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, February 29, 2024

Dennis Ledwon

Contents

1 Introduction **1**

2 Related Work **3**

3 Method **5**

3.1 Time-Integration of Physical Systems 5
3.1.1 Newton's Ordinary Differential Equation 5
3.1.2 Numerical Integration of Newton's Ordinary Differential Equation 6
3.2 Unconstrained Optimization 8
3.2.1 Line Search Methods 9
3.3 Dynamic Simulation 18
3.3.1 Stiff Springs 19
3.3.2 Penalty Forces 20
3.3.3 Hard Constraints 21
3.3.4 Compliant Constraints 23
3.4 Position Based Dynamics 25
3.4.1 Overview Over the PBD Framework 26
3.4.2 PBD Constraint Projection 28
3.4.3 Properties of PBD 29
3.4.4 xPBD Constraint Projection 32
3.4.5 Properties of xPBD 35
3.5 Projective Dynamics 37
3.5.1 Energy Potentials 38
3.5.2 Projective Implicit Euler Solver 40
3.5.3 Properties of Projective Dynamics 41
3.5.4 Projective Dynamics as a Special Case of Quasi-Newton Methods 43
3.6 Quasi-Newton methods for Physical Simulations 45
3.6.1 Quasi-Newton Methods for PD Energy Potentials 45
3.6.2 Quasi-Newton Methods for Valanis-Landel Energies . . . 47

3.6.3 Properties of Quasi-Newton Methods for Physical Simulations 49

Bibliography **51**

Index **55**

Chapter 1

Introduction

Chapter 2

Related Work

Chapter 3

Method

3.1 Time-Integration of Physical Systems

In most approaches for the simulation of physical systems, the motion of the system is assumed to be in accordance with Newton's laws of motion. Due to Newton's second law, it is possible to derive accelerations from forces acting on the system. The motion of the system can then be described in terms of an ordinary differential equation (ODE) which is integrated over time in order to arrive at the configuration of the system at the next time step. Usually, this is achieved via numerical integration schemes. In particular, both xPBD (Section 3.4.4) and PD (Section 3.5.2) are based on a numerical integration technique called implicit Euler integration. The ODE is introduced in Section 3.1.1. Common approaches for numerical integration are briefly reviewed in Section 3.1.2.

3.1.1 Newton's Ordinary Differential Equation

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 following ODE called Newton's ODE

$$\begin{aligned} \boldsymbol{q}(t)\prime &= \boldsymbol{v}(t) \\ \boldsymbol{v}(t)\prime &= \boldsymbol{M}^{-1}\boldsymbol{f}(\boldsymbol{q}(t), \boldsymbol{v}(t)), \end{aligned} \tag{3.1}$$

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 $\boldsymbol{M}$ is a diagonal matrix with the particle masses as diagonal entries. Depending on the context, either $\boldsymbol{q}(t), \boldsymbol{v}(t), \boldsymbol{f}(\boldsymbol{q}(t), \boldsymbol{v}(t)) \in \mathbb{R}^{m\times 3}$ and $\boldsymbol{M} \in \mathbb{R}^{m\times m}$ or $\boldsymbol{q}(t), \boldsymbol{v}(t), \boldsymbol{f}(\boldsymbol{q}(t), \boldsymbol{v}(t)) \in \mathbb{R}^{3m}$ and $\boldsymbol{M} \in \mathbb{R}^{3m\times 3m}$. $\boldsymbol{q}(t)\prime$ and $\boldsymbol{v}(t)\prime$ 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 $\boldsymbol{q}(t)\prime$ for time-dependent quantities for the sake of brevity.

The positions $\boldsymbol{q}$ and velocities $\boldsymbol{v}$ of the system at time t can be determind by solving this ODE. For general nonlinear forces, analytical solutions to Newton's ODE are usually not available. Thus the ODE needs to be solved numerically.

3.1.2 Numerical Integration of Newton's Ordinary Differential Equation

The simplest approach to numerical integration is the explict Euler integration [CC05]. When applied to Newton's ODE, the positions and velocities are computed at discrete timesteps via the following update formulas:

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

Here, h is the time step. The idea is to simplify the integration of the functions $\dot{\boldsymbol{q}}, \dot{\boldsymbol{v}}$ over the timestep by using constant approximations. Then, time integration is as simple as multiplying this constant function value with the timestep. In the explicit Euler method, we approximate $\dot{\boldsymbol{q}}, \dot{\boldsymbol{v}}$ by their function values $\dot{\boldsymbol{q}}(t_n), \dot{\boldsymbol{v}}(t_n)$ at the beginning of the timestep. While simple, the explicit Euler method is not stable for stiff systems, i.e. systems with accelerations of large magnitude [CC05]. It can be shown that the explicit Euler amplifies the energy of the simulated system [SD06]. For example, the amplititude of a swinging pendulum increases with time if integrated via the explicit Euler method, even if small time steps are used. Using the explicit Euler method with larger time steps often manifests itself in exploding simulations.

A variation of the explicit Euler method applied to Newton's ODE, 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 following update formula:

$$\begin{aligned}\boldsymbol{q}_{n+1} &= \boldsymbol{q}_n + h\boldsymbol{v}_{n+1} \\ \boldsymbol{v}_{n+1} &= \boldsymbol{v}_n + h\boldsymbol{M}^{-1}\boldsymbol{f}(\boldsymbol{q}_n, \boldsymbol{p}_n)\end{aligned} \tag{3.2}$$

Even though the symplectic Euler method has the same computational cost as the explicit Euler method, 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 Newton's ODE is implicit Euler integration, resulting in the update formula

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

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 set of nonlinear algebraic equations. Despite the added complexity compared to the explicit and symplectic Euler integration schemes implicit Euler integration is popular since it can be shown to be unconditionally stable and first-order accurate [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. However, 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.

By rewriting the first line of Eq. (3.3) as

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

and substituting into the velocity update of Eq. (3.3) the following equation can be derived

$$\boldsymbol{M}(\boldsymbol{q}_{n+1} - \boldsymbol{q}_n - h\boldsymbol{v}_n) = h^2(\boldsymbol{f}(\boldsymbol{q}_{n+1}, \boldsymbol{v}_{n+1})). \tag{3.4}$$

We separate forces $\boldsymbol{f}(\boldsymbol{q}, \boldsymbol{v})$ into internal forces $\boldsymbol{f}_{\text{int}}(\boldsymbol{q}, \boldsymbol{p}) = \sum_{i \in \mathcal{I}_{\text{int}}} \boldsymbol{f}^i_{\text{int}}(\boldsymbol{q}, \boldsymbol{p})$ and external forces $\boldsymbol{f}_{\text{ext}}(\boldsymbol{q}, \boldsymbol{p}) = \sum_{i \in \mathcal{I}_{\text{ext}}} \boldsymbol{f}^i_{\text{ext}}(\boldsymbol{q}, \boldsymbol{p})$ 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 $\boldsymbol{f}(\boldsymbol{q}, \boldsymbol{v}) = \boldsymbol{f}(\boldsymbol{q}) = \boldsymbol{f}_{\text{int}}(\boldsymbol{q}) + \boldsymbol{f}_{\text{ext}} = -\sum_j \nabla\psi_j(\boldsymbol{q}) + \boldsymbol{f}_{\text{ext}}$. Plugging into Eq. (3.4), it is

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

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

$$\min_{\boldsymbol{q}_{n+1}} \frac{1}{2h^2}\|\boldsymbol{q}_{n+1} - \boldsymbol{s}_n\|_F^2 + \sum_j \psi_j(\boldsymbol{q}_{n+1}). \tag{3.6}$$

where $\boldsymbol{s}_n = \boldsymbol{q}_n + h\boldsymbol{v}_n + h^2\boldsymbol{M}^{-1}\boldsymbol{f}_{\text{ext}}$. This minimization problem whose solution corresponds to the next iteration of the state of the implicit Euler integration is called the variational form of implicit Euler integration [BML+14]. The first and second term of the objective function are called the momentum potential and the elastic potential, respectively. 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 $\boldsymbol{M}$, the timestep h and the material stiffness of the elastic potentials ψ_j. According to Noether's theorem, the solution preserves linear and angular momentum as long as the elastic potentials are rigid motion invariant.

Approaching implicit Euler integration in its variational form can often be advantageous. That is because the objective function of Eq. (3.6) 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 [NW06]. As an example, the objective function is used in many step length selection algorithms to ensure that step sizes satisfy the 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].

3.2 Unconstrained Optimization

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 [NW06]. 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 $\boldsymbol{x}^*$ [NW06]. 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 [NW06]: 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 [NW06].

