldsymbol{g}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) = \mathbf{0}. \tag{3.46}$$

If initial guesses $\boldsymbol{q}_0 = \tilde{\boldsymbol{q}}$ and $\boldsymbol{\lambda}_0 = \mathbf{0}$ are used, plugging into Equation 3.41 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 3.43 with both assumptions in place:

$$\begin{pmatrix} \mathbf{M} & -\nabla \boldsymbol{C}^T(\boldsymbol{q}_i) \\ \nabla \boldsymbol{C}(\boldsymbol{q}_i) & \tilde{\alpha} \end{pmatrix} \begin{pmatrix} \Delta \boldsymbol{p} \\ \Delta \boldsymbol{\lambda} \end{pmatrix} = - \begin{pmatrix} \mathbf{0} \\ \boldsymbol{h}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) \end{pmatrix}. \tag{3.47}$$

We refer to the system matrix as $\mathbf{H}_{\text{simp}}$. After the first iteration, the upper row of Equation 3.47 is satisfied. Thus, after the first iteration with $\boldsymbol{q}_0 = \tilde{\boldsymbol{q}}$ and $\boldsymbol{\lambda}_0 = \mathbf{0}$, it is

$$\begin{aligned} \mathbf{0} = \mathbf{M}\Delta \boldsymbol{q} - \nabla \boldsymbol{C}(\boldsymbol{q}_0)^T \Delta \boldsymbol{\lambda} &= \mathbf{M}(\boldsymbol{q}_1 - \boldsymbol{q}_0) - \nabla \boldsymbol{C}(\boldsymbol{q}_0)^T (\boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_0) \\ &= \mathbf{M}(\boldsymbol{q}_1 - \tilde{\boldsymbol{q}}) - \nabla \boldsymbol{C}(\boldsymbol{q}_0)^T \boldsymbol{\lambda}_1. \end{aligned} \tag{3.48}$$

Using Equation 3.48, $\boldsymbol{g}(\boldsymbol{q}_1, \boldsymbol{\lambda}_1)$ can be rewritten as

$$\begin{aligned}
&\boldsymbol{g}(\boldsymbol{q}_1, \boldsymbol{\lambda}_1) = \boldsymbol{g}(\boldsymbol{q}_1, \boldsymbol{\lambda}_1) - \mathbf{0} \\
&= \mathbf{M}(\boldsymbol{q}_1 - \tilde{\boldsymbol{q}}) - \nabla \boldsymbol{C}(\boldsymbol{q}_1)^T \boldsymbol{\lambda}_1 - \mathbf{0} \\
&= \mathbf{M}(\boldsymbol{q}_1 - \tilde{\boldsymbol{q}}) - \nabla \boldsymbol{C}(\boldsymbol{q}_1)^T \boldsymbol{\lambda}_1 - \mathbf{M}(\boldsymbol{q}_1 - \tilde{\boldsymbol{q}}) - \nabla \boldsymbol{C}(\boldsymbol{q}_0)^T \boldsymbol{\lambda}_1 \\
&= \nabla \boldsymbol{C}(\boldsymbol{q}_0)^T \boldsymbol{\lambda}_1 - \nabla \boldsymbol{C}(\boldsymbol{q}_1)^T \boldsymbol{\lambda}_1 \\
&= (\nabla \boldsymbol{C}(\boldsymbol{q}_0)^T - \nabla \boldsymbol{C}(\boldsymbol{q}_1)^T) \boldsymbol{\lambda}_1
\end{aligned} \tag{3.49}$$

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 $\Delta\boldsymbol{\lambda}$:

$$(\nabla \boldsymbol{C}(\boldsymbol{q}_i) \mathbf{M}^{-1} \nabla \boldsymbol{C}(\boldsymbol{q}_i)^T + \tilde{\alpha}) \Delta \boldsymbol{\lambda} = -\boldsymbol{C}(\boldsymbol{q}_i) - \tilde{\alpha} \boldsymbol{\lambda}_i \tag{3.50}$$

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

$$\Delta \boldsymbol{q} = \mathbf{M}^{-1} \nabla \boldsymbol{C}(\boldsymbol{q}_i)^T \Delta \boldsymbol{\lambda}. \tag{3.51}$$

Up until here, all constraints were handled together during each iteration. In order to make a connection to PBD and to return to the framework of a non-linear Gauss-Seidel solver Section 3.4.1, it is necessary to specify how to solve a single constraint. To that end we rewrite Equation 3.50 for a single constraint C_j and get the update for its scalar Lagrange multiplier λ_j by computing

$$\Delta \lambda_j = \frac{-C_j(\boldsymbol{q}_i) - \tilde{\alpha}_j \lambda_{ji}}{\nabla C_j(\boldsymbol{q}_i) \mathbf{M}^{-1} \nabla C_j(\boldsymbol{q}_i)^T + \tilde{\alpha}_j}. \tag{3.52}$$

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

$$\Delta \boldsymbol{q}_l = w_l \nabla_{\boldsymbol{q}_l} C_j(\boldsymbol{q}_i)^T \Delta \lambda_j. \tag{3.53}$$

Note that $\Delta \lambda_j$ is scalar. Thus, the position update is a multiple of the mass-weighted gradient, just like in PBD (Eq. (3.30)).

