
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, March 19, 2024

Dennis Ledwon

# Contents

# Chapter 1

# Introduction

# Chapter 2

# Related Work

# Chapter 3

# Method

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

## 3.1 Time-Integration of Physical Systems

Fabian recommends citing [GSS+15].

In most approaches for the simulation of physical systems, the motion of the system is assumed to be in accordance with Newton's laws of motion. Maybe list all of them. Newton's first law is definitely going to be mentioned, not sure about the third one. I don't want to spend too much time on contact. Due to Newton's second law

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

it is possible to relate the force $\boldsymbol{f}$ acting on a particle and the resulting acceleration $\boldsymbol{a}$ via the particle mass $m$. The motion of a particle system can then be described in terms of a system of ordinary differential equations (ODEs). This system of ODEs is commonly referred to as the equations of motion. The equations of motion are integrated over time in order to arrive at the configuration of the system at the next time step. 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 [MMC16; BML+14]. The equations of motions due to Newton's second law are reviewed in Section 3.1.1. Common approaches for numerical integration are 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 equations of motion

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

where $\boldsymbol{q}(t)$, $\boldsymbol{v}(t)$, $\boldsymbol{f}(\boldsymbol{q}(t), \boldsymbol{v}(t))$ are the particle positions, particle velocities and forces acting on each particle at time $t$, respectively, and $\mathbf{M}$ is a diagonal matrix with the particle masses as diagonal entries. In the context of projective dynamics (Section 3.5), it is $\boldsymbol{q}(t), \boldsymbol{v}(t), \boldsymbol{f}(\boldsymbol{q}(t), \boldsymbol{v}(t)) \in \mathbb{R}^{m\times 3}$ and $\mathbf{M} \in \mathbb{R}^{m\times m}$. In the context of position based dynamics (Section 3.4), it is $\boldsymbol{q}(t), \boldsymbol{v}(t), \boldsymbol{f}(\boldsymbol{q}(t), \boldsymbol{v}(t)) \in \mathbb{R}^{3m}$ and $\mathbf{M} \in \mathbb{R}^{3m\times 3m}$. $\dot{\boldsymbol{q}}(t)$ and $\dot{\boldsymbol{v}}(t)$ are short for $\frac{d\boldsymbol{q}}{dt}(t)$ and $\frac{d\boldsymbol{v}}{dt}(t)$, respectively. From now on, we write $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ instead of $\boldsymbol{q}(t)$ and $\dot{\boldsymbol{q}}(t)$ for time-dependent quantities for the sake of brevity.

The positions $\boldsymbol{q}$ and velocities $\boldsymbol{v}$ of the system at time $t$ can be determind by solving the equations of motion. For general nonlinear forces, analytical solutions are usually not available. Thus, the equations of motion need 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. When applied to Equation 3.2, the positions and velocities are computed at discrete timesteps via the update formula

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

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 without additional damping terms [SD06]. For example, the amplitude 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 the equations of motion, called the symplectic Euler method, arises when the new velocities $\boldsymbol{v}_{n+1}$ instead of the old velocities $\boldsymbol{v}_n$ are used in the position update [SD06]. This leads to the following update formula:

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

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 the equations of motion is implicit Euler integration, resulting in the update formula

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

Note how $\boldsymbol{q}_{n+1}$ and $\boldsymbol{v}_{n+1}$ appear on both sides of the equations. Consequently, performing implicit Euler integration includes solving a system of algebraic equations. For general nonlinear forces, the entire system of equations is nonlinear. 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.

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

summarize the derivation of the optimization problem according to Bouaziz et al. [BML+14] below.

By rewriting the first line of Equation 3.4 as

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

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

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

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 $\psi_j$ 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 Equation 3.5, it is

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

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

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

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

Approaching implicit Euler integration in its variational form can often be advantageous. That is because the objective function of Equation 3.7 presents a natural merit function that can be employed to improve the step size along the search direction that arises in common unconstrained optimization algorithms based on Newton's method [NW06]. As an example, the objective function is used in many step length selection algorithms to ensure that step sizes satisfy the Armijo condition or both Wolfe conditions (Section 3.2.1). In fact, many algorithms for solving

non-linear systems of equations construct approximate merit functions if the objective function from an equivalent optimization problem is not available [NW06].

## 3.2 Unconstrained Optimization

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

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

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

Here, $f$ is called the objective function.

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

### 3.2.1 Line Search Methods

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

Ideally, $\alpha_k$ 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 $\alpha_k$ exactly. Instead, the idea is to compute an approximation of $\alpha_k$ such that the objective function decreases sufficiently and that $\alpha_k$ is close enough to the true minimizer. Formally, these requirements are captured in the strong Wolfe conditions for step lengths $\alpha_k$:

$$f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k) \leq f(\boldsymbol{x}_k) + c_1 \alpha_k \nabla f(\boldsymbol{x}_k)^T \boldsymbol{p}_k \tag{3.8}$$

$$\left| \nabla f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)^T \boldsymbol{p}_k \right| \leq c_2 \left| \nabla f(\boldsymbol{x}_k)^T \boldsymbol{p}_k \right| \tag{3.9}$$

for some constants $c_1 \in (0, 1), c_2 \in (c_1, 1)$. Equation 3.8 is called the sufficient decrease or the Armijo condition and states that the reduction in $f$ should be proportional to both the step length $\alpha_k$ and the directional derivative $\nabla f(\boldsymbol{x}_k)^T \boldsymbol{p}_k$. Informally, the second condition (Eq. (3.9)), known as the curvature condition, ensures that there is no more fast progress to be made along the search direction $\boldsymbol{p}_k$, indicated by the fact that $\left| \nabla f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)^T \boldsymbol{p}_k \right|$ is already rather small.

Step sizes satisfying the strong Wolfe conditions have the following properties under mild assumptions. Firstly, if $\boldsymbol{p}_k$ is a descent direction, it is possible to find a step size that satisfies the strong Wolfe conditions. In particular, the Armijo condition is always satisfied once $\alpha_k$ is sufficiently close to zero. Secondly, it can be shown that line search methods where $\alpha_k$ 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.

### Steepest Descent

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

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

This search direction is called the steepest descent direction. Using the steepest descent direction in Algorithm 1 is called the steepest descent method. While simple, steepest descent exhibits poor performance, even for simple problems. Its convergence rate is only linear and depends on the eigenvalue distribution of the

**Algorithm 1** Line Search Methods

**require** $\epsilon > 0$
**procedure** LINESEARCHMETHOD($\boldsymbol{x}_0$, $\epsilon$)
  $\boldsymbol{x}_k = \boldsymbol{x}_0$
  **while** $\left\|\nabla f(\boldsymbol{x}_k)\right\| > \epsilon$ **do**
    compute a descent direction $\boldsymbol{p}_k$
    compute $\alpha_k$ 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**

Hessian $\nabla^2 f(\boldsymbol{x}^*)$ at the local minimizer $\boldsymbol{x}^*$. If the eigenvalue distribution is wide, steepest descent often requires an unacceptably large number of iterations to find a stationary point.