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 [NW06]:

$$f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k) \leq f(\boldsymbol{x}_k) + c_1 \alpha_k \nabla f(\boldsymbol{x}_k)^T \boldsymbol{p}_k \tag{3.7}$$

$$\left| \nabla f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)^T \boldsymbol{p}_k \right| \leq c_2 \left| \nabla f(\boldsymbol{x}_k)^T \boldsymbol{p}_k \right| \tag{3.8}$$

for some constants $c_1 \in (0, 1), c_2 \in (c_1, 1)$. Eq. (3.7) 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. (3.8)), 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 [NW06]. 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 [NW06]. 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 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 [NW06]. 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

$$f(\boldsymbol{x}_k + \boldsymbol{p}) \approx f(\boldsymbol{x}_k) + \boldsymbol{p}^T \nabla f(\boldsymbol{x}_k) + \frac{1}{2}\boldsymbol{p}^T \nabla^2 f(\boldsymbol{x}_k)\boldsymbol{p} =: m_k(\boldsymbol{p}). \tag{3.9}$$

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:

$$\boldsymbol{p}_k^N = -(\nabla^2 f(\boldsymbol{x}_k))^{-1} \nabla f(\boldsymbol{x_k}). \tag{3.10}$$

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 [NW06]. 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 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 [NW06]. 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.

Despite its favorable convergence properties, Newton's method comes with a couple of disadvantages. Firstly, computing the Hessian matrix $\nabla^2 f(\boldsymbol{x}_k)$ is expensive and error prone. Additionally, 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

Algorithm 2 Newton's Method

require $\epsilon > 0$
procedure NEWTONMETHOD($\boldsymbol{x}_0$, 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

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 effective 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 [NW06]. Often, multiple Quasi-Newton iterations fit into the same time budget as a single Newton iteration. As a result, Quasi-Newton methods can converge to a solution in a shorter amount of time than the Newton 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

$$\boldsymbol{p}_k = -\boldsymbol{B}_k^{-1} \nabla f(\boldsymbol{x}_k), \tag{3.11}$$

where $\boldsymbol{B}_k \in \mathbb{R}^{n\times n}$ is positive definite [NW06]. Note that the Newton direction is a special case of Eq. (3.11) with $\boldsymbol{B}_k = \nabla^2 f(\boldsymbol{x}_k)$. Just like for the Newton direction $\boldsymbol{p}_k^N$ in Eq. (3.9), a model function m_k that attains its minimum at $\boldsymbol{p}_k = -\boldsymbol{B}_k^{-1}\nabla f(\boldsymbol{x}_k)$ can be defined via

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

As $\boldsymbol{B}_k \neq \nabla^2 f(\boldsymbol{x}_k)$, m_k does not correspond to a second-order Taylor approximation of f around $\boldsymbol{x}_k$ anymore. Instead, $\boldsymbol{B}_k$ 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 $\nabla m_k(\boldsymbol{0}) = \nabla f(\boldsymbol{x}_k)$, the first condition is true independent of $\boldsymbol{B}_k$. The second condition yields

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

Rearranging gives

$$\boldsymbol{B}_k\boldsymbol{s}_{k-1} = \boldsymbol{y}_{k-1}, \tag{3.13}$$

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

$$\boldsymbol{s}_{k-1}^T\boldsymbol{y}_{k-1} > 0, \tag{3.14}$$

since $\boldsymbol{B}_k$ 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 [NW06]. Thus, a proper line search strategy is vital for the viability of Quasi-Newton methods.

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

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

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

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

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 $\boldsymbol{H}_k$

$$\boldsymbol{H}_k = (I - \rho_{k-1}\boldsymbol{s}_{k-1}\boldsymbol{y}_{k-1}^T)\boldsymbol{H}_{k-1}(I - \rho_{k-1}\boldsymbol{s}_{k-1}\boldsymbol{y}_{k-1}^T) + \rho_{k-1}\boldsymbol{s}_{k-1}\boldsymbol{s}_{k-1}^T, \quad (3.15)$$

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 $\boldsymbol{B}_k$. Generally, using the formulation in terms of the inverse matrices $\boldsymbol{H}_k$ 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 $\boldsymbol{B}_k$ is maintained instead. An overview over the algorithm is given in Algorithm 3.

Algorithm 3 BFGS method

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

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 [NW06] under mild conditions. Thus, just like the Newton 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 $\boldsymbol{H}_0$. One obvious choice is $\boldsymbol{H}_0 = \nabla^2 f(\boldsymbol{x}_0)$. 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 $\boldsymbol{H}_k$ or $\boldsymbol{B}_k$ needs to be stored explicitly, which can be infeasible for large-scale problems. While the BFGS update formula using the inverse matrices $\boldsymbol{H}_k$ replaces the need for a matrix factorization with a simple matrix-vector multiplication, $\boldsymbol{H}_k$ and $\boldsymbol{B}_k$ 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

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 $\boldsymbol{H}_k$. 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 $\boldsymbol{H}_k$. 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 $\boldsymbol{H}_k$ (Eq. (3.15)) [NW06]. 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, $\boldsymbol{H}_k$ can be constructed from the initial matrix $\boldsymbol{H}_0$ 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 $Eq.$ (3.15). 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 $\boldsymbol{H}_0$ at each iteration k.

It is important to note that its not the L-BFGS Hessian approximation $\boldsymbol{H}_k$ itself, but the search direction $\boldsymbol{p}_k = -\boldsymbol{H}_k \nabla f(\boldsymbol{x}_k)$ that is of interest. It turns out that the L-BFGS search direction $\boldsymbol{p}_k$ can be computed without explicitly constructing $\boldsymbol{H}_k$ using an algorithm called the L-BFGS two-loop recursion (Algo-

rithm 4) [NW06]. This algorithm can be specified in terms of the initial Hessian approximation $\boldsymbol{B}_0$ 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.

Algorithm 4 L-BFGS two-loop Recursion

1: **require** $\boldsymbol{H}_0$ or $\boldsymbol{B}_0$ symmetric positive definite
2: **procedure** TWOLOOPRECURSION($\boldsymbol{H}_0$ or $\boldsymbol{B}_0$, $\boldsymbol{x}_k$, $\boldsymbol{s}$, $\boldsymbol{y}$, ρ, m, k)
3: $m^* = \min(m, k), \boldsymbol{t} = -\nabla f(\boldsymbol{x}_k)$
4: **for** $i = k-1, k-2, \ldots, k-m^*$ **do**
5: $\alpha_i = \rho_i \boldsymbol{s}_i^T \boldsymbol{t}$
6: $\boldsymbol{t} = \boldsymbol{t} - \alpha_i \boldsymbol{y}_i$
7: **end for**
8: $\boldsymbol{r} = \boldsymbol{H}_0 \boldsymbol{t}$ or solve $\boldsymbol{B}_0 \boldsymbol{r} = \boldsymbol{t}$
9: **for** $i = k-m^*, k-m^*+1, \ldots, k-1$ **do**
10: $\beta = \rho_i \boldsymbol{y}_i^T \boldsymbol{r}$
11: $\boldsymbol{r} = \boldsymbol{r} + \boldsymbol{s}_i(\alpha_i - \beta)$
12: **end for**
13: **return with result** $-\boldsymbol{H}_k \nabla f(\boldsymbol{x}_k) = \boldsymbol{r}$.
14: **end procedure**

Excluding the matrix-vector multiplication $\boldsymbol{H}_0 \boldsymbol{t}$, the two-loop recursion scheme has time complexity $\mathcal{O}(nm) = \mathcal{O}(n)$ since $m << n$. Thus, if $\boldsymbol{H}_0$ 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 $\boldsymbol{H}_0$ is diagonal. Even if $\boldsymbol{H}_0$ 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 $\boldsymbol{B}_0$ is sparse, but not diagonal. In this case, a factorization of $\boldsymbol{B}_k$ 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 $\boldsymbol{H}_0$, 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 size m is a parameter that needs to be tuned on a per-problem basis [NW06]. 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

Algorithm 5 L-BFGS method

require $\boldsymbol{H}_0$ symmetric positive definite, $\epsilon > 0$
procedure LBFGS($\boldsymbol{x}_0$, $\boldsymbol{H}_0$, m, ϵ)
  $\boldsymbol{x}_k = \boldsymbol{x}_0, \boldsymbol{H}_k = \boldsymbol{H}_0, k = 0$
  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$(\boldsymbol{H}_0, \boldsymbol{x_k}, \boldsymbol{s}, \boldsymbol{y}, \boldsymbol{\rho}, m, k)$ (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

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 [NW06].