In summary, we simply compute $\Delta \lambda_j$ via Equation 3.52 and use it to update $\lambda_{j+1} = \lambda_j + \Delta\lambda$ and to determine $\Delta \boldsymbol{q}$ via Equation 3.51 while solving the j-th constraint. This leads to a natural extension of the PBD algorithm, where the general structure in Algorithm 7 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 9. Note that Algorithm 9 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 3.51 and Equation 3.52 in order to maintain notational consistency with the PBD solver in Algorithm 8.

Algorithm 9 XPBD Constraint Solver

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

3.4.5 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.

The derivation of the XPBD constraint projection builds on the concept of compliant constraints developed by Servin et al. [SLM06]. As discussed in Section 3.3.4, the compliant constraint formulation often allows handling infinite stiffness by setting $\alpha = \mathbf{0}$. This requires removing all occurrences of stiffnesses in favor of compliances. In XPBD, this is achieved by pulling large stiffness values out of the constraints in Equation 3.37 and introducing artificial Lagrange multipliers (Eq. (3.40)) that hide large stiffness values and treating them as unknowns

in Equation 3.41 and Equation 3.42 . Indeed, a closer look at the PBD update formula Equation 3.30 and the XPBD update formulas in Equation 3.52, Equation 3.53 reveals that XPBD and PBD are equivalent if the compliance term $\tilde{\alpha}_j$ 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 (Section 3.4.3). If $\tilde{\alpha}_j \neq 0$, the compliance terms in Equation 3.52 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 (Section 3.4.3). Since the time step is also baked into $\tilde{\alpha}_j$, coupling between time step size and stiffness is also reduced.

The discussion above highlights the importance of pulling large stiffness values out of the constraints in Equation 3.38. However, in many cases it can be challenging to pull stiffness factors out of the constraints. In such cases, it is interesting to ask whether XPBD – despite the iterative nature of the Gauss-Seidel solver and the assumptions made during its derivation (Eq. (3.45), Eq. (3.46)) – behaves the same for both formulations if only large, but finite, stiffnesses are used. To this end, the constraint potentials in Equation 3.38 can be rewritten in terms of the alternative constraint function $\boldsymbol{C}'$ given by

$$\boldsymbol{C}'(\boldsymbol{q}) = \alpha^{-1/2}\boldsymbol{C}(\boldsymbol{q}). \tag{3.54}$$

leading to the equivalent constraint potentials

$$\psi(\boldsymbol{q}) = \frac{1}{2}\boldsymbol{C}'(\boldsymbol{q})^T\mathbf{I}\boldsymbol{C}'(\boldsymbol{q}) = \frac{1}{2}\boldsymbol{C}'(\boldsymbol{q})^T\alpha'^{-1}\boldsymbol{C}'(\boldsymbol{q}). \tag{3.55}$$

Here, $\alpha'^{-1} = \mathbf{I}$ without relation to the constraint stiffnesses k_j. Thus, using zero compliance in order to model infinite stiffness becomes impossible. If $\boldsymbol{C}'$ and α' are used, the update formulas for individual Lagrange multipliers (Eq. (3.52)) and individual positions (Eq. (3.53)) become

$$\begin{aligned}
\Delta\lambda_j &= \frac{-C'_j(\boldsymbol{q}_i) - \frac{1}{h^2}\lambda_{ij}}{\nabla C'_j(\boldsymbol{q}_i)\mathbf{M}^{-1}\nabla C'_j(\boldsymbol{q}_i)^T + \frac{1}{h^2}} \\
&= \frac{-\sqrt{k_j}C_j(\boldsymbol{q}_i) - k_j\tilde{\alpha}_j\lambda_{ij}}{k_j\nabla C_j(\boldsymbol{q}_i)\mathbf{M}^{-1}\nabla C_j(\boldsymbol{q}_i)^T + k_j\tilde{\alpha}_j} \\
&= \frac{-\frac{1}{\sqrt{k_j}}C_j(\boldsymbol{q}_i) - \tilde{\alpha}_j\lambda_{ij}}{\nabla C_j(\boldsymbol{q}_i)\mathbf{M}^{-1}\nabla C_j(\boldsymbol{q}_i)^T + \tilde{\alpha}_j}.
\end{aligned} \tag{3.56}$$

and

$$\begin{aligned}\Delta \boldsymbol{q}_l &= \frac{-\frac{1}{\sqrt{k_j}} C_j(\boldsymbol{q}_i) - \tilde{\alpha}_j \lambda_{ij}}{\nabla C_j(\boldsymbol{q}_i) \mathbf{M}^{-1} \nabla C_j(\boldsymbol{q}_i)^T + \tilde{\alpha}_j} w_l \nabla C_j'(\boldsymbol{q}_i)^T \\ &= \frac{-\frac{1}{\sqrt{k_j}} C_j(\boldsymbol{q}_i) - \tilde{\alpha}_j \lambda_{ij}}{\nabla C_j(\boldsymbol{q}_i) \mathbf{M}^{-1} \nabla C_j(\boldsymbol{q}_i)^T + \tilde{\alpha}_j} w_l \sqrt{k_j} \nabla C_j(\boldsymbol{q}_i)^T.\end{aligned} \tag{3.57}$$

By looking at the last lines of Equation 3.56 and Equation 3.57, it is easy to see that using C_j' and α_j' with $\boldsymbol{\lambda}_0 = 0$ yields the same positions, but decreases the Lagrange multipliers by a factor of $1/\sqrt{k_j}$. Note that numerical differences between both formulations might still arise (No idea how to reason about this. Looks this up or ask Fabian).