### Newton's Method

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

$$f(\boldsymbol{x}_k + \boldsymbol{p}) \approx f(\boldsymbol{x}_k) + \boldsymbol{p}^T \nabla f(\boldsymbol{x}_k) + \frac{1}{2}\boldsymbol{p}^T \nabla^2 f(\boldsymbol{x}_k)\boldsymbol{p} =: m_k(\boldsymbol{p}). \tag{3.10}$$

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:

$$\boldsymbol{p}_k^N = -(\nabla^2 f(\boldsymbol{x}_k))^{-1} \nabla f(\boldsymbol{x_k}). \tag{3.11}$$

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

It is easy to show that $\boldsymbol{p}_k^N$ is a descent direction, given that $\nabla^2 f(\boldsymbol{x}_k)$ is positive definite. Otherwise, the Newton direction is not guaranteed to exist, or to be a descent direction if it does. In such cases, the Newton direction cannot be used without modification. Thus, in its naive form, the Newton's method is not globally convergent. However, if $\nabla^2 f(\boldsymbol{x}^*)$ is positive definite at a local solution

$\boldsymbol{x}^*$ and $f$ is twice differentiable, then $\nabla^2 f(\boldsymbol{x})$ is also positive definite for $\boldsymbol{x} \in \mathcal{N}$ for some neighborhood $\mathcal{N}$ of $\boldsymbol{x}^*$. If we have $\boldsymbol{x}_0 \in \mathcal{N}$ for the starting point of $\boldsymbol{x}_0$ of Newton's method and $\boldsymbol{x}_0$ is sufficiently close to the solution $\boldsymbol{x}^*$ it can be shown that Newton's method with step length $\alpha_k = 1$ converges to $\boldsymbol{x}^*$ with a quadratic rate of convergence under mild conditions. Thus, Newton's method has satisfactory convergence properties close to the solution $\boldsymbol{x}^*$ and the Newton direction $\boldsymbol{p}_k^N$ has a natural step size $\alpha_k = 1$ associated with it. Since $\alpha_k = 1$ often does not satisfy the Wolfe conditions when the current iterate $\boldsymbol{x}_k$ is still far away from the solution $\boldsymbol{x}^*$, line searches are still necessary in Newton's method. However, it is recommended to use $\alpha_k = 1$ as the initial guess as $\alpha_k = 1$ guarantees quadratic convergence once $\boldsymbol{x}_k$ gets sufficiently close to $\boldsymbol{x}^*$.

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

**Algorithm 2** Newton's Method

**require** $\epsilon > 0$
**procedure** NEWTONSMETHOD($\boldsymbol{x}_0$, $\epsilon$)
  $\boldsymbol{x}_k = \boldsymbol{x}_0$
  **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** $\alpha_k$ does not satisfy the strong Wolfe conditions **then**
      compute $\alpha_k$ that satisfies the strong Wolfe conditions
    **end if**
    $\boldsymbol{x}_k = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
  **end while**
  **return with result** $\boldsymbol{x}_k$
**end procedure**

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

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

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

### Quasi-Newton Methods

Due to the shortcomings of Newton's method mentioned in Section 3.2.1, it can be favorable to simply approximate the Newton direction in order to find an 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. Often, multiple Quasi-Newton iterations fit into the same time budget as a single Newton iteration. As a result, Quasi-Newton methods can sometimes converge to a solution in a shorter amount of time than Newton's method, even though their search directions are not as effective as the exact Newton direction.

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

$$\boldsymbol{p}_k = -\mathbf{B}_k^{-1}\nabla f(\boldsymbol{x}_k), \tag{3.12}$$

where $\mathbf{B}_k \in \mathbb{R}^{n\times n}$ is positive definite. Note that the Newton direction is a special case of Equation 3.12 with $\mathbf{B}_k = \nabla^2 f(\boldsymbol{x}_k)$. Just like for the Newton direction $\boldsymbol{p}_k^N$ in Equation 3.10, a model function $m_k$ that attains its minimum at $\boldsymbol{p}_k = -\mathbf{B}_k^{-1}\nabla f(\boldsymbol{x}_k)$ can be defined via

$$m_k(\boldsymbol{p}) = f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T\boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^T\mathbf{B}_k\boldsymbol{p}. \tag{3.13}$$

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

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

Rearranging gives

$$\mathbf{B}_k \boldsymbol{s}_{k-1} = \boldsymbol{y}_{k-1}, \tag{3.14}$$

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

$$\boldsymbol{s}_{k-1}^T \boldsymbol{y}_{k-1} > 0, \tag{3.15}$$

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

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

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

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

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

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

$$\mathbf{H}_k = (I - \rho_{k-1}\boldsymbol{s}_{k-1}\boldsymbol{y}_{k-1}^T)\mathbf{H}_{k-1}(I - \rho_{k-1}\boldsymbol{s}_{k-1}\boldsymbol{y}_{k-1}^T) + \rho_{k-1}\boldsymbol{s}_{k-1}\boldsymbol{s}_{k-1}^T, \tag{3.16}$$

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

While global convergence of the BFGS method cannot be established for general nonlinear smooth functions, it is possible to show that it converges superlinearly if the initial guess $\boldsymbol{x}_0$ is close to the solution $\boldsymbol{x}^*$ and $\alpha_k = 1$ for sufficiently

**Algorithm 3** BFGS method

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

large $k$ under mild conditions. Thus, just like Newton's method (Section 3.2.1), the BFGS method has a natural step length $\alpha = 1$, which should be the initial guess for all line search algorithms. Typically, the BFGS method dramatically outperforms steepest descent and performs comparably to Newton's method on many practical problems.

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

Even though BFGS iterations are typically faster to compute than Newton iterations, the BFGS method is still not suitable for large problems in its naive form. Just like in Newton's method, either $\mathbf{H}_k$ or $\mathbf{B}_k$ needs to be stored explicitly, which can be infeasible for large-scale problems. While the BFGS update formula using the inverse matrices $\mathbf{H}_k$ replaces the need for a matrix factorization with a simple matrix-vector multiplication, $\mathbf{H}_k$ and $\mathbf{B}_k$ are generally dense, even if $\nabla^2 f(\boldsymbol{x}_k)$ is sparse. This removes the possibility of alleviating storage requirements and speeding up computations via sparse matrix techniques when using the naive BFGS method.