Step Length Selection Algorithms

In Section 3.2.1, the need for step lengths α_k that satisfy the strong Wolfe conditions (Eq. (3.7), Eq. (3.8)) 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 [NW06]. 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 [NW06]. As step length algorithms are a common source of bugs, Nocedal and Wright [NW06] recommend using publically available implementations.

To avoid the complexities of correct step length algorithms, the insight that the Armijo condition (Eq. (3.7)) is always satisfied once α is sufficiently close to

zero (Section 3.2.1) can be exploited ([NW06]): 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.

Algorithm 6 Backtracking

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

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 method is used, this tradeoff is often justifiable [NW06]. For less effective search directions, including search directions from most Quasi-Newton methods, backtracking might be less suitable.

3.3 Dynamic Simulation

- Write a paragraph that gives an overview
- 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 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) $\boldsymbol{f} = \boldsymbol{M}\boldsymbol{a}$, where $\boldsymbol{M}$ is a $n_d \times n_d$ diagonal matrix and n_d is the total number of independent degrees of freedom for the particles. Explain Newton's ODE elsewhere and refer to it here.

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 timestep 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):

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

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

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

In these cases, $\boldsymbol{H}$ 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 $\boldsymbol{\phi(x)}$ 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 $\boldsymbol{\phi}_{\text{spring}}(\boldsymbol{x}) = (\|\boldsymbol{x_i} - \boldsymbol{x_j}\|) - l$ with k_{spring} recovers the behavior of a spring with stiffness k_{spring} (Section 3.3.1). Its displacement function $\boldsymbol{\phi}_{\text{spring}}(\boldsymbol{x})$ 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 timestep 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, Newton's ODE (reference somewhere) is 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 $\boldsymbol{\phi(x)} = 0$. Note that the distance constraint $\boldsymbol{\phi}(x)$ is formulated in terms of the particle positions, whereas the ODE works on accelerations or velocities. 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? What are the challenges here? Is this exactly what xPBD is? Is there a way to view the simplifications made in terms of the other frameworks?. Using $\boldsymbol{J} = \frac{\delta \boldsymbol{\phi}}{\delta \boldsymbol{x}}$, where $\boldsymbol{J}$ is a $n_c \times n_d$ matrix and n_c is the number of scalar constraints, this leads to the following constraint formulations:

$$\boldsymbol{Jv} = 0$$
$$\boldsymbol{Ja} = \boldsymbol{c(v)}$$

for some $\boldsymbol{c(v)}$. 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]:

$$\boldsymbol{f_c} = \boldsymbol{J}^T \boldsymbol{\lambda} \tag{3.17}$$

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]:

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

Note that the lower block-row of the system drives towards accelerations that satisfy the constraints imposed by $\boldsymbol{\phi(x)}$ (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:

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

Here, the external forces $\boldsymbol{f}_{\text{ext}}$ and the constraint gradients $\boldsymbol{J}$ are considered constant across the timestep and $\boldsymbol{J}(\boldsymbol{x}_{n+1})$ is not approximated using the Taylor expansion like it is in [BW98]. If internal forces are taken into account, the upper-left matrix $\boldsymbol{M}$ is replaced by the matrix $\boldsymbol{H}$ from Eq. (3.16).

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

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

and perform implicit integration:

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

If we assume that $\boldsymbol{f}_e$ and the constraint gradients $\boldsymbol{J}$ 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]:

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

If the constraints are not redundant, $\boldsymbol{J M^{-1} J^T}$ 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 $\boldsymbol{\phi(x)} = 0$ 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. (3.1)) 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 (Not sure whether I have to repeat again which shape all the quantities have. For now, I don't think it is necessary since I refer to Section 3.3.3 and Section 3.3.2, where the relevant quantities should have been introduced). However, using large, but finite, stiffness values has adverse affects on the numerical properties of the system matrix (Section 3.3.1). Thus, the equations of motion are rewritten in terms of the inverse stiffness. Let $\boldsymbol{C} = [C_i, \dots, C_r]^T$ be the vector function whose entries consist of the individual constraint functions C_j. The potential energy for $\boldsymbol{C}$ is then defined as:

$$\psi(\boldsymbol{q}) = \frac{1}{2} \boldsymbol{C}(\boldsymbol{q})^T \boldsymbol{\alpha}^{-1} \boldsymbol{C}(\boldsymbol{q}) \tag{3.18}$$

where $\boldsymbol{\alpha}$ is a symmetric, positive definite matrix of dimension $n_c \times n_c$ (If I recall correctly, α can only ever be a matrix if we are dealing with constraints that map to vectors. This will never be the case in this thesis, so the formulation should be adapted so that $\boldsymbol{\alpha}$ is simply a scalar). The correspondence to the penalty terms (refer to the appropriate equation) above is the case where $\boldsymbol{\alpha}^{-1}$ is a diagonal matrix with diagonal entries $\frac{1}{k_j}$ for the stiffness k_j of constraint $C_j(\boldsymbol{q})$. The resulting forces are given by

$$\boldsymbol{f_c} = \nabla \psi(\boldsymbol{q}) = -\nabla \boldsymbol{C}(\boldsymbol{q})^T \boldsymbol{\alpha}^{-1} \boldsymbol{C}(\boldsymbol{q}). \tag{3.19}$$

In order to replace the large parameters $\boldsymbol{\alpha}^{-1}$ with the small $\boldsymbol{\alpha}$ in the equations of motion, artificial variables $\boldsymbol{\lambda} = -\boldsymbol{\alpha}^{-1} \boldsymbol{C}$ are introduced, yielding

$$\boldsymbol{f}_c = \nabla \boldsymbol{C}(\boldsymbol{q})^T \boldsymbol{\lambda} \tag{3.20}$$

This leads to the following DAE:

$$\begin{aligned} \dot{\boldsymbol{q}} &= \boldsymbol{v} \\ \boldsymbol{M}\dot{\boldsymbol{v}} &= \boldsymbol{f}_e + \nabla \boldsymbol{C}(\boldsymbol{q})^T \boldsymbol{\lambda} \\ \boldsymbol{\alpha}\boldsymbol{\lambda}(\boldsymbol{q}) &= -\boldsymbol{C}(\boldsymbol{q}) \end{aligned} \tag{3.21}$$

Note, that in the limit of infinite stiffness, 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 the system matrix of the resulting LSE only contains references to the small compliance $\boldsymbol{\alpha}$ instead of the large stiffness $\boldsymbol{\alpha}^{-1}$, improving the condition matrix of the system matrix. This is achieved by treating $\boldsymbol{\lambda} = -\boldsymbol{\alpha}^{-1}\boldsymbol{C}(\boldsymbol{q})$ as an unknown, pulling it out of the system matrix and hiding all occurrences of the large $\boldsymbol{\alpha}^{-1}$ 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 timestep allows pulling $\boldsymbol{\lambda}$ out of the system matrix entirely [TNGF15]. The entries of the resulting system matrix 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

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

leading to the following LSE [TNGF15]

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

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

Regarding the backwards differentiation above, this does not perform the Taylor approximation again. It should be something like:

$$\begin{aligned} C+ \approx C + h\dot{C}+ &= C + hJ_+v_+ \\ &\approx C + h(J + h\dot{J})v_+ \\ &= C + h(J + h\frac{\delta J}{\delta x}\frac{\delta x}{\delta t})v_+ \\ &= C + h(J + h\frac{\delta J}{\delta x}v)v_+ \end{aligned}$$

Now, we need second derivatives of the constraints. This can be seen in [BW98] and is also mentioned in [SLM06].

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. Secondly, in the limit of $\boldsymbol{\alpha} = \mathbf{0}$ the behavior of hard constraints is recovered in an elegant fashion. Lastly, in comparison to penalty forces, entries of large magnitudes in the system matrix due to high stiffness terms are exchanged for small entries in terms of inverse stiffness, which improves the condition number of the matrix.

All these concepts from numerics are a bit unclear to me. I might have to go back to some textbook and do some reading to improve my understanding. I might have to go back to some textbook and do some reading to improve my understanding. Not sure the last part is entirely true.

In [SLM06], a solver based on symplectic Euler which does not require second derivatives is derived. I do not understand some of the estimations made in that derivation. In particular the mean of a function f over and interval (a, b) is defined as $\frac{1}{b-a}\int_a^b f(x)dx$, so what they are saying does not make a lot of sense.

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, typically using one of various numerical integration schemes. 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. Position Based Dynamics (PBD) [MHHR06] addresses these needs by manipulating positions directly on a per-constraint basis without integrating accelerations or velocities. 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 timestep 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 Newton's ODE 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 equiivalent 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 $\nabla_{\boldsymbol{p}_l} C_j(\boldsymbol{q}) = \boldsymbol{0}$.

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.