As mentioned in Section 3.4.4, assumption 1 (Eq. (3.45)) 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 $\boldsymbol{\lambda}_0 = \vec{0}$, this assumption is trivially satisfied during the first iteration. According to Equation 3.40, 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 3.41 and Equation 3.42, 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 3.43 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 3.48 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 3.48 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 3.43 and Equation 3.47. 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].

When investigating the properties of PBD (Section 3.4.3), 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 3.34 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

$$\Delta \boldsymbol{q}_i = -\frac{C(\boldsymbol{q})}{\sum_{j \in \{i_1, \ldots, i_{n_c}\}} \frac{w_j}{w_i} |\nabla_{\boldsymbol{q}_j} C(\boldsymbol{q})|^2 + \tilde{\alpha} a m_i} \nabla_{\boldsymbol{q}_i} C(\boldsymbol{q}). \tag{3.58}$$

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 masses from Equation 3.47.

Both assumptions 1 and 2 benefit from 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 3.40. 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.

While motion due to external forces is handled by the symplectic Euler integration in lines 3-4 of Algorithm 7, the way elasticity is handled is derived from the implicit equations of motions (Eq. (3.35)). As a result elastic forces are subjected to numerical damping that gets more severe with growing time step sizes in XPBD (Section 3.1.2). 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.

Finally, it is worth pointing out that the implications of using a Gauss-Seidel type solver discussed in the context of PBD (Section 3.4.3) apply to XPBD as well.

3.5 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 efficienty.

Glue text where the following subsections are listed!

3.5.1 Overview Over Projective Dynamics

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

$$\min_{\mathbf{q}_{n+1}} \frac{1}{2h^2} \left\| \mathbf{M}^{\frac{1}{2}} (\mathbf{q}_{n+1} - \tilde{\mathbf{q}}) \right\|_F^2 + \sum_j \psi_j(\mathbf{q}_{n+1}). \tag{3.59}$$

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 (Section 3.1.2) and can be determined using one of the line search algorithms introduced in Section 3.2.1. 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 momentum potential (always quadratic) and the elastic potential (quadratic by assumption) the entire objective function of Equation 3.59 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 in order 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 Section 3.5.2). 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

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

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

$$W_j(\mathbf{q}_j, \mathbf{p}_j) = d_j(\mathbf{q}_j, \mathbf{p}_j) + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j). \tag{3.60}$$

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

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

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 Section 3.5.2). 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 q, p and q', p' it is

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

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

$$\psi(\mathbf{q}) = \min_{\mathbf{p}_j} W(\mathbf{S}_j\mathbf{q}, \mathbf{p}_j) = \min_{\mathbf{p}_j} d_j(\mathbf{q}_j, \mathbf{p}_j) + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j), \tag{3.61}$$

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 (Eq. (3.59)) yields

$$\min_{\mathbf{q},\mathbf{p}_j} \frac{1}{2h^2} \left\| \mathbf{M}^{\frac{1}{2}}(\mathbf{q} - \tilde{\mathbf{q}}) \right\|_F^2 + \sum_j d_j(\mathbf{S}_j\mathbf{q}, \mathbf{p}_j) + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j). \tag{3.62}$$

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

$$\min_{\mathbf{q}} \frac{1}{2h^2} \left\| \mathbf{M}^{\frac{1}{2}}(\mathbf{q} - \tilde{\mathbf{q}}) \right\|_F^2 + \sum_j d_j(\mathbf{S}_j\mathbf{q}, \mathbf{p}_{\mathbf{q}_j}). \tag{3.63}$$

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 3.62 over the projection variables $\mathbf{p}_j$ while keeping the positions $\mathbf{q}$ fixed, yielding the following optimization problem

$$\min_{\mathbf{p}_j} \sum_j d_j(\mathbf{S}_j\mathbf{q}, \mathbf{p}_j) + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j). \tag{3.64}$$

Since each constraint has its own auxiliary projection variables, this minimization can be carried out independently for each constraint. In the global step, Equation 3.63 is minimized, which is equivalent to minimizing Equation 3.62 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.