### Limited-Memory Quasi-Newton Methods

Without knowing all the details I would say that something between a half and full page should suffice on LBFGS (with references to the details of course)

As discussed in Section 3.2.1, the BFGS method is unsuitable for large-scale problems due to the storage requirements of the typically dense inverse Hessian approximation $\mathbf{H}_k$. This highlights the need for effective Hessian approximations that are not only cheap to compute, but also cheap to store. Just like Quasi-Newton methods use approximations of the Newton direction in order to keep the computational cost of a single iteration low, limited-memory Quasi-Newton methods approximate Quasi-Newton directions with the goal of reducing the memory footprint of a single iteration. This comes at the prize of inferior convergence properties. On the upside, limited-memory Quasi-Newton directions can sometimes be computed by using only a couple of vectors of size $n$, without the need to explicitly form the inverse Hessian approximation $\mathbf{H}_k$. This can drop the space complexity of a single iteration to $\mathcal{O}(n)$ compared to $\mathcal{O}(n^2)$ for the BFGS method.

A popular limited-memory method called L-BFGS can be derived from the BFGS update formula for the inverse Hessian approximation $\mathbf{H}_k$ (Eq. (3.16)). Note that the BFGS update in iteration $k$ is specified entirely by the vector pair $(\boldsymbol{s}_n, \boldsymbol{y}_n) \in \mathbb{R}^n$. Consequently, $\mathbf{H}_k$ can be constructed from the initial matrix $\mathbf{H}_0$ and the familiy of vector pairs $((\boldsymbol{s}_i, \boldsymbol{y}_i))_{i\in[0,k-1]}$ by simply performing $k$ update steps according to $Equation$ 3.16. The idea of L-BFGS is to only keep track of the most recent $m$ vector pairs and generate a modified version of the inverse Hessian approximation from the BFGS method by applying the $m$ updates defined by $((\boldsymbol{s}_i, \boldsymbol{y}_i))_{i\in[k-m,k-1]}$ to the initial matrix $\mathbf{H}_0$ at each iteration $k$.

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

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

**Algorithm 4** L-BFGS two-loop Recursion

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

if $\mathbf{B}_0$ is sparse, but not diagonal. In this case, a factorization of $\mathbf{B}_k$ needs to be computed which is not guaranteed to stay sparse.

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

L-BFGS shares many properties with the BFGS method discussed in Section 3.2.1. The performance of the L-BFGS method depends on the choice of the initial matrix $\mathbf{H}_0$, with scaled diagonal matrices being popular choices. Again, there is no generally viable strategy for picking the scaling factor $\gamma \in \mathbb{R}$. Similarly, the initial guess for the step size $\alpha_k = 1$ should be used. The window size $m$ is a parameter that needs to be tuned on a per-problem basis. While the L-BFGS algorithm is generally less robust if $m$ is small, making $m$ arbitrarily large increases the amount of time required to perform the two-loop recursion. If the matrix-vector multiplication in Algorithm 4 is expensive to compute, the additional computational cost incurred by increasing $m$ is usually overshadowed by the matrix-vector multiplication. Still, larger values of $m$ do not necessarily lead to better performance. [LBK17] suggest that curvature information from vector pairs $(\boldsymbol{s}_i, \boldsymbol{y}_i)$ from iterations i with $i << k$ can become out of date, making moderately large values of $m$ more beneficial. The main weakness of the L-BFGS method is its slow convergence on problems where the true Hessian matrices $\nabla^2 f(\boldsymbol{x})_k$ are ill-conditioned.

**Algorithm 5** L-BFGS method

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

### Step Length Selection Algorithms

In Section 3.2.1, the need for step lengths $\alpha_k$ that satisfy the strong Wolfe conditions (Eq. (3.8), Eq. (3.9)) for the convergence of line search methods was discussed. Appropriate step lengths are determined via step length selection algorithms. These algorithms are typically split into two phases. The bracketing phase determines an interval $[\alpha_{\text{min}}, \alpha_{\text{min}}]$ that is guaranteed to contain suitable step lengths. The selection phase is an iterative process that interpolates function values and gradients from previous iterations in order to shrink the interval and eventually pick the final step length. For more details, the reader is referred to Chapter 3 of. As step length algorithms are a common source of bugs, Nocedal and Wright recommend using publically available implementations.

To avoid the complexities of correct step length algorithms, the insight that the Armijo condition (Eq. (3.8)) is always satisfied once $\alpha$ is sufficiently close to zero (Section 3.2.1) can be exploited: If a good first estimate $\alpha = \tilde{\alpha}$ is available, we check whether it satisfies the Armijo condition. Otherwise, $\alpha$ 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 Line Search

**require** $\tilde{\alpha} > 0, c_1 \in (0,1), \beta \in (0,1), t \in (0, \tilde{\alpha})$
**procedure** BACKTRACK($\boldsymbol{x}_k$, $\boldsymbol{p}_k$, $\tilde{\alpha}$, $\beta$, $t$)
  $\alpha = \tilde{\alpha}$
  **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. 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) $\boldsymbol{f} = \mathbf{M}\boldsymbol{a}$, where $\mathbf{M}$ is a $n_d \times n_d$ diagonal matrix and $n_d$ is the total number of independent degrees of freedom for the particles.

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

$$\begin{aligned} \boldsymbol{v}_{n+1} &= \boldsymbol{v}_n + h\boldsymbol{a}_n \\ \boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + h\boldsymbol{v}_{n+1} \end{aligned}$$

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):

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

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

$$\mathbf{H} = \mathbf{M} - h^2\mathbf{K}. \tag{3.17}$$

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

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

### 3.3.2 Penalty Forces

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

Using the geometric displacement function $\phi_{\text{spring}}(\boldsymbol{x}) = (\|\boldsymbol{x_i} - \boldsymbol{x_j}\|) - l$ with $k_{\text{spring}}$ recovers the behavior of a spring with stiffness $k_{\text{spring}}$ (Section 3.3.1). Its displacement function $\phi_{\text{spring}}(\boldsymbol{x})$ is satisfied when the distance of the particles $\boldsymbol{x_i}, \boldsymbol{x_j}$ is equal to a desired rest length $l$. By constructing different geometric displacement functions, various properties such as the bending angle between triangles and in-plane shearing of triangles can be controlled via the corresponding quadratic energy potentials [BW98]. Geometric displacement functions with the desired effect are often intuitive and simple to define. However, as the corresponding energy potentials are not physically derived, choosing stiffness parameters that correspond to measurable physical properties of the simulated material and orchestrating multiple constraints becomes challenging [SLM06; NMK+06]. Additionally, the generated penalty forces do not converge in the limit of infinite stiffess, leading to oscillations unless the 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, the equations of motion (reference somewhere) are handled together with algebraic equations that model the constraints on the positions of the system. Distance constraints are typically implemented using holonomic constraints of the form $\phi(\boldsymbol{x}) = 0$. Note that the distance constraint $\phi(x)$ is formulated in terms of the particle positions, whereas the equations of motion work on velocities and accelerations. Consequently, the constraints need to be differentiated with respect