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** $\boldsymbol{v}_i = \boldsymbol{v}_i + h w_i \boldsymbol{f}_{\text{ext}}(\boldsymbol{x}_i)$
  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

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 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

$$C(\boldsymbol{p} + \Delta\boldsymbol{p}) = 0. \tag{3.22}$$

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

$$\Delta\boldsymbol{p} = \lambda \boldsymbol{W} \nabla C(\boldsymbol{p}) \tag{3.23}$$

for some $\lambda \in \mathbb{R}$ and $\boldsymbol{W} = diag(w_1, w_1, w_1, \ldots, w_m, w_m, w_m)$. Plugging into $Eq.$ (3.22) and approximating by first-order Taylor expansion yields

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

Solving for λ yields

$$\lambda = -\frac{C(\boldsymbol{p})}{\sum_{i \in \{i_1, \ldots, i_{n_c}\}} w_i |\nabla_{\boldsymbol{p}_i} C(\boldsymbol{p})|^2}. \tag{3.24}$$

Plugging λ into Eq. (3.23) results in the final position update

$$\Delta\boldsymbol{p} = -\frac{C(\boldsymbol{p})}{\sum_{i \in \{i_1, \ldots, i_{n_c}\}} w_i |\nabla_{\boldsymbol{p}_i} C(\boldsymbol{p})|^2} \boldsymbol{W} \nabla C(\boldsymbol{p}). \tag{3.25}$$

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

$$\Delta\boldsymbol{p}_i = -\frac{C(\boldsymbol{p})}{\sum_{i \in \{i_1, \ldots, i_{n_c}\}} w_i |\nabla_{\boldsymbol{p}_i} C(\boldsymbol{p})|^2} w_i \nabla_{\boldsymbol{p}_i} C(\boldsymbol{p}) \tag{3.26}$$

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

$$C_{\text{dist}}(\boldsymbol{p}) = |\boldsymbol{p}_{i_1} - \boldsymbol{p}_{i_2}|. \tag{3.27}$$

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 $k\prime = 1-(1-k)^{1/n_s}$. 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

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

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 Eq. (3.25) 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.)

$$x_{i+1} = x_i - \frac{f(x_i)}{f\prime(x_0)}. \tag{3.28}$$

Indeed, applying the Newton-Raphson update to

$$f\colon \mathbb{R} \to \mathbb{R}, \lambda \mapsto C(\boldsymbol{p} + \lambda \boldsymbol{W} \nabla C(\boldsymbol{p})) \tag{3.29}$$

yields

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

and with $\lambda_0 = 0$ we arrive at

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

Note that this is exactly the same as the λ used in the constraint solver given in Eq. (3.24). Thus, a single constraint projection corresponds to the first iteration of the Newton-Raphson method applied to Eq. (3.29) 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 Eq. (3.26) 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 timestep [MHHR06]. Again, we take a look at a distance constraint with rest length 0 (Eq. (3.27)). 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 $k\prime = 1 - (1-k)^{1/n_s}$ 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 intertial 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 Eq. (3.26) it is only the ratio of the particle masses that matters. This can be seen by multiplying all inverse masses w_i in Eq. (3.26) with a constant factor $a \in \mathbb{R}^+$:

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

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 an implicit position-level time discretization of Newton's ODE (Eq. (3.5)), which is restated here again for the sake of convenience

$$\boldsymbol{M}(\boldsymbol{q}_{n+1} - \boldsymbol{q}_n - h\boldsymbol{v}_n) = h^2(\boldsymbol{f}_{\text{ext}} - \sum_j \nabla\psi_j(\boldsymbol{q}_{n+1})). \tag{3.30}$$

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 Eq. (3.30) yields

$$\boldsymbol{M}\,(\boldsymbol{q}_{n+1} - \tilde{\boldsymbol{q}}) = -h^2 \sum_j \nabla\psi_j(\boldsymbol{q}_{n+1}), \tag{3.31}$$

where $\tilde{\boldsymbol{q}} = \boldsymbol{q}_n + h\boldsymbol{v}_n + h^2\boldsymbol{M}^{-1}\boldsymbol{f}_{\text{ext}}$ is the predicted, inertial position. XPBD builds on top of the compliant constraint formulation discussed in Section 3.3.4. The key results relevant to the derivation are restated here again. In the compliant constraint framework each ψ_j can be written in terms of some positional constraint function C_j

$$\psi_j(\boldsymbol{q}) = \frac{1}{2}\alpha_j^{-1}C_j(\boldsymbol{q})^2, \tag{3.32}$$

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 $\boldsymbol{\alpha} = \text{diag}(\alpha_1, \ldots, \alpha_r)$, then

$$\psi(\boldsymbol{q}) :- \sum_j \psi_j(\boldsymbol{q}) = \frac{1}{2}\boldsymbol{C}(\boldsymbol{q})^T\boldsymbol{\alpha}^{-1}\boldsymbol{C}(\boldsymbol{q}). \tag{3.33}$$

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

$$\boldsymbol{f}_{\text{int}} = -\nabla\psi(\boldsymbol{q}) = -\nabla\boldsymbol{C}(\boldsymbol{q})^T\boldsymbol{\alpha}^{-1}\boldsymbol{C}(\boldsymbol{q}). \tag{3.34}$$

Plugging the internal force $\boldsymbol{f}_{\text{int}}$ into Eq. (3.31) and pulling h^2 into the compliance matrix $\boldsymbol{\alpha}$ results in

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

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

$$\boldsymbol{\lambda} = -\tilde{\boldsymbol{\alpha}}\boldsymbol{C}(\boldsymbol{q}) \tag{3.35}$$

according to the compliant constraint formulation by Servin et al. [SLM06]. This leads to the following non-linear system of equations in terms of $\boldsymbol{q}_{n+1}$ and $\boldsymbol{\lambda}_{n+1}$:

$$\boldsymbol{M}(\boldsymbol{q}_{n+1} - \tilde{\boldsymbol{q}}) - \nabla(\boldsymbol{q}_{n+1})^T\boldsymbol{\lambda}_{n+1} = \mathbf{0} \tag{3.36}$$

$$\boldsymbol{C}(\boldsymbol{q}_{n+1}) + \tilde{\boldsymbol{\alpha}}\boldsymbol{\lambda}_{n+1} = \mathbf{0}. \tag{3.37}$$

The left-hand side of Eq. (3.36) and Eq. (3.37) 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 Eq. (3.36) and Eq. (3.37):

$$\begin{pmatrix} \boldsymbol{K} & -\boldsymbol{\nabla C^T}(\boldsymbol{q_i}) \\ \boldsymbol{\nabla C}(\boldsymbol{q_i}) & \tilde{\boldsymbol{\alpha}} \end{pmatrix} \begin{pmatrix} \boldsymbol{\Delta p} \\ \boldsymbol{\Delta\lambda} \end{pmatrix} = - \begin{pmatrix} \boldsymbol{g}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) \\ \boldsymbol{h}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) \end{pmatrix}, \tag{3.38}$$

Here, $\boldsymbol{K} = \frac{\delta \boldsymbol{g}}{\delta \boldsymbol{q}}(\boldsymbol{q}_i)$. At this point, two simplifying assumptions are made.

Assumption 1: Computing K requires evaluating second derivatives of the constraint functions C_j, which is expensive and error-prone. In order to simplify and to re-establish a connection to PBD (Section 3.4.2), Macklin et al. [MMC16] drop geometric stiffness and constraint Hessians by approximating $\boldsymbol{K} \approx \boldsymbol{M}$. 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 $\boldsymbol{g}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) = \mathbf{0}$. If initial guesses $\boldsymbol{q}_0 = \tilde{\boldsymbol{q}}$ and $\boldsymbol{\lambda}_0 = \mathbf{0}$ are used, plugging into Eq. (3.36) 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 Eq. (3.38) with both assumptions in place:

$$\begin{pmatrix} \boldsymbol{M} & -\boldsymbol{\nabla C^T}(\boldsymbol{q_i}) \\ \boldsymbol{\nabla C}(\boldsymbol{q_i}) & \tilde{\boldsymbol{\alpha}} \end{pmatrix} \begin{pmatrix} \boldsymbol{\Delta p} \\ \boldsymbol{\Delta\lambda} \end{pmatrix} = - \begin{pmatrix} \mathbf{0} \\ \boldsymbol{h}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) \end{pmatrix}. \tag{3.39}$$

After the first iteration, the upper row of Eq. (3.39) is satisfied. Thus, after the first iteration with $\boldsymbol{q}_0 = \tilde{\boldsymbol{q}}$ and $\boldsymbol{\lambda}_0 = \mathbf{0}$, it is

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

Note how the last term is almost identical to $\boldsymbol{g}(\boldsymbol{q}_1, \boldsymbol{\lambda}_1) = \boldsymbol{M}(\boldsymbol{q}_1 - \tilde{\boldsymbol{q}}) - \nabla\boldsymbol{C}(\boldsymbol{q}_1)^T\boldsymbol{\lambda}_1$, the only difference being that $\nabla\boldsymbol{C}(\boldsymbol{q}_0)^T$ is replaced by $\nabla\boldsymbol{C}(\boldsymbol{q}_1)^T$. Thus, Macklin et al. [MMC16] argue that $\boldsymbol{g}(\boldsymbol{q}_1, \boldsymbol{\lambda}_1) \approx \mathbf{0}$ as well, as long as $\nabla\boldsymbol{C}(\boldsymbol{q})$ does not change too quickly.