3.5.2 PD Energy Potentials

In order to arrive at a working implementation of the PD solver outlined in Section 3.5.1, 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

$$d_j(\mathbf{q}_j, \mathbf{p}_j) = \frac{w_j}{2}\left\|\mathbf{A}_j\mathbf{q}_j - \mathbf{B}_j\mathbf{p}_j\right\|_F^2, \tag{3.65}$$

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. 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.

$$d_j(a\mathbf{q}_j, a\mathbf{p}_j) = a^2 d_j(\mathbf{q}_j, \mathbf{p}_j) \neq a d_j(\mathbf{q}_j, \mathbf{p}_j).$$

Plugging the definition of d_j in Equation 3.65 into the PD energy potentials in Equation 3.61 yields

$$\psi_j(\mathbf{q}) = \min_{\mathbf{p}_j} \frac{w_j}{2}\left\|\mathbf{A}_j\mathbf{q}_j - \mathbf{B}_j\mathbf{p}_j\right\|_F^2 + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j). \tag{3.66}$$

Example: Strain Energies As an elucidating example, we briefly recap how to formulate strain energies in terms of PD energy potentials (Equation 3.66) according to Bouaziz et al. [BML+14]. Strain energies measure the change of local variation between the deformed and undeformed state. If the simulated body consists of linear tetrahedral elements, the continuous strain energy can be discretized across the tetrahedra according to (refer to appropriate section). The energy potential for a single tetrahedron with index j can be approximated in terms of PD energy potentials via a constraint acting on the four tetrahedral particles. Thus, it is $\mathbf{q}_j, \mathbf{p}_j \in \mathbb{R}^{4\times 3}$. The constraint manifold $\boldsymbol{C}_j$ contains 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. If $\mathbf{D} \in \mathbb{R}^{3\times 4}$ is 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

$$\boldsymbol{C}_j = \{\mathbf{p} \mid \mathbf{D}\mathbf{p} = \mathbf{F}_{\mathbf{p}}^T \in \mathrm{SO}(3)\}, \tag{3.67}$$

where $\mathrm{SO}(3)$ 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

$$\begin{aligned}\psi_j(\mathbf{q}) = \min_{\mathbf{p}_j} W(\mathbf{S}_j\mathbf{q}, \mathbf{p}_j) &= \min_{\mathbf{p}_j} \frac{k_j V_j}{2} \left\|\mathbf{D}\mathbf{q}_j - \mathbf{D}\mathbf{p}_j\right\|_F^2 + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j) \\ &= \min_{\mathbf{p}_j} \frac{k_j V_j}{2} \left\|\mathbf{F}_{\mathbf{q}_j}^T - \mathbf{F}_{\mathbf{p}_j}^T\right\|_F^2 + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j).\end{aligned} \tag{3.68}$$

Essentially, $\psi_j(\mathbf{q})$ computes the squared distance of $\mathbf{q}_j$ to the closest undeformed tetrahedral configuration, where the distance is defined in terms of the Frobenius norm of the difference of their deformation gradients. It is easy to show that this energy is zero if and only if $\mathbf{F}_{\mathbf{q}_j}$ itself is a rotational matrix and grows as the singular values of $\mathbf{F}_{\mathbf{q}_j}$ move away from 1 [BML+14]. Informally, the energy increases the more the tetrahedron gets stretched or squashed. The projection $\mathbf{p}_j$ can be computed as $\mathbf{p}_j = \mathbf{U}\mathbf{I}\mathbf{V}^T$ where $\mathbf{F}_{\mathbf{q}_j}^T = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$ is the singular value decomposition of the inverse deformation gradient $\mathbf{F}_{\mathbf{q}_j}^T$.

3.5.3 Projective Implicit Euler Solver

The PD solver can be derived by plugging the squared distance functions from Equation 3.65 into the variational form of implicit Euler integration for PD energy potentials (Eq. (3.62)). This yields the optimization problem given by

$$\min_{\mathbf{q},\mathbf{p}_j} \frac{1}{2h^2}\left\|\mathbf{M}^{\frac{1}{2}}(\mathbf{q}-\tilde{\mathbf{q}})\right\|_F^2 + \sum_j \frac{w_j}{2}\left\|\mathbf{A}_j\mathbf{S}_j\mathbf{q}-\mathbf{B}_j\mathbf{p}_j\right\|_F^2 + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j) \tag{3.69}$$

In order 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 $\mathbf{p}_j \in \mathbb{R}^{r\times 3}$. Consider the energy potentials for strain energies introduced in Eq. (3.66). There, $\boldsymbol{C}_j' = \mathrm{SO}(3)$ and $\mathbf{p}_j \in \mathbb{R}^{3\times 3}$, meaning that the projection variables now correspond to the transpose of the deformation gradient instead of the configuration of the projected tetrahedron. If further $\mathbf{A}_j\mathbf{S}_j$ are combined into a single matrix $\mathbf{G}_j$, an equivalent optimization problem given by

$$\min_{\mathbf{q},\mathbf{p}_j} \tilde{g}(\mathbf{q},\mathbf{p}_j) = \min_{\mathbf{q},\mathbf{p}_j} \frac{1}{2h^2}\left\|\mathbf{M}^{1/2}(\mathbf{q}-\tilde{\mathbf{q}})\right\|_F^2 + \sum_j \frac{w_j}{2}\left\|\mathbf{G}_j\mathbf{q}-\mathbf{p}_j\right\|_F^2 + \delta_{\boldsymbol{C}_j'}(\mathbf{p}_j) \tag{3.70}$$