to time once or twice so that they can be combined with the ODE in terms of velocties or accelerations, respectively. In XPBD, we go the other way! The ODE is tanslated so that it is in terms of positions, so that it can be handled together with the constraints. Is there a reason nobody bothered to do this before? Fabian notes that the translation in terms of positions in XPBD is in fact just rewriting the implicit Euler integration in terms of positions. Thus, we have baked the integration method into the formulation in XPBD. In the DAE framework, there is more flexibility w.r.t. the integration scheme of choice. Using $\mathbf{J} = \frac{\delta \boldsymbol{\phi}}{\delta \boldsymbol{x}}$, where $\mathbf{J}$ is a $n_c \times n_d$ matrix and $n_c$ is the number of scalar constraints, this leads to the following constraint formulations:

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

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

$$\boldsymbol{f_c} = \mathbf{J}^T \boldsymbol{\lambda} \tag{3.18}$$

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

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

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

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

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

Here, the external forces $\boldsymbol{f}_{\text{ext}}$ and the constraint gradients $\mathbf{J}$ are considered constant across the timestep and $\mathbf{J}(x_{n+1})$ is not approximated using the Taylor expansion like it is in [BW98]. If internal forces are taken into account, the upper-left matrix $\mathbf{M}$ is replaced by the matrix $\mathbf{H}$ from Equation 3.17.

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

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

and perform implicit integration:

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

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

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

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

If the constraints are not redundant, $\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T$ is non-singular and symmetric positive definite [Bar96], which are desirable properties for many solvers. According to [SLM06], the common approaches for linearizing the constraint forces and stabilizing the constraints $\boldsymbol{\phi}(\boldsymbol{x}) = 0$ are notoriously unstable (I need to look this up again. I do not understand what exactly this means anymore). Additionally, instabilities in the traverse direction of the constraints occur when the tensile force with respect to particle masses is large when using hard constraints [TNGF15].

### 3.3.4 Compliant Constraints

By combining ideas from hard constraints (Section 3.3.3) and penalty forces (Section 3.3.2), it is possible to formulate the system matrix for hard constraints such that constraints do not have to be enforced exactly. In this approach, called compliant constraints, the constraints are combined with Newton's ODE (Eq. (3.2)) in a way that allows relaxation of constraints in a physically meaningful manner [SLM06]. The key insight is that constraints of the form $C_j(\boldsymbol{q})$ are the physical limit of strong forces from potentials of the form $\frac{k_j}{2}C_j(\boldsymbol{q})^2$ with high stiffness values $k_j$. It can be advantageous to express the system equations in terms of small inverse stiffnesses, also called compliances, over working with large stiffness values directly. In particular, setting the compliance to zero often provides an elegant avenue for modelling infinite stiffness.

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

$$\psi(\boldsymbol{q}) = \frac{1}{2}\boldsymbol{C}(\boldsymbol{q})^T\alpha^{-1}\boldsymbol{C}(\boldsymbol{q}) \tag{3.19}$$

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

$$\boldsymbol{f_c} = \nabla\psi(\boldsymbol{q}) = -\nabla\boldsymbol{C}(\boldsymbol{q})^T\alpha^{-1}\boldsymbol{C}(\boldsymbol{q}). \tag{3.20}$$

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

$$\boldsymbol{f}_c = \nabla\boldsymbol{C}(\boldsymbol{q})^T\boldsymbol{\lambda} \tag{3.21}$$

This leads to the following DAE:

$$\begin{aligned} \dot{\boldsymbol{q}} &= \boldsymbol{v} \\ \mathbf{M}\dot{\boldsymbol{v}} &= \boldsymbol{f}_e + \nabla\boldsymbol{C}(\boldsymbol{q})^T\boldsymbol{\lambda} \\ \alpha\boldsymbol{\lambda}(\boldsymbol{q}) &= -\boldsymbol{C}(\boldsymbol{q}) \end{aligned} \tag{3.22}$$

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

Usually, the DAE is solved by employing some numerical integration scheme (Section 3.1.2) which eventually requires the solution of a linear system of equations. Here, the goal is to arrive at a formulation where both the system matrix and the right side of the resulting LSE do not contain references to the large stiffness values $\alpha^{-1}$. This is achieved by treating $\boldsymbol{\lambda} = -\alpha^{-1}\boldsymbol{C}(\boldsymbol{q})$ as an unknown, pulling it out of the system matrix and hiding all occurrences of the large $\alpha^{-1}$ 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 and the corresponding right side are small, since they do not depend on the large stiffness terms. Backwards differentiation while assuming that $\nabla \boldsymbol{C}$ is constant across the time step yields

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

leading to the following LSE [TNGF15]

$$\begin{pmatrix} \mathbf{M} & -\nabla\boldsymbol{C}(\boldsymbol{q}_n)^T \\ \nabla\boldsymbol{C}(\boldsymbol{q}_n) & \frac{1}{h^2}\alpha \end{pmatrix} \begin{pmatrix} \boldsymbol{v}_{n+1} \\ \boldsymbol{\mu}_{n+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{p} + h\boldsymbol{f}_e \\ -\frac{1}{h}\boldsymbol{C}(\boldsymbol{q}_n) \end{pmatrix}, \tag{3.23}$$

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

This formulation comes with a couple of advantages. Firstly, relaxing the constraints by keeping a finite but large penalty parameter helps counteracting numerical problems in the presence of over defined or degenerate constraints. In comparison to the system matrix that arises for hard constraints (reference), the lower-right zero block is replaced by the compliance matrix $\alpha$ in Equation 3.23 which prevents the system matrix from becoming singular in the presence of redundant constraints. Thus, the compliant constraint formulation can be interpreted as a regularization of the hard constraint formulation in ((reference to section)) [TNGF15]. Secondly, setting $\alpha = \mathbf{0}$ allows modelling infinite stiffness since $\alpha^{-1}$ does not occur in Equation 3.23. Additionally, Tournier et al. [TNGF15] claim that, in comparison to penalty forces where high stiffness makes the system matrix in (reference equation) almost singular, decreasing the compliance terms in the lower-right block of the system matrix in Equation 3.23 makes working with high stiffness numerically tractable. However, close inspection of the system matrix of Equation 3.23 shows that it quickly becomes singular for high stiffnesses as well when the lower-right block containing the corresponding compliances approaches zero.