Since the mass matrix $\boldsymbol{M}$ in the upper-left block of the system matrix of Eq. (3.39) is invertible by design, it is possible to take the Schur complement with respect to $\boldsymbol{M}$ to obtain a reduced system in terms of $\Delta\boldsymbol{\lambda}$:

$$(\nabla\boldsymbol{C}(\boldsymbol{q}_i)\boldsymbol{M}^{-1}\nabla\boldsymbol{C}(\boldsymbol{q}_i)^T + \tilde{\boldsymbol{\alpha}})\Delta\boldsymbol{\lambda} = -\boldsymbol{C}(\boldsymbol{q}_i) - \tilde{\boldsymbol{\alpha}}\boldsymbol{\lambda}_i \tag{3.40}$$

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

$$\Delta\boldsymbol{q} = \boldsymbol{M}^{-1}\nabla\boldsymbol{C}(\boldsymbol{q}_i)^T\Delta\boldsymbol{\lambda}. \tag{3.41}$$

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 Eq. (3.40) for a single constraint C_j and get the update for its scalar Lagrange multiplier λ_j by computing

$$\Delta\lambda_j = \frac{-C_j(\boldsymbol{q}_i) - \tilde{\alpha}_j\lambda_{ji}}{\nabla C_j(\boldsymbol{q}_i)\boldsymbol{M}^{-1}\nabla C_j(\boldsymbol{q}_i)^T + \tilde{\alpha}_j}. \tag{3.42}$$

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

$$\Delta\boldsymbol{q}_l = w_l\nabla_{\boldsymbol{q}_l}\boldsymbol{C}_j(\boldsymbol{q}_i)^T\Delta\boldsymbol{\lambda}_j. \tag{3.43}$$

The position update remains unchanged from Eq. (3.41), but note that $\Delta\boldsymbol{\lambda}$ is a scalar once only a single constraint C_j is considered. Thus, the position update is a multiple of the mass-weighted gradient, just like in PBD (Eq. (3.26)).

In summary, we simply compute $\Delta\lambda_j$ via Eq. (3.42) and use it to update $\lambda_{j+1} = \lambda_j + \Delta\lambda$ and to determine $\Delta\boldsymbol{q}$ via Eq. (3.41) 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 Eq. (3.41) and Eq. (3.42) in order to maintain notational consistency with the PBD solver in Algorithm 8.

Algorithm 9 xPBD Constraint Solver

procedure PROJECTCONSTRAINTS($C_1, \ldots, C_r, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_m$)
  for all constraints C_j **do** $\lambda_j = 0$
  for all iterations n_s **do**
    for all constraints C_j with cardinality n_j, particle indices $i_1, \ldots, i_{n_j}$, Lagrange multiplier λ_j **do**
      if C_j is an inequality constraint and $C_j(\boldsymbol{p}) \geq 0$ **then**
        continue to next constraint
      end if
      $\Delta\lambda_j = \frac{-C_j(\boldsymbol{p}) - \tilde{\alpha}_j \lambda_j}{\nabla C_j(\boldsymbol{p}) \boldsymbol{M}^{-1} \nabla C_j(\boldsymbol{p})^T + \tilde{\alpha}_j}$
      $\lambda_j = \lambda_j + \Delta\lambda_j$
      for all particles $i \in \{i_1, \ldots, i_{n_j}\}$ **do**
        $\Delta\boldsymbol{p}_i = -\frac{C_j(\boldsymbol{p})}{\sum_{i \in \{i_1, \ldots, i_{n_j}\}} w_i |\nabla_{\boldsymbol{p}_i} C_j(\boldsymbol{p})|^2} w_i \nabla_{\boldsymbol{p}_i} C_j(\boldsymbol{p})$
        $\boldsymbol{p}_i = \boldsymbol{p}_i + \Delta\boldsymbol{p}_i$
      end for
    end for
  end for
  return with result $\boldsymbol{p}$
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 $\boldsymbol{\alpha} = \boldsymbol{0}$. Indeed, a closer look at the PBD update formula Eq. (3.26) and the xPBD update formulas in Eq. (3.42), Eq. (3.43) 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 Eq. (3.42) 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.

Another advantage of the compliant constraint formulation is that large stiffness values can be often be hidden inside of the Lagrange multipliers $\boldsymbol{\lambda}$ (Eq. (3.35)), which are then pulled out of the system matrix of resulting LSEs by treating $\boldsymbol{\lambda}$ as an unknown. This is also exploited in the xPBD derivation in Eq. (3.38). Note how the lower-right block of the system matrix only contains the small compliance term $\tilde{\boldsymbol{\alpha}}$. Additionally, $\nabla \boldsymbol{C}(\boldsymbol{q}_i)$ in the lower-left and upper-right blocks should not have any large entries since the large stiffness values are pulled out of the constraints in $Eq.$ (3.32). However, the upper-left block of the system matrix $\boldsymbol{K}$ given by

$$\boldsymbol{K} = \frac{\delta \boldsymbol{g}}{\delta \boldsymbol{q}}(\boldsymbol{q}_i) = \boldsymbol{M} - \frac{\delta \nabla \boldsymbol{C}(\boldsymbol{q_i})^T \boldsymbol{\lambda_i}}{\delta \boldsymbol{q}_i} \tag{3.44}$$

still contains references to $\boldsymbol{\lambda}$ which contains the potentially large constraint stiffnesses. Due to the first assumption $\boldsymbol{K} \approx \boldsymbol{M}$, these occurrences of $\boldsymbol{\lambda}$ are also removed and the system matrix of the simplified LSE in Eq. (3.39) is specified in terms of small compliances only. As a result, the simplified LSE has favorable numerical properties (Section 3.3.4).

The discussion above highlights the importance of pulling large stiffness values out of the constraints in Eq. (3.32). The potential $\psi_j(\boldsymbol{q}) = \frac{1}{2}\alpha_j^{-1} C_j(\boldsymbol{q})^2$ could also be written as $\psi_j(\boldsymbol{q}) = \frac{1}{2} C'_j(\boldsymbol{q})^2$ in terms of the alternative constraint function C'_j given by

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

However, this would result in large constraint gradients $\nabla \boldsymbol{C}'(\boldsymbol{q}_i)$, meaning that even the system matrix of the simplified LSE (Eq. (3.39)) would have large entries.

- Discuss the justifications for the assumptions
  - Convergence properties of Quasi-Newton methods for NLSE. Hessian approximation eventually needs to be close to the Hessian at the solution to guarantee convergence. Whether $\boldsymbol{K} \approx \boldsymbol{M}$ is difficult to say, but it is clear that pulling out large stiffness terms helps with keeping the magnitudes of the entries of $\boldsymbol{K}$ small, that the error from the first approximation is kept smaller as well. Either way, these convergence properties are for Newton's method where the LSE is solved directly.
  - I believe that assumption 1 simply assumes that ∇C is constant across the time step

  - Even though constraint gradients don't change too drastically if large stiffness values are pulled out, they appear in a product with λ, which usually end up being pretty large for constraint gradients. Thus, it is likely that the value of $\boldsymbol{g}$ is sensitive even to small changes in the constraint gradients (or alternatively small constraint Hessians).
  - The justification of the second assumption is based on the choice $\boldsymbol{q}_0 = \tilde{\boldsymbol{q}}, \boldsymbol{\lambda} = \mathbf{0}$. It seems quite obvious that $\boldsymbol{\lambda} = \mathbf{0}$ is not a particularly good initial guess if Eq. (3.35) is taken into account.
  - Due to the fact that we assume $\boldsymbol{g}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) = 0$, all occurrences of $\tilde{\boldsymbol{q}}$ are removed from the update equations of xPBD. $\tilde{\boldsymbol{q}}$ is only used as the initial guess. Due to the compliance terms in the update formulas, the degree to which individual momenta can be washed out is reduced. My notes show that increasing masses by a constant factor actually does decrease the size of the position updates taken, but the effect is hard to quantify. Still, the masses only affect how mobile a particle is in general, without taking directionality into account. Moving towards the initial position should be rewarded, moving away from it should be punished. This is not possible if $\tilde{\boldsymbol{q}}$ does not appear in the update formulas.