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

As discussed in Section 3.5.1, the local step consists of minimizing the objective function Equation 3.70 over the auxiliary variables $\mathbf{p}_j$ while keeping the positions $\mathbf{q}$ fixed. For each energy potential, we solve the following minimization problem

$$\min_{\mathbf{p}_j} \tilde{g}(\mathbf{q},\mathbf{p}_j) = \min_{\mathbf{p}_j} \frac{w_j}{2}\left\|\mathbf{G}_j\mathbf{q}-\mathbf{p}_j\right\|_F^2 + \delta_{\boldsymbol{C}_j}(\mathbf{p}_j). \tag{3.71}$$

In the global step, the minimization problem Equation 3.70 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

$$\min_{\mathbf{q}} \tilde{g}(\mathbf{q},\mathbf{p}_j) = \min_{\mathbf{q}} \frac{1}{2h^2}\left\|\mathbf{M}^{1/2}(\mathbf{q}-\tilde{\mathbf{q}})\right\|_F^2 + \sum_j \frac{w_j}{2}\left\|\mathbf{G}_j\mathbf{q}-\mathbf{p_j}\right\|_F^2. \tag{3.72}$$

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

$$\nabla_{\mathbf{q}}\tilde{g}(\mathbf{q},\mathbf{p}_j) = \frac{1}{h^2}\mathbf{M}(\mathbf{q}-\tilde{\mathbf{q}}) + \sum_j w_j\mathbf{G}_j^T\mathbf{G}_j\mathbf{q} + \sum_j w_j\mathbf{G}_j^T\mathbf{p}_j. \tag{3.73}$$

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

$$(\frac{1}{h^2}\mathbf{M} + \sum_j w_j \mathbf{G}_j^T \mathbf{G}_j)\mathbf{q} = \frac{1}{h^2}\mathbf{M}\tilde{\boldsymbol{q}} + \sum_j w_j \mathbf{G}_j^T \mathbf{p}_j. \tag{3.74}$$

In the rest of Section 3.5 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 10.

Algorithm 10 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

3.5.4 Properties of Projective Dynamics

- The authors claim that since the constraint manifolds already capture nonlinearities the need for complicated squared distance functions d can be relaxed while still achieving visually plausible simulations [BML+14]. This belongs into the discussion.

- Talk about how handling everything implicitly can also cause dampening of rigid body motion. I believe this is mentioned in [SLM06].

The structure of the PD energy potentials allows for Algorithm 10 to be implemented efficiently. Since the constraint projections in Equation 3.71 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 3.74, the global system can be solved independently and in parallel for each coordinate.

The last property follows from the fact that the squared distance functions d_j in Equation 3.65 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 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 Neo-Hookean and St. Venant-Kirchoff energies do not fit into the PD framework [LBK17]. On the other hand, the authors show that various different types of constraints, including strain constraints, bending constraints, collisions and positional 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.

It is also important to note that it is impossible to implement hard constraints via energy potentials. Increasing the weights of constraints allows approximating hard constraints. However, this comes with adverse effects to the numerical properties of the system matrix $\mathbf{S}$ (Elaborate on this!!).

While a simplified minimization problem is constructed by restricting to PD energy potentials, the solver does converge to a true solution of Equation 3.70. That means that the solution strikes a balance between preserving the momenta of particles while minimizing the energy potentials. 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.

The PD solver (Algorithm 10) performs implicit integration and introduces numerical damping as a result (see Section 3.1.2). According to [BML+14], this is particularly severe when the optimization is terminated early and large meshes are used. One possible explanation is that external forces might not be able to propagate fully through the mesh if the optimization is not run for enough iterations [BML+14] (Investigate this with experiments!!!).

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

lead to unpredictable solver speeds that are infeasible in the context of real-time simulations.

- Not sure whether comparisons to the Newton solver belong here or not. Leave them as bullet points for now.
- In terms of iterations, of course inferior to a Newton solver since the PD solver only exhibits linear convergence. Either way, after a couple of iterations of the PD solver, the results are visually indistinguishable from the true solution computed by Newton's method.
- Simplicity. Much easier than implementing a Newton solver, which requires second derivatives, a line search and possible Hessian modifications if the Hessian is not positive definite.