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

## 3.4 Position Based Dynamics

As discussed in Section 3.1.1, classical approaches for dynamics simulation are force-based. Forces are accumulated and resulting accelerations are computed

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

The main drawback of PBD is that constraints become arbitrarily stiff when the iteration count is increased or when the 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 the equations of motion (Eq. (3.2)) with constraint potentials based on PBD constraints. The overall structure of the PBD algorithms is preserved with minor changes to the projection of individual constraints. XPBD reduces the coupling of stiffness to iteration count and time step and relates constraints to corresponding, well-defined constraint forces. According to Macklin et al., XPBD and PBD are equivalent in the limit of infinite stiffness.

Since PBD and XPBD only differ in the way individual constraints are projected, we give a general overview over PBD-style algorithms in Section 3.4.1. The details of individual constraint projection in PBD and XPBD are covered in Section 3.4.2 and Section 3.4.4, respectively.

### 3.4.1 Overview Over the PBD Framework

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

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

where $j$ is the constraint index. Note how constraints are defined solely in terms of particle positions. Equality and inequality constraints are satisfied if $C(\boldsymbol{q}) = 0$ and $C(\boldsymbol{q}) \geq 0$, respectively. In PBD, each constraint has an additional stiffness parameter $k_j \in [0, 1]$. Each constraint has a cardinality $n_j \in \mathbb{N}$ and particle

indices $i_1, \ldots, i_{n_j}$ of particles that actively contribute to the constraint value. In other words, for $l \in [1, \ldots, r]$ with $l \notin \{i_1, \ldots, i_{n_j}\}$ it is $\nabla_{\boldsymbol{p}_l} C_j(\boldsymbol{q}) = \mathbf{0}$.

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

For general, non-linear constraints, moving the initial estimates from the symplectic Euler integration to positions that satisfy the constraints requires solving a non-linear system of equations. Solving this system of equations is further complicated by the presence of inequality constraints, which need to be added or removed depending on whether they are satisfied during the current iteration. Thus, Müller et al. [MHHR06] opt for a non-linear adaptation of the Gauss-Seidel solver in their original PBD solver. Macklin et al. [MMC16] adapt this approach in XPBD. Just like the original Gauss-Seidel algorithm, which is only suitable for linear systems of equations, constraints are solved independently one after another. During each constraint solve, only the particles that contribute to the current constraint are moved while all the other particle positions remain untouched. Additionally, position updates from the projection of a constraint are immediately visible during the projection of the constraints following thereafter. Inequality constraints that are already satisfied are simply skipped. During each solver iteration, all constraints are cycled through once.

Due to the fact that PBD is a geometrical method that is not derived from Newton's laws of motion (Section 3.4.2) and that constraints are solved locally one after each other, Müller et al. [MHHR06] take great care that projections for internal constraints, i.e. constraints that are independent of rigid-body motion, preserve linear and angular momentum. Otherwise, internal constraints may introduce ghost forces which manifest themselves in artificial rigid-body motion [MHHR06]. Of course, non-internal constraints such as collision or attachment constraints may have global effects on an object. For internal constraints, it is easy to show that both momenta are automatically preserved if the PBD position updates are performed in the direction of the the mass-weighted constraint gradient [MHHR06]. Even though XPBD – unlike PBD – is in fact derived from Newton's second law, Macklin et al. [MMC16] arrive at position updates that are multiples

**Algorithm 7** Position Based Dynamics Framework

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

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

$$C(\boldsymbol{p} + \Delta\boldsymbol{p}) = 0. \tag{3.24}$$

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

$$\Delta\boldsymbol{p} = \lambda\mathbf{W}\nabla C(\boldsymbol{p}) \tag{3.25}$$

for some $\lambda \in \mathbb{R}$ and $\mathbf{W} = diag(w_1, w_1, w_1, \ldots, w_m, w_m, w_m)$. Plugging into $Equation\ 3.24$ and approximating by first-order Taylor expansion yields

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

Solving for $\lambda$ yields

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

Plugging $\lambda$ into Equation 3.25 results in the final position update

$$\Delta \boldsymbol{p} = -\frac{C(\boldsymbol{p})}{\sum_{i\in\{i_1,\dots,i_{n_c}\}} w_i |\nabla_{\boldsymbol{p}_i} C(\boldsymbol{p})|^2} \mathbf{W} \nabla C(\boldsymbol{p}). \tag{3.27}$$

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

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

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

$$C_{\text{dist}}(\boldsymbol{p}) = |\boldsymbol{p}_{i_1} - \boldsymbol{p}_{i_2}|. \tag{3.29}$$

Then, after $n_s$ solver iterations the remaning error is going to be $|\boldsymbol{p}_{i_1} - \boldsymbol{p}_{i_2}|(1 - k)^{n_s}$. Müller et al. suggest establishing a linear relationship by multiplying corrections by $k' = 1 - (1 - k)^{1/n_s}$. This way, the error becomes $|\boldsymbol{p}_{i_1} - \boldsymbol{p}_{i_2}|(1 - k)$ after $n_s$ solver iterations. A summary of the constraint solver is given in Algorithm 8.

### 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.

**Algorithm 8** PBD Constraint Solver

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

The position update due to a single constraint $C$ in Equation 3.27 is related to the Newton-Raphson method for finding roots of non-linear functions $f\colon \mathbb{R} \to \mathbb{R}$ [MHHR06]. There, the current guess $x_i \in \mathbb{R}$ for a root of $f$ is refined using the following update formula (Citation needed? Falls into the curriculum of a B.Sc. )

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_0)}. \tag{3.30}$$

Indeed, applying the Newton-Raphson update to

$$f\colon \mathbb{R} \to \mathbb{R}, \lambda \mapsto C(\boldsymbol{p} + \lambda \mathbf{W} \nabla C(\boldsymbol{p})) \tag{3.31}$$

yields

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

and with $\lambda_0 = 0$ we arrive at

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

Note that this is exactly the same as the $\lambda$ used in the constraint solver given in Equation 3.26. Thus, a single constraint projection corresponds to the first iteration of the Newton-Raphson method applied to Equation 3.31 with $\lambda_0 = 0$.

The correspondence breaks down if $\lambda_0 \neq 0$ or if multiple position updates are performed consecutively for the same constraint.