- Derivation is based on implicit Euler integration, numerical damping is introduced and depends on the chosen time step. In that sense, the stiffness does stil depend on the time step.
- λ only correspond to force magnitudes if there the constraint gradients are unit vectors. Tempted to think that if only one type of constraints is used, λ can still be picked to represent the force magnitude, but this would only be true if the norm of the constraint gradients would be constant across the entire domain, which is generally not the case.
- General properties of Gauss-Seidel solver apply here again.

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 timestep. 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*), a different approach is used. Energy potentials are restricted to a specific structure which allow

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 timesteps. 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.

3.5.1 Energy Potentials

Let the positions of m particles in a mesh be stored in a matrix $\boldsymbol{q} \in \mathbb{R}^{m\times 3}$ with deformation gradient $\boldsymbol{F} \coloneqq \boldsymbol{F}(\boldsymbol{q}) \in \mathbb{R}^{3\times 3}$. Then, energy potentials of the general form $\psi(\boldsymbol{E}(\boldsymbol{F}))$, where $\boldsymbol{E}(\boldsymbol{F})$ is a strain measure that depends on the deformation gradient of a discrete element, are frequently used in nonlinear continuum mechanics. If $\boldsymbol{E}$ is Green's strain measure $\boldsymbol{E}_{\text{Green}}$ defined by

$$\boldsymbol{E}_{\text{Green}}(\boldsymbol{F}) = \frac{1}{2}(\boldsymbol{F}^{\boldsymbol{T}}\boldsymbol{F} - \boldsymbol{I})$$

then $\boldsymbol{E}_{\text{Green}}(\boldsymbol{F}) = 0$ is equivalent to $\boldsymbol{F}^{\boldsymbol{T}}\boldsymbol{F} = \boldsymbol{I}$. Thus, $\boldsymbol{E}_{\text{Green}}(\boldsymbol{F}) = 0$ defines a constraint manifold that accepts deformations whose deformation gradients $\boldsymbol{F}$ are rotation matrices. These deformations are exactly the rigid-body transforms, i.e. transforms that alter the body's position and orientation but keep the body's volume undeformed. Assuming that $\psi(\boldsymbol{0}) = \rho$ for some $\rho \in \mathbb{R}$ and that ψ reaches its minimum at the undeformed configuration, then

$$d(\boldsymbol{E}_{\text{Green}}(\boldsymbol{F})) = \psi(\boldsymbol{E}_{\text{Green}}(\boldsymbol{F})) - \rho$$

can be considered a distance measure of how far the configuration is away from the constraint manifold defined by the undeformed configurations.

The energy potentials in PD are designed to fit into this framework [BML+14]: Energy potentials are defined by a constraint manifold $\boldsymbol{C}$ – which can be different from $\boldsymbol{E}_{\text{Green}}(\boldsymbol{F}) = 0$ – and a distance measure d of the body's current configuration to that constraint manifold. Formally, this leads to energy potentials which of the following form:

$$\psi(\boldsymbol{q}) = \min_{\boldsymbol{p}} d(\boldsymbol{q}, \boldsymbol{p}) + \delta_{\boldsymbol{C}}(\boldsymbol{p}).$$

Here, $\boldsymbol{p} \in \mathbb{R}^{r\times 3}, r \in \mathbb{N}$ are auxiliary projection variables and $\delta_{\boldsymbol{C}}(\boldsymbol{p})$ is an indicator function with

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

Define $\boldsymbol{p_q}$ such that $\psi(\boldsymbol{q}) = d(\boldsymbol{q}, \boldsymbol{p_q}) + \delta_{\boldsymbol{C}}(\boldsymbol{p_q})$. Then obviously $\delta_{\boldsymbol{C}}(\boldsymbol{p_q}) = 0$, meaning that $\boldsymbol{p_q}$ lies on $\boldsymbol{C}$. Together, $\boldsymbol{p_q}$ is the configuration on the constraint manifold $\boldsymbol{C}$ with minimal distance $d(\boldsymbol{q}, \boldsymbol{p_q})$ to current configuration $\boldsymbol{q}$. Consequently, $\psi(\boldsymbol{q})$ measures the distance of $\boldsymbol{q}$ to the constraint manifold $\boldsymbol{C}$.

The authors claim that since the constraint manifolds already capture nonlinearities the need for complicated distance functions d can be relaxed while still achieving visually plausible simulations [BML+14]. In PD, distance measures d are restricted to quadratic functions of the form

$$d(\boldsymbol{q}) = \frac{w}{2} \|\boldsymbol{G}\boldsymbol{q} - \boldsymbol{p}\|_F^2 , \tag{3.45}$$

where $\boldsymbol{G} \in \mathbb{R}^{r \times m}$ for some $r \in \mathbb{N}$ and w is the constraint stiffness. In summary, PD energy potentials have the following form:

$$\psi(\boldsymbol{q}) = \min_{\boldsymbol{p}} \frac{w}{2} \|\boldsymbol{G}\boldsymbol{q} - \boldsymbol{p}\|_F^2 + \delta_{\boldsymbol{C}}(\boldsymbol{p}). \tag{3.46}$$

Example: Strain Energies

As an elucidating example, we briefly recap how to formulate strain energies in terms of PD energy potentials (Eq. (3.46)) 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 is a 3-manifold simplicial complex, the continuous strain energy can be discretized across the tetrahedra according to (refer to appropriate section). The energy for a single tetrahedradron can be approximated in terms of PD energy potentials (Eq. (3.46)) by setting $\boldsymbol{G} \in \mathbb{R}^{3 \times m}$ to the matrix $\boldsymbol{A}_i$ that maps $\boldsymbol{q}$ to the transpose of the deformation gradient of the tetrahedron $\boldsymbol{F}^T$ and $\boldsymbol{C}$ to the set of rotational matrices SO(3). Typically, the weight $w = kV$, where V is the volume of the undeformed tetrahedron and k is a user-defined stiffness value. This yields the following energy potential, which we also call PD strain potential from now on:

$$\psi(\boldsymbol{q}) = \min_{\boldsymbol{p}} \frac{w}{2} \|\boldsymbol{A}\boldsymbol{q} - \boldsymbol{p}\|_F^2 + \delta_{\mathrm{SO(3)}}(\boldsymbol{p}) = \min_{\boldsymbol{p}} \frac{w}{2} \left\| \boldsymbol{F}^T - \boldsymbol{p} \right\|_F^2 + \delta_{\mathrm{SO(3)}}(\boldsymbol{p}). \tag{3.47}$$

It is easy to show that this energy is zero if and only if $\boldsymbol{F}$ itself is a rotational matrix and grows as the singular values of $\boldsymbol{F}$ move away from 1 [BML+14]. Informally, the energy increases the more the tetrahedron gets stretched or squashed. The

projection $\boldsymbol{p}$ can be computed as $\boldsymbol{p} = \boldsymbol{U}\boldsymbol{I}\boldsymbol{V}^T$ where $\boldsymbol{F}^T = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ is the singular value decomposition of the inverse deformation gradient $\boldsymbol{F}^T$.

3.5.2 Projective Implicit Euler Solver

We start by substituting the PD energy potentials (Eq. (3.46)) into the variatonal form of implicit Euler integration (Eq. (3.6)). With abuse of notation, let $\boldsymbol{p}_j$ denote 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. This yields the following joint optimization problem over the positions $\boldsymbol{q}$ and auxiliary variables $\boldsymbol{p}_j$

$$\min_{\boldsymbol{q},\boldsymbol{p}_j} \tilde{g}(\boldsymbol{q},\boldsymbol{p}_j) = \min_{\boldsymbol{q},\boldsymbol{p}_j} \frac{1}{2h^2}\left\|\boldsymbol{M}^{1/2}(\boldsymbol{q}-\boldsymbol{s}_n)\right\|_F^2 + \sum_j \frac{w_j}{2}\left\|\boldsymbol{G}_j\boldsymbol{q}-\boldsymbol{p}_j\right\|_F^2 + \delta_{\boldsymbol{C}_j}(\boldsymbol{p}_j). \tag{3.48}$$

This optimization problem is optimized using a local/global alternating minimization technique. Local and global steps are carried out sequentially for a fixed number of iterations during each timestep.