3.5.5 Projective Dynamics 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 (Eq. (3.70)) is solved by using a specialized local/global alternating minimization technique (Section 3.5.3). If the projection points $\mathbf{p}_\mathbf{q}^j$ with

$$\psi_j(\mathbf{q}) = \min_\mathbf{p} \frac{w_j}{2}\|\mathbf{gq} - \mathbf{p}\|_F^2 + \delta_C(\mathbf{p}) = \frac{w_j}{2}\left\|\mathbf{gq} - \mathbf{p}_\mathbf{q}^i\right\|_F^2$$

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

$$\min_\mathbf{q} g(\mathbf{q}) = \min_\mathbf{q} \frac{1}{2h^2}\left\|\mathbf{M}^{1/2}(\mathbf{q} - \tilde{\mathbf{q}})\right\|_F^2 + \sum_j \frac{w_j}{2}\left\|\mathbf{G}_j\mathbf{q} - \mathbf{p_j}(\mathbf{q})\right\|_F^2 \tag{3.75}$$

can be solved using general purpose algorithms for unconstrained optimization, including Newton's method (Section 3.2.1), the BFGS method (Section 3.2.1) or the L-BFGS method (Section 3.2.1) It is better to use the argmin notation here again, possibly even $W(\boldsymbol{p}, \mathbf{q})$. In fact, it can be shown that the PD solver applied to Equation 3.70 is a special case of a Quasi-Newton method with constant Hessian approximation applied to Equation 3.75 [LBK17].

The gradient ∇g of the objective function g from Equation 3.75 needs to be computed for all the line search methods mentioned above. Liu et al. [LBK17] show that (maybe put the derivation into the appendix) ∇g is given by

$$\nabla g(\mathbf{q}) = \frac{1}{h^2}\mathbf{M}(\mathbf{q} - \tilde{\mathbf{q}}) + \sum_j w_j \mathbf{G}_j^T \mathbf{G}_j \mathbf{q} + \sum_j w_j \mathbf{G}_j^T \mathbf{p}_j(\mathbf{q}). \tag{3.76}$$

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 (Eq. (3.73)). Thus, the gradient is unaffected by whether $\mathbf{p}_j$ is considered constant or a function $\mathbf{p}_j(\mathbf{q})$ of the positions $\mathbf{q}$. In Newton's method, the search direction for the current iteration $\mathbf{r}_k^N$ is given by $-(\nabla^2 g(\mathbf{q}))^{-1}\nabla g(\mathbf{q})$, which is a descent direction if $\nabla^2 g(\mathbf{q})$ is positive definite (Section 3.2.1). In Quasi-Newton methods, the true Hessian $\nabla^2 g(\mathbf{q})$ is approximated by some positive definite matrix $\mathbf{B}_k$ instead (Section 3.2.1). If $\mathbf{B}_k = \mathbf{S}$ – i.e. the system matrix (Eq. (3.74)) from the global optimization in PD – is used with step size $\alpha = 1$ the following update is recovered:

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

Note that

$$\mathbf{q}^* \coloneqq \mathbf{S}^{-1}(\frac{1}{h^2}\mathbf{M}\tilde{\mathbf{s}} + \sum_j w_j \mathbf{G}_j^T \mathbf{p}_j(\mathbf{q}))$$

is exactly the solution of the global linear system from PD in Equation 3.74. Together

$$\mathbf{q}^* = \mathbf{q} - \mathbf{S}^{-1}\nabla g(\mathbf{q})$$

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 3.75 is equivalent to performing a local/global iteration of the PD solver (Algorithm 10). The Quasi-Newton version of PD is summarized in Algorithm 11.

This insight that PD is a special case of Quasi-Newton methods applied to $Equation\ 3.75$ 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 3.6.

3.6 Quasi-Newton methods for Physical Simulations

In Section 3.5.5, we discussed that the PD solver for the minimization problem Equation 3.70 can be interpreted as a special case of a Quasi-Newton method with

Algorithm 11 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

constant Hessian approximation and step size $\alpha = 1$ applied to the corresponding minimization problem Equation 3.75. The minimization problem for the Quasi-Newton method

$$\min_{\mathbf{q}} g(\mathbf{q}) = \min_{\mathbf{q}} \frac{1}{2h^2}\left\|\mathbf{M}^{1/2}(\mathbf{q} - \tilde{\mathbf{q}})\right\|_F^2 + \sum_j \frac{w_j}{2}\left\|\mathbf{G}_j\mathbf{q} - \mathbf{p_j}(\mathbf{q})\right\|_F^2 \tag{3.77}$$

and the global system matrix $\mathbf{S}$ which is used as the constant Hessian approximation

$$\mathbf{S} \coloneqq \frac{1}{h^2}\mathbf{M} + \sum_j w_j \mathbf{G}_j^T \mathbf{G}_j \tag{3.78}$$

are stated here once more for convenience.

As the Hessian matrix $\nabla^2 g(\mathbf{q}_k)$ generally changes between iterations, the Hessian approximations of the BFGS (Section 3.2.1) and L-BFGS method (Section 3.2.1) do so as well. This is in contrast to the constant Hessian approximation $\mathbf{S}$, which does not incorporate local curvature information at the current iterate at all. Combining both approaches in order to accelerate convergence is discussed in Section 3.6.1.