Müller et al. [MHHR06] claim that PBD is unconditionally stable since the projected positions $\boldsymbol{p}_i$ computed by the constraint solver are physically valid in the sense that all constraints are satisfied and no extrapolation into the future takes place in lines 7-8 of Algorithm 7. They further state that the only source of instabilities is the constraint solver itself, which is based on the Newton-Raphson method. The position updates in Equation 3.28 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. (3.29)). As discussed in Section 3.4.2, the remaining error after $n_s$ solver iterations is simply $|\boldsymbol{p}_{i_1} - \boldsymbol{p}_{i-2}|(1-k)^{n_s}$. In the limit of infinite iterations

$$\lim_{n_s \to \infty} (|\boldsymbol{p}_{i_1} - \boldsymbol{p}_{i-2}|(1-k)^{n_s}) = 0,$$

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

While the simulated object's global linear and angular momentum are preserved, the linear momentums of individual particles are at risk of being washed out by the PBD constraint solver [BML+14]. This is because even though the structure of the position updates preserves global momentum, there is no punishment for moving individual particles away from their inertial positions. Gen-

erally, the penalty for moving particles away from their inertial positions should increase with growing particle masses. However, in the PBD position update in Equation 3.28 it is only the ratio of the particle masses that matters. This can be seen by multiplying all inverse masses $w_i$ in Equation 3.28 with a constant factor $a \in \mathbb{R}^+$:

$$\begin{aligned}\Delta \boldsymbol{p}_i &= -\frac{C(\boldsymbol{p})}{\sum_{j\in\{i_1,\dots,i_{n_c}\}} aw_j|\nabla_{\boldsymbol{p}_j}C(\boldsymbol{p})|^2} aw_i\nabla_{\boldsymbol{p}_i}C(\boldsymbol{p}) \\ &= -\frac{C(\boldsymbol{p})}{\sum_{j\in\{i_1,\dots,i_{n_c}\}} \frac{aw_j}{aw_i}|\nabla_{\boldsymbol{p}_j}C(\boldsymbol{p})|^2}\nabla_{\boldsymbol{p}_i}C(\boldsymbol{p}) \\ &= -\frac{C(\boldsymbol{p})}{\sum_{j\in\{i_1,\dots,i_{n_c}\}} \frac{w_j}{w_i}|\nabla_{\boldsymbol{p}_j}C(\boldsymbol{p})|^2}\nabla_{\boldsymbol{p}_i}C(\boldsymbol{p})\end{aligned} \tag{3.32}$$

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

### 3.4.4 XPBD Constraint Projection

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

$$\mathbf{M}(\boldsymbol{q}_{n+1} - \boldsymbol{q}_n - h\boldsymbol{v}_n) = h^2(\boldsymbol{f}_{\text{ext}} - \sum_j \nabla\psi_j(\boldsymbol{q}_{n+1})). \tag{3.33}$$

Let $m$ be the number of particles in the simulated body and $r$ be the number of conservative potentials $\psi_j$ with $j \in [1, r]$. In the context of XPBD, $\boldsymbol{q}, \boldsymbol{v}, \boldsymbol{f}_{\text{ext}} \in \mathbb{R}^{3m}$ and $\psi_j \colon \mathbb{R}^{3m} \to \mathbb{R}$. Simple manipulation of Equation 3.33 yields

$$\mathbf{M}\,(\boldsymbol{q}_{n+1} - \tilde{\boldsymbol{q}}) = -h^2 \sum_j \nabla\psi_j(\boldsymbol{q}_{n+1}), \tag{3.34}$$

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

$$\psi_j(\boldsymbol{q}) = \frac{1}{2}\alpha_j^{-1} C_j(\boldsymbol{q})^2, \tag{3.35}$$

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

$$\psi(\boldsymbol{q}) :- \sum_j \psi_j(\boldsymbol{q}) = \frac{1}{2}\boldsymbol{C}(\boldsymbol{q})^T \alpha^{-1} \boldsymbol{C}(\boldsymbol{q}). \tag{3.36}$$

where $\psi$ is the combined internal energy potential. The force from the internal potential is given by

$$\boldsymbol{f}_{\text{int}} = -\nabla\psi(\boldsymbol{q}) = -\nabla\boldsymbol{C}(\boldsymbol{q})^T \alpha^{-1} \boldsymbol{C}(\boldsymbol{q}). \tag{3.37}$$

Plugging the internal force $\boldsymbol{f}_{\text{int}}$ into Equation 3.34 and pulling $h^2$ into the compliance matrix $\alpha$ results in

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

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

$$\boldsymbol{\lambda} = -\tilde{\alpha}^{-1} \boldsymbol{C}(\boldsymbol{q}). \tag{3.38}$$

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

$$\mathbf{M}(\boldsymbol{q}_{n+1} - \tilde{\boldsymbol{q}}) - \nabla(\boldsymbol{q}_{n+1})^T \boldsymbol{\lambda}_{n+1} = \mathbf{0} \tag{3.39}$$

$$\boldsymbol{C}(\boldsymbol{q}_{n+1}) + \tilde{\alpha}\boldsymbol{\lambda}_{n+1} = \mathbf{0}. \tag{3.40}$$

The left-hand side of Equation 3.39 and Equation 3.40 are referred to as $\boldsymbol{g}$ and $\boldsymbol{h}$, respectively. The non-linear system of equations is solved using a fixed-point iteration based on Newton's method. We replace the index $(n+1)$ indicating the current time step by the index of the current guess in the fixed-point iteration indicated by $(i+1)$ for the sake of clarity. During each iteration, guesses $\boldsymbol{q}_i, \boldsymbol{\lambda}_i$ for a solution of the non-linear system are improved by updates $\Delta\boldsymbol{q}, \Delta\boldsymbol{\lambda}$ to yield new iterates $\boldsymbol{q}_{i+1}, \boldsymbol{\lambda}_{i+1}$. The updates are determined by solving the following linear system of equations, which arises from the linearization of Equation 3.39 and Equation 3.40:

$$\begin{pmatrix} \mathbf{K} & -\nabla \boldsymbol{C}^T(\boldsymbol{q}_i) \\ \nabla \boldsymbol{C}(\boldsymbol{q}_i) & \tilde{\alpha} \end{pmatrix} \begin{pmatrix} \Delta \boldsymbol{q} \\ \Delta \boldsymbol{\lambda} \end{pmatrix} = - \begin{pmatrix} \boldsymbol{g}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) \\ \boldsymbol{h}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) \end{pmatrix}, \tag{3.41}$$

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

$$\mathbf{K} = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{q}}(\boldsymbol{q}_i) = \mathbf{M} - \frac{\partial \nabla \boldsymbol{C}(\boldsymbol{q}_i)^T \boldsymbol{\lambda}_i}{\partial \boldsymbol{q}}. \tag{3.42}$$

Note how the second term corresponds to the geometric stiffness (see [TNGF15], reference appropriate section). We refer to the system matrix of Equation 3.41 as $\mathbf{H}$. At this point, two simplifying assumptions are made.