The local step consists of minimizing the objective function Eq. (3.48) over the auxiliary variables $\boldsymbol{p}_j$ while keeping the positions $\boldsymbol{q}$ fixed. This corresponds to finding the projection points of the current positions onto the constraint manifolds used to define the PD energy potentials. Each constraint has its own set of auxiliary variables, meaning that the projection steps can be carried out independently. For each energy potential, we solve the following minimization problem

$$\min_{\boldsymbol{p}_j} \tilde{g}(\boldsymbol{q},\boldsymbol{p}_j) = \min_{\boldsymbol{p}_j} \frac{w_j}{2}\left\|\boldsymbol{G}_j\boldsymbol{q}-\boldsymbol{p}_j\right\|_F^2 + \delta_{\boldsymbol{C}_j}(\boldsymbol{p}_j). \tag{3.49}$$

In the global step, the minimization problem Eq. (3.48) is optimized over the positions $\boldsymbol{q}$ while keeping the auxiliary variables $\boldsymbol{p}_j$ fixed. This corresponds to moving the positions $\boldsymbol{q}$ according to their momentum and external forces while trying to maintain short distances to the projections points as defined by the distance measures of the PD energy potentials. The optimization problem for the global solve is given by

$$\min_{\boldsymbol{q}} \tilde{g}(\boldsymbol{q},\boldsymbol{p}_j) = \min_{\boldsymbol{q}} \frac{1}{2h^2}\left\|\boldsymbol{M}^{1/2}(\boldsymbol{q}-\boldsymbol{s}_n)\right\|_F^2 + \sum_j \frac{w_j}{2}\left\|\boldsymbol{G}_j\boldsymbol{q}-\boldsymbol{p_j}\right\|_F^2. \tag{3.50}$$

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

$$\nabla_{\boldsymbol{q}}\tilde{g}(\boldsymbol{q},\boldsymbol{p}_j) = \frac{\boldsymbol{M}}{h^2}(\boldsymbol{q}-\boldsymbol{s}_n) + \sum_j w_j \boldsymbol{G}_j^T \boldsymbol{G}_j \boldsymbol{q} + \sum_j w_j \boldsymbol{G}_j^T \boldsymbol{p}_j. \tag{3.51}$$

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

$$(\frac{\boldsymbol{M}}{h^2} + \sum_j w_j \boldsymbol{G}_j^T \boldsymbol{G}_j)\boldsymbol{q} = \frac{\boldsymbol{M}}{h^2}\boldsymbol{s}_n + \sum_j w_j \boldsymbol{G}_j^T \boldsymbol{p}_j. \tag{3.52}$$

In the rest of Section 3.5 we refer to the system matrix of the global system by $\boldsymbol{S} \coloneqq \frac{\boldsymbol{M}}{h^2} + \sum_j w_j \boldsymbol{G}_j^T \boldsymbol{G}_j$.

Note that $\boldsymbol{S}$ is constant as long as the constraint set remains unchanged. The right side needs to be recomputed in every iteration as the projections $\boldsymbol{p}_j$ change during the local optimization steps. An overview over the algorithm is given in Algorithm 10.

Algorithm 10 Projective Implicit Euler Solver

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

3.5.3 Properties of Projective Dynamics

The structure of the PD energy potentials allows for Algorithm 10 to be implemented efficiently. Since the constraint projections in Eq. (3.49) can be carried out independently, the local optimization step lends itself to massive parallelization. Further, because the system matrix $\boldsymbol{S}$ is constant its prefactorization can be computed at initialization, enabling efficient solves of the linear system in the

global optimization step. Note that since $\boldsymbol{q} \in \mathbb{R}^{m \times 3}$ in Eq. (3.52), the global system can be solved independently and in parallel for each coordinate.

The last property follows from the fact that the distance measure d in Eq. (3.45) has 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. Additionally, 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 $\boldsymbol{S}$.

While a simplified minimization problem is constructed by restricting to PD energy potentials, the solver does converge to a true solution of Eq. (3.48). 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 **??**). 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 $\boldsymbol{S}$ 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.

3.5.4 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. (3.48)) is solved by using a specialized local/global alternating minimization technique (Section 3.5.2). If the projection points $\boldsymbol{p}_{\boldsymbol{q}}^j$ with

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

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

$$\min_{\boldsymbol{q}} g(\boldsymbol{q}) = \min_{\boldsymbol{q}} \frac{1}{2h^2}\left\|\boldsymbol{M}^{1/2}(\boldsymbol{q} - \boldsymbol{s}_n)\right\|_F^2 + \sum_j \frac{w_j}{2}\left\|\boldsymbol{G}_j\boldsymbol{q} - \boldsymbol{p}_j(\boldsymbol{q})\right\|_F^2 \tag{3.53}$$

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). In fact, it can be shown that the PD solver applied to Eq. (3.48) is a special case of a Quasi-Newton method with constant Hessian approximation applied to Eq. (3.53) [LBK17].

The gradient ∇g of the objective function g from Eq. (3.53) 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

$$\nabla g(\boldsymbol{q}) = \frac{\boldsymbol{M}}{h^2}(\boldsymbol{q} - \boldsymbol{s}_n) + \sum_j w_j \boldsymbol{G}_j^T\boldsymbol{G}_j\boldsymbol{q} + \sum_j w_j \boldsymbol{G}_j^T\boldsymbol{p}_j(\boldsymbol{q}). \tag{3.54}$$

Surprisingly, this is the same as the gradient $\nabla_{\boldsymbol{q}}\tilde{g}(\boldsymbol{q}, \boldsymbol{p}_j)$ of the global optimization problem of the original PD algorithm (Eq. (3.51)). Thus, the gradient

is unaffected by whether $\boldsymbol{p}_j$ is considered constant or a function $\boldsymbol{p}_j(\boldsymbol{q})$ of the positions $\boldsymbol{q}$. In Newton's method, the search direction for the current iteration $\boldsymbol{r}_k^N$ is given by $-(\nabla^2 g(\boldsymbol{q}))^{-1}\nabla g(\boldsymbol{q})$, which is a descent direction if $\nabla^2 g(\boldsymbol{q})$ is positive definite (Section 3.2.1). In Quasi-Newton methods, the true Hessian $\nabla^2 g(\boldsymbol{q})$ is approximated by some positive definite matrix $\boldsymbol{B}_k$ instead (Section 3.2.1). If $\boldsymbol{B}_k = \boldsymbol{S}$ – i.e. the system matrix (Eq. (3.52)) from the global optimization in PD – is used with step size $\alpha = 1$ the following update is recovered:

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

Note that

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

is exactly the solution of the global linear system from PD in Eq. (3.52). Together

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

shows that performing a Quasi-Newton step with Hessian approximation $\boldsymbol{B}_k = \boldsymbol{S}$ and step size $\alpha_k = 1$ on the minimization problem in Eq. (3.53) 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.

Algorithm 11 Projective Dynamics as a Quasi-Newton Method

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

This insight that PD is a special case of Quasi-Newton methods applied to $Eq.$ (3.53) 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 3.5.4, we discussed that the PD solver for the minimization problem Eq. (3.48) can be interpreted as a special case of a Quasi-Newton method with constant Hessian approximation and step size $\alpha = 1$ applied to the corresponding minimization problem Eq. (3.53). The minimization problem for the Quasi-Newton method

$$\min_{\boldsymbol{q}} g(\boldsymbol{q}) = \min_{\boldsymbol{q}} \frac{1}{2h^2} \left\| \boldsymbol{M}^{1/2}(\boldsymbol{q} - \boldsymbol{s}_n) \right\|_F^2 + \sum_j \frac{w_j}{2} \left\| \boldsymbol{G}_j \boldsymbol{q} - \boldsymbol{p_j}(\boldsymbol{q}) \right\|_F^2 \tag{3.55}$$

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

$$\boldsymbol{S} \coloneqq \frac{\boldsymbol{M}}{h^2} + \sum_j w_j \boldsymbol{G}_j^T \boldsymbol{G}_j \tag{3.56}$$

are stated here once more for convenience.

As the Hessian matrix $\nabla^2 g(\boldsymbol{q}_k)$ 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 $\boldsymbol{S}$, 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. (3.6)), 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 $\boldsymbol{S}$ anymore. Liu et al. suggest a way to emulate the global system matrix 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 $\boldsymbol{H}_0$ 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 picked heuristically instead. This suggests that finding more suitable candidates

for $\boldsymbol{H}_0$ – or alternatively for $\boldsymbol{B}_0$ – is an avenue towards improving the convergence properties of Quasi-Newton methods. It stands to reason that due to its effectiveness in PD, $\boldsymbol{S}$ is a viable choice for the initial Hessian approximation $\boldsymbol{B}_0$ for the minimization of Eq. (3.55) [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 $\boldsymbol{B}_0 = \boldsymbol{S}$ [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 $\boldsymbol{s}, \boldsymbol{y}, \rho$ 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($\boldsymbol{q}_n$, $\boldsymbol{v}_n$, $\boldsymbol{f}_{\text{ext}}$, m, h)
    $\boldsymbol{s}_n = \boldsymbol{q}_n + h\boldsymbol{v}_n + h^2\boldsymbol{M}^{-1}\boldsymbol{f}_{\text{ext}}$
    $\boldsymbol{q}_k = \boldsymbol{s}_n$
    for all iterations **do**
        $\boldsymbol{s_k} = \boldsymbol{q}_k - \boldsymbol{q}_{k-1}, \boldsymbol{y}_k = \nabla g(\boldsymbol{q}_k) - \nabla g(\boldsymbol{q}_{k-1}), \rho_k = 1/(\boldsymbol{s}_k^T\boldsymbol{y}_k)$
        $\boldsymbol{p}_k =$ TWOLOOPRECURSION$(\boldsymbol{S}, \boldsymbol{q}_k, \boldsymbol{s}, \boldsymbol{y}, \rho, m, k)$ (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$(\boldsymbol{q}_k, \boldsymbol{p}_k, 1, \beta, t)$ (Algorithm 6)
        end if
        $\boldsymbol{q}_k = \boldsymbol{q}_k + \alpha_k\boldsymbol{p}_k$
    end for
    return with result $\boldsymbol{q}_{n+1} = \boldsymbol{q}_k, \boldsymbol{v}_{n+1} = (\boldsymbol{q}_{n+1} - \boldsymbol{q}_n)/h$
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 $\boldsymbol{S}$, it is not viable in L-BFGS. The step-size $\alpha_k = 1$ is not guaranteed to satisfy the strong Wolfe conditions (Eq. (3.7), Eq. (3.8)) if the L-BFGS update is used. However, the strong Wolfe conditions are required to ensure that the curvature condition (Eq. (3.14)) is satisfied and $\boldsymbol{H}_k$ 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 $\boldsymbol{H}_k$ that arise from the L-BFGS method with initial matrix $\boldsymbol{B}_0 = \boldsymbol{S}$ 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. (3.6)) without the need to restrict to PD energy potentials (Eq. (3.46)), a suitable candidate for the initial Hessian approximation $\boldsymbol{B}_0$ or its inverse $\boldsymbol{H}_0$ needs to be found. The choice $\boldsymbol{B}_0 = \boldsymbol{S} = \frac{\boldsymbol{M}}{h^2} + \sum_j w_j \boldsymbol{G}_j^T \boldsymbol{G}_j$ from Algorithm 12 cannot be applied to the general setting without modification as the matrices $\boldsymbol{G}_j$ 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 3.5.1, the strain energy of a single tetrahedron can be emulated with special PD strain potentials (Eq. (3.47)). Here $\boldsymbol{G}_j = \boldsymbol{A}_j$, where $\boldsymbol{A}_j$ is the linear operator that maps $\boldsymbol{q}$ to the tranpose of the deformation gradient $\boldsymbol{F}_j^T$, $\boldsymbol{C}_j = \mathrm{SO}(3)$ 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 $\boldsymbol{S}$ that arises as the initial Hessian approximation $\boldsymbol{B}_0$ given by

$$\boldsymbol{B}_0 = \boldsymbol{S} = \frac{\boldsymbol{M}}{h^2} + \sum_j w_j \boldsymbol{A}_j^T \boldsymbol{A}_j. \tag{3.57}$$

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 $\boldsymbol{F_j}$ 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

$$\Psi(\sigma_1, \sigma_2, \sigma_3) = a(\sigma_1) + a(\sigma_2) + a(\sigma_3) + b(\sigma_1\sigma_2) + b(\sigma_2\sigma_3) + b(\sigma_1\sigma_3) \tag{3.58}$$

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

$$\left.\frac{d\psi}{d\sigma_i}\right|_{\sigma_j,\sigma_k=1} = a\prime(\sigma_i) + 2b\prime(\sigma_i) + c\prime(\sigma_i) \tag{3.59}$$

$$\left.\frac{d^2\psi}{d\sigma_i}\right|_{\sigma_j,\sigma_k=1} = a\prime\prime(\sigma_i) + 2b\prime\prime(\sigma_i) + c\prime\prime(\sigma_i), \tag{3.60}$$

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

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

runs into issues as the expression often evaluates to zero, even in common non-pathological cases. This issue arises because the second derivative in Eq. (3.60) 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 Eq. (3.59) by computing the slope of its best linear approximation over the interval $[\sigma_{\text{min}}, \sigma_{\text{max}}]$. Formally, k is defined by

$$k :=_k \int_{\sigma_{\text{min}}}^{\sigma_{\text{max}}} (k(\sigma - 1) - (a\prime(\sigma) + 2b\prime(\sigma) + c\prime(\sigma))^2 d\sigma \tag{3.61}$$

Using these stiffness values in Eq. (3.57) 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 $\boldsymbol{B}_0$, the function evaluations $g(\boldsymbol{q}_k)$ and gradients $\nabla g(\boldsymbol{q}_k)$ 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. ISBN: 0897917464. DOI: 10.1145/237170.237226. URL: https://doi.org/10.1145/237170.237226 (cited on pages 21–23).

[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. ISBN: 0897919998. DOI: 10.1145/280814.280821. URL: https://doi.org/10.1145/280814.280821 (cited on pages 19, 20, 22, 25).

[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). ISSN: 0730-0301. DOI: 10.1145/2601097.2601116. URL: https://doi.org/10.1145/2601097.2601116 (cited on pages 8, 29, 31, 38, 39, 42).

[CC05] Steven C. Chapra and Raymond Canale. *Numerical Methods for Engineers*. 5th ed. USA: McGraw-Hill, Inc., 2005. ISBN: 0073101567 (cited on pages 6, 7).

[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). ISSN: 0730-0301. DOI: 10.1145/3072959.2990496. URL: https://doi.org/10.1145/3072959.2990496 (cited on pages 17, 42, 43, 46–48).

[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. ISBN: 9781450345927. DOI: 10.1145/2994258.2994272. URL: https://doi.org/10.1145/2994258.2994272 (cited on pages 25, 27, 32–35).

[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. ISBN: 3-905673-61-4. DOI: 10.2312/PE/vriphys/vriphys06/071-080 (cited on pages 25–31).

[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. DOI: https://doi.org/10.1111/cgf.14105. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14105. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14105 (cited on page 22).

[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. DOI: https://doi.org/10.1111/j.1467-8659.2006.01000.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2006.01000.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2006.01000.x (cited on page 20).

[NW06] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. 2e. New York, NY, USA: Springer, 2006 (cited on pages 8–18, 41, 47).

[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. DOI: https://doi.org/10.1002/cpa.3160100103. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpa.3160100103. URL: https:

//onlinelibrary.wiley.com/doi/abs/10.1002/cpa.3160100103 (cited on page 20).

[SLM06] Martin Servin, Claude Lacoursière, and Niklas Melin. "Interactive Simulation of Elastic Deformable Materials". In: *Proc. SIGRAD* (Jan. 2006) (cited on pages 6, 7, 20, 23, 25, 33, 35).

[SB12] Eftychios Sifakis and Jernej Barbic. "FEM simulation of 3D deformable solids: a practitioner's guide to theory, discretization and model reduction". In: *ACM SIGGRAPH 2012 Courses*. SIGGRAPH '12. Los Angeles, California: Association for Computing Machinery, 2012. ISBN: 9781450316781. DOI: 10.1145/2343483.2343501. URL: https://doi.org/10.1145/2343483.2343501 (cited on page 47).

[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. ISBN: 1595933646. DOI: 10.1145/1185657.1185669. URL: https://doi.org/10.1145/1185657.1185669 (cited on pages 6, 7).

[TPBF87] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. "Elastically Deformable Models". In: *SIGGRAPH Comput. Graph.* 21.4 (Aug. 1987), pp. 205–214. ISSN: 0097-8930. DOI: 10.1145/37402.37427. URL: https://doi.org/10.1145/37402.37427 (cited on page 20).

[TNGF15] Maxime Tournier, Matthieu Nesme, Benjamin Gilles, and François Faure. "Stable Constrained Dynamics". In: *ACM Trans. Graph.* 34.4 (July 2015). ISSN: 0730-0301. DOI: 10.1145/2766969. URL: https://doi.org/10.1145/2766969 (cited on pages 19, 22–24).

[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. DOI: 10.1080/08905459508905232. eprint: https://doi.org/10.1080/08905459508905232. URL: https://doi.org/10.1080/08905459508905232 (cited on pages 21, 22).

Index

PD, 37