Since Quasi-Newton methods can be applied to the variational form of implicit Euler integration with general conservative energy potentials (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-Kirchoff energies. This extension is covered in Section 3.6.2

3.6.1 Quasi-Newton Methods for PD Energy Potentials

As discussed in Section 3.2.1, the choice of the initial inverse Hessian approximation $\mathbf{H}_0$ has a strong impact on the performance of Quasi-Newton methods such as the BFGS and L-BFGS method. Popular choices like scaled versions of the identitiy matrix generally do not satisfy any formal optimality conditions but are the result of trial and error. This suggests that finding more suitable candidates for $\mathbf{H}_0$ – or alternatively for $\mathbf{B}_0$ – is an avenue towards improving the convergence properties of Quasi-Newton methods. It stands to reason that due to its effectiveness in PD, $\mathbf{S}$ is a viable choice for the initial Hessian approximation $\mathbf{B}_0$ for the minimization of Equation 3.77 [LBK17]. From the lens of the Quasi-Newton version of the PD solver (Algorithm 11), benefits of incorporating local curvature information into the constant Hessian approximations S by applying BFGS or L-BFGS updates are to be expected.

Due to its favorable space complexity over the BFGS method for large-scale problems (see Section 3.2.1), Liu et al. focus on the L-BFGS method with $\mathbf{B}_0 = \mathbf{S}$ [LBK17]. This leads to the algorithm that is laid out in Algorithm 12. The occurrences of f in the two-loop recursion (Algorithm 4) need to be replaced by g in the appropriate places. Again, the details of maintaining the history of $\mathbf{s}, \mathbf{y}, \rho$ are omitted for the sake of clarity.

Algorithm 12 L-BFGS method for PD energies

require $\beta \in (0,1), t \in (0,1)$
procedure SOLVEPDVIALBFGS($\mathbf{q}_n$, $\mathbf{v}_n$, $\mathbf{f}_{\text{ext}}$, m, h)
    $\tilde{\mathbf{q}} = \mathbf{q}_n + h\mathbf{v}_n + h^2\mathbf{M}^{-1}\mathbf{f}_{\text{ext}}$
    $\mathbf{q}_k = m\tilde{a}tmq$
    for all iterations **do**
        $\mathbf{s_k} = \mathbf{q}_k - \mathbf{q}_{k-1}, \mathbf{y}_k = \nabla g(\mathbf{q}_k) - \nabla g(\mathbf{q}_{k-1}), \rho_k = 1/(\mathbf{s}_k^T\mathbf{y}_k)$
        $\mathbf{p}_k =$ TWOLOOPRECURSION$(\mathbf{S}, \mathbf{q}_k, \mathbf{s}, \mathbf{y}, \rho, m, k)$ (Algorithm 4)
        $\alpha_k = 1$
        if α_k does not satisfy the strong Wolfe conditions **then**
            compute α_k that satisfies the strong Wolfe conditions
            or $\alpha_k =$ BACKTRACK$(\mathbf{q}_k, \mathbf{p}_k, 1, \beta, t)$ (Algorithm 6)
        end if
        $\mathbf{q}_k = \mathbf{q}_k + \alpha_k\mathbf{p}_k$
    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

Note that step-size $\alpha_k = 1$ is used during each iteration of the Quasi-Newton version of the PD solver (Algorithm 11). While this works well for the constant

Hessian approximation $\mathbf{S}$, it is not viable in L-BFGS. The step-size $\alpha_k = 1$ is not guaranteed to satisfy the strong Wolfe conditions (Eq. (3.10), Eq. (3.11)) if the L-BFGS update is used. However, the strong Wolfe conditions are required to ensure that the curvature condition (Eq. (3.17)) is satisfied and $\mathbf{H}_k$ stays symmetric positive definite.

While Liu et al. [LBK17] show that always picking $\alpha_k = 1$ can lead to unstable simulations, they report that using backtracking (Algorithm 6) yields satisfactory results, even though the resulting step-size is not guaranteed to satisfy the second Wolfe condition. This is in contrast to Nocedal's and Wright's recommendation [NW06] to avoid backtracking in the context of Quasi-Newton methods. One possible explanation why backtracking seems to suffice could be that the inverse Hessian approximations $\mathbf{H}_k$ that arise from the L-BFGS method with initial matrix $\mathbf{B}_0 = \mathbf{S}$ are better than Hessian approximations that arise when traditional initial matrices are used and make up for the inaccuracy in the step length algorithm.

3.6.2 Quasi-Newton Methods for Valanis-Landel Energies

In order to achieve satisfactory performance when applying the L-BFGS method to the general form of the implicit Euler integration (Eq. (3.7)) without the need to restrict energy potentials to the PD framework (Eq. (3.66)), 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 12 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 3-manifold simplicial complexes where energy potentials are defined for each tetrahedron and the energy of the entire body is the sum of the individual tetrahedral energies (refer to the appropriate section). As shown in Section 3.5.2, the strain energy of a single tetrahedron can be emulated with special PD strain potentials (Eq. (3.68)). 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

$$\mathbf{B}_0 = \mathbf{S} = \frac{1}{h^2}\mathbf{M} + \sum_j w_j \mathbf{M}_j^T \mathbf{A}_j. \tag{3.79}$$

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 the subclass of these material models that can be written in Valanis-Landel form

$$\Psi(\sigma_1, \sigma_2, \sigma_3) = a(\sigma_1) + a(\sigma_2) + a(\sigma_3) + b(\sigma_1\sigma_2) + b(\sigma_2\sigma_3) + b(\sigma_1\sigma_3) \quad (3.80)$$

with $a, b, c : \mathbb{R} \to \mathbb{R}$. Many popular material models including St. Venant-Kirchhoff and Neo-Hookean can be expressed in this form (How? I can't seem to find how to do this in detail. There is a paper that might be useful called 'The Valanis–Landel Strain-Energy Function', but I am not sure it is useful . . .).