**Assumption 1:** Computing the geometric stiffness in $\mathbf{K}$ requires evaluating second derivatives of the constraint functions $C_j$. This is expensive and error-prone. In order to avoid the challenges of computing second derivatives and to re-establish a connection to PBD (Section 3.4.2), Macklin et al. [MMC16] drop the geometric stiffness by approximating

$$\mathbf{K} \approx \mathbf{M}. \tag{3.43}$$

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

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

$$\boldsymbol{g}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) = \mathbf{0}. \tag{3.44}$$

If initial guesses $\boldsymbol{q}_0 = \tilde{\boldsymbol{q}}$ and $\boldsymbol{\lambda}_0 = \mathbf{0}$ are used, plugging into Equation 3.39 shows that this assumption is trivially satisfied during the first iteration. To understand the justification for further iterations, it is helpful to take a look at the simplified version of Equation 3.41 with both assumptions in place:

$$\begin{pmatrix} \mathbf{M} & -\nabla \boldsymbol{C}^T(\boldsymbol{q}_i) \\ \nabla \boldsymbol{C}(\boldsymbol{q}_i) & \tilde{\alpha} \end{pmatrix} \begin{pmatrix} \Delta \boldsymbol{p} \\ \Delta \boldsymbol{\lambda} \end{pmatrix} = - \begin{pmatrix} \mathbf{0} \\ \boldsymbol{h}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) \end{pmatrix}. \tag{3.45}$$

We refer to the system matrix as $\mathbf{H}_{\text{simp}}$. After the first iteration, the upper row of Equation 3.45 is satisfied. Thus, after the first iteration with $\boldsymbol{q}_0 = \tilde{\boldsymbol{q}}$ and $\boldsymbol{\lambda}_0 = \mathbf{0}$, it is

$$\begin{aligned} \mathbf{0} = \mathbf{M}\Delta \boldsymbol{q} - \nabla \boldsymbol{C}(\boldsymbol{q}_0)^T \Delta \boldsymbol{\lambda} &= \mathbf{M}(\boldsymbol{q}_1 - \boldsymbol{q}_0) - \nabla \boldsymbol{C}(\boldsymbol{q}_0)^T (\boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_0) \\ &= \mathbf{M}(\boldsymbol{q}_1 - \tilde{\boldsymbol{q}}) - \nabla \boldsymbol{C}(\boldsymbol{q}_0)^T \boldsymbol{\lambda}_1. \end{aligned} \tag{3.46}$$

Note how the last term is almost identical to $\boldsymbol{g}(\boldsymbol{q}_1, \boldsymbol{\lambda}_1) = \mathbf{M}(\boldsymbol{q}_1 - \tilde{\boldsymbol{q}}) - \nabla \boldsymbol{C}(\boldsymbol{q}_1)^T \boldsymbol{\lambda}_1$, 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 $\mathbf{M}$ in the upper-left block of $\mathbf{H}_{\text{simp}}$ is invertible by design, it is possible to take the Schur complement with respect to $\mathbf{M}$ to obtain a reduced system in terms of $\Delta\boldsymbol{\lambda}$:

$$(\nabla \boldsymbol{C}(\boldsymbol{q}_i) \mathbf{M}^{-1} \nabla \boldsymbol{C}(\boldsymbol{q}_i)^T + \tilde{\alpha}) \Delta\boldsymbol{\lambda} = -\boldsymbol{C}(\boldsymbol{q}_i) - \tilde{\alpha} \boldsymbol{\lambda}_i \tag{3.47}$$

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

$$\Delta\boldsymbol{q} = \mathbf{M}^{-1} \nabla \boldsymbol{C}(\boldsymbol{q}_i)^T \Delta\boldsymbol{\lambda}. \tag{3.48}$$

Up until here, all constraints were handled together during each iteration. In order to make a connection to PBD and to return to the framework of a non-linear Gauss-Seidel solver Section 3.4.1, it is necessary to specify how to solve a single constraint. To that end we rewrite Equation 3.47 for a single constraint $C_j$ and get the update for its scalar Lagrange multiplier $\lambda_j$ by computing

$$\Delta\lambda_j = \frac{-C_j(\boldsymbol{q}_i) - \tilde{\alpha}_j \lambda_{ji}}{\nabla C_j(\boldsymbol{q}_i) \mathbf{M}^{-1} \nabla C_j(\boldsymbol{q}_i)^T + \tilde{\alpha}_j}. \tag{3.49}$$

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

$$\Delta\boldsymbol{q}_l = w_l \nabla_{\boldsymbol{q}_l} C_j(\boldsymbol{q}_i)^T \Delta\lambda_j. \tag{3.50}$$

Note that $\Delta\lambda_j$ is scalar. Thus, the position update is a multiple of the mass-weighted gradient, just like in PBD (Eq. (3.28)).

In summary, we simply compute $\Delta\lambda_j$ via Equation 3.49 and use it to update $\lambda_{j+1} = \lambda_j + \Delta\lambda$ and to determine $\Delta\boldsymbol{q}$ via Equation 3.48 while solving the $j$-th constraint. This leads to a natural extension of the PBD algorithm, where the general structure in Algorithm 7 is preserved. The only changes occur in the computation of the scaling factor for the mass-weighted constraint gradient in PROJECTCONSTRAINTS in line 5. The XPBD version of PROJECTCONSTRAINTS is given in Algorithm 9. Note that Algorithm 9 is specified in terms of the projection points $\boldsymbol{p}$ or $\boldsymbol{p}_i$ instead of the positions $\boldsymbol{q}$ or $\boldsymbol{q}_i$ used in Equation 3.48 and Equation 3.49 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 $\lambda_j$ **do**
      **if** $C_j$ is an inequality constraint and $C_j(\boldsymbol{p}) \geq 0$ **then**
        **continue** to next constraint
      **end if**
      $\Delta\lambda_j = \frac{-C_j(\boldsymbol{p}) - \tilde{\alpha}_j \lambda_j}{\nabla C_j(\boldsymbol{p}) \mathbf{M}^{-1} \nabla C_j(\boldsymbol{p})^T + \tilde{\alpha}_j}$
      $\lambda_j = \lambda_j + \Delta\lambda_j$
      **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 $\alpha = \mathbf{0}$. This requires removing all occurrences of stiffnesses in favor of compliances. In XPBD, this is achieved by pulling large stiffness values out of the constraints in Equation 3.35 and introducing artificial Lagrange multipliers (Eq. (3.38)) that hide large stiffness values and treating them as unknowns in Equation 3.39 and Equation 3.40 . Indeed, a closer look at the PBD update formula Equation 3.28 and the XPBD update formulas in Equation 3.49, Equation 3.50 reveals that XPBD and PBD are equivalent if the compliance term $\tilde{\alpha}_j$ of constraint $C_j$ is zero. This matches with the observation that bodies simulated by PBD are infinitely stiff in the limit of infinite iterations (Section 3.4.3). If $\tilde{\alpha}_j \neq 0$, the compliance terms in Equation 3.49 regularize the constraint in such

a way that the constraint force is limited and corresponds to the constraint potential [MMC16]. This addresses the issue of coupling between iteration count and stiffness in the original PBD algorithm (Section 3.4.3). Since the time step is also baked into $\tilde{\alpha}_j$, coupling between time step size and stiffness is also reduced.

The discussion above highlights the importance of pulling large stiffness values out of the constraints in Equation 3.36. However, in many cases it can be challenging to pull stiffness factors out of the constraints. In such cases, it is interesting to ask whether XPBD – despite the iterative nature of the Gauss-Seidel solver and the assumptions made during its derivation (Eq. (3.43), Eq. (3.44)) – behaves the same for both formulations if only large, but finite, stiffnesses are used. To this end, the constraint potentials in Equation 3.36 can be rewritten in terms of the alternative constraint function $\boldsymbol{C}'$ given by

$$\boldsymbol{C}'(\boldsymbol{q}) = \alpha^{-1/2}\boldsymbol{C}(\boldsymbol{q}). \tag{3.51}$$

leading to the equivalent constraint potentials

$$\psi(\boldsymbol{q}) = \frac{1}{2}\boldsymbol{C}'(\boldsymbol{q})^T\mathbf{I}\boldsymbol{C}'(\boldsymbol{q}) = \frac{1}{2}\boldsymbol{C}'(\boldsymbol{q})^T\alpha'^{-1}\boldsymbol{C}'(\boldsymbol{q}). \tag{3.52}$$

Here, $\alpha'^{-1} = \mathbf{I}$ without relation to the constraint stiffnesses $k_j$. Thus, using zero compliance in order to model infinite stiffness becomes impossible. If $\boldsymbol{C}'$ and $\alpha'$ are used, the update formulas for individual Lagrange multipliers (Eq. (3.49)) and individual positions (Eq. (3.50)) become

$$\begin{aligned}
\Delta\lambda_j &= \frac{-C'_j(\boldsymbol{q}_i) - \frac{1}{h^2}\lambda_{ij}}{\nabla C'_j(\boldsymbol{q}_i)\mathbf{M}^{-1}\nabla C'_j(\boldsymbol{q}_i)^T + \frac{1}{h^2}} \\
&= \frac{-\sqrt{k_j}C_j(\boldsymbol{q}_i) - k_j\tilde{\alpha}_j\lambda_{ij}}{k_j\nabla C_j(\boldsymbol{q}_i)\mathbf{M}^{-1}\nabla C_j(\boldsymbol{q}_i)^T + k_j\tilde{\alpha}_j} \\
&= \frac{-\frac{1}{\sqrt{k_j}}C_j(\boldsymbol{q}_i) - \tilde{\alpha}_j\lambda_{ij}}{\nabla C_j(\boldsymbol{q}_i)\mathbf{M}^{-1}\nabla C_j(\boldsymbol{q}_i)^T + \tilde{\alpha}_j}.
\end{aligned} \tag{3.53}$$

and

$$\begin{aligned}
\Delta\boldsymbol{q}_l &= \frac{-\frac{1}{\sqrt{k_j}}C_j(\boldsymbol{q}_i) - \tilde{\alpha}_j\lambda_{ij}}{\nabla C_j(\boldsymbol{q}_i)\mathbf{M}^{-1}\nabla C_j(\boldsymbol{q}_i)^T + \tilde{\alpha}_j} w_l\nabla C'_j(\boldsymbol{q}_i)^T \\
&= \frac{-\frac{1}{\sqrt{k_j}}C_j(\boldsymbol{q}_i) - \tilde{\alpha}_j\lambda_{ij}}{\nabla C_j(\boldsymbol{q}_i)\mathbf{M}^{-1}\nabla C_j(\boldsymbol{q}_i)^T + \tilde{\alpha}_j} w_l\sqrt{k_j}\nabla C_j(\boldsymbol{q}_i)^T.
\end{aligned} \tag{3.54}$$

By looking at the last lines of Equation 3.53 and Equation 3.54, it is easy to see that using $C'_j$ and $\alpha'_j$ with $\boldsymbol{\lambda}_0 = 0$ yields the same positions, but decreases the Lagrange multipliers by a factor of $1/\sqrt{k_j}$. Note that numerical differences between both formulations might still arise (No idea how to reason about this. Looks this up or ask Fabian).

As mentioned in Section 3.4.4, assumption 1 (Eq. (3.43)) corresponds to dropping the geometric stiffness $\delta\nabla \boldsymbol{C}(\boldsymbol{q_i})^T \boldsymbol{\lambda_i}/\delta \boldsymbol{q}$. It is worth noting that with $\boldsymbol{\lambda}_0 = \vec{0}$, this assumption is trivially satisfied during the first iteration. According to Equation 3.38, after a sufficient number of iterations it is $\boldsymbol{\lambda}_i \approx \tilde{\alpha}\boldsymbol{C}(\boldsymbol{q}_i)$, meaning that the entries of the geometric stiffness grow with increasing constraint stiffness and iteration count. Thus, with stiffer constraints and larger iteration counts, assumption 1 becomes more aggressive. Tournier et al. [TNGF15] state that the iterative nature of PBD-style solvers ameliorates instabilities in the transverse direction of stiff constraints that is often observed as a result of neglecting geometric stiffness.

Macklin et al. [MMC16] claim that replacing $\mathbf{H}$ with $\mathbf{H}_{\text{simp}}$ in assumption 1 can be interpreted as applying a quasi-Newton method – also known as Broyden methods – to the NLSE given by Equation 3.39 and Equation 3.40, only affecting the convergence rate. While it is true that changing the Hessian matrix in Newton's method for unconstrained optimization to some positive definite approximation only changes the convergence rate under mild conditions [NW06], it is difficult to verify whether this also holds for the simplification from $\mathbf{H}$ to $\mathbf{H}_{\text{simp}}$ in the context of solving NLSEs. There, it is often the case that quasi-Newton methods are not guaranteed to converge at all if aggressive approximations of the system matrix are performed [NW06]. This is because there is no natural merit function available to help with the selection of step sizes along the suggested update directions. In their XPBD derivation, Macklin et al. [MMC16] do point out that a line search strategy might be required to keep the fixed-point iteration based on Equation 3.41 robust, but it is also important to mention that approximations of the system matrix without an appropriate line search procedure in place are potentially more aggressive.

Assumption 2 in the XPBD derivation is justified by the observation that Equation 3.46 and $\boldsymbol{g}(\boldsymbol