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

$$\left.\frac{d\psi}{d\sigma_i}\right|_{\sigma_j,\sigma_k=1} = a'(\sigma_i) + 2b'(\sigma_i) + c'(\sigma_i) \quad (3.81)$$

$$\left.\frac{d^2\psi}{d\sigma_i}\right|_{\sigma_j,\sigma_k=1} = a''(\sigma_i) + 2b''(\sigma_i) + c''(\sigma_i), \quad (3.82)$$

where $i, j, k \in [1, 3], i \neq j \neq k$. Thus, first and second partial derivatives at the rest configuration are the same for each of the principal stretches, allowing for convenient representation of 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 3.82 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 simulation.

To alleviate this, Liu et al. [LBK17] propose approximating the rate of change of the first derivative Equation 3.81 by computing the slope of its best linear approximation over the interval $[\sigma_{\text{min}}, \sigma_{\text{max}}]$. Formally, k is defined by

$$k := \underset{k}{\text{argmin}} \int_{\sigma_{\text{min}}}^{\sigma_{\text{max}}} (k(\sigma - 1) - (a'(\sigma) + 2b'(\sigma) + c'(\sigma))^2 d\sigma \quad (3.83)$$

Using these stiffness values in Equation 3.79 yields a suitable initial Hessian approximation $_0 = S$ for the minimization of the variational form of the implicit Euler integration with Valanis-Landel materials via Algorithm 12. According to Liu et al. Algorithm 12 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 12.

3.6.3 Properties of Quasi-Newton Methods for Physical Simulations

- Stability, some nice properties of PD are lost due to the L-BFGS update and step length algorithm / backtracking
- Efficient structure ($\mathbb{R}^{m\times m}$ vs $\mathbb{R}^{3m\times 3m}$) that allows solving in parallel.
- Gradients and function evaluations can be computed in parallel
- Generally very few line searches required, but Armijo safeguard is essential
- Considerations about window size
- Convergence properties compared to regular PD, Newton's method, L-BFGS with classical initializations
- Complexity compared to Newton's method (no second derivatives required)
- Backtracking vs proper step length algorithm
- Comparison to QN methods with iterative solvers
- Collisions
- Numerical Damping
- Issue: In Hooke's law, the derivatives are in terms of positions, but in our derivations the derivatives are in terms of the singular values!
- How to write Neo-Hookean in Valanis-Landel form?

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 pages 25–27).

[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 10, 24–27).

[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, 34, 35, 44–51).

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

[GSS+15] Theodore F. Gast, Craig Schroeder, Alexey Stomakhin, Chenfanfu Jiang, and Joseph M. Teran. “Optimization Integrator for Large Time Steps”. In: *IEEE Transactions on Visualization and Computer Graphics* 21.10 (2015), pp. 1103–1115 (cited on page 5).

[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 21, 50, 52, 54–57).

[LLF+23] Andreas Longva, Fabian Löschner, José Antonio Fernández-Fernández, Egor Larionov, Uri M. Ascher, and Jan Bender. *Pitfalls of Projection: A study of Newton-type solvers for incremental potentials*. 2023 (cited on page 14).

[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, 30–32, 36, 38–40, 42).

[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 30–32, 34, 35).

[MMC+20] Matthias Müller, Miles Macklin, Nuttapong Chentanez, Stefan Jeschke, and Tae-Yong Kim. "Detailed Rigid Body Simulation with Extended Position Based Dynamics". In: *Computer Graphics Forum* 39.8 (2020), pp. 101–112 (cited on page 27).

[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 25).

[NW06] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. 2e. New York, NY, USA: Springer, 2006 (cited on pages 10, 11, 42, 45, 49, 55).

[OBLN17] Matthew Overby, George E. Brown, Jie Li, and Rahul Narain. "ADMM $\supseteq$ 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 page 50).

[RU57] Hanan Rubin and Peter Ungar. "Motion under a strong constraining force". In: *Communications on Pure and Applied Mathematics* 10.1 (1957), pp. 65–87 (cited on page 25).

[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, 22, 24, 25, 27, 40, 50).

[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 page 56).

[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 7, 8).

[TPBF87] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. “Elastically Deformable Models”. In: *SIGGRAPH Comput. Graph.* 21.4 (Aug. 1987), pp. 205–214 (cited on pages 24, 25).

[TNGF15] Maxime Tournier, Matthieu Nesme, Benjamin Gilles, and François Faure. “Stable Constrained Dynamics”. In: *ACM Trans. Graph.* 34.4 (July 2015) (cited on pages 10, 22, 23, 26–29, 38, 42).

[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 pages 25, 26).

[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 45).

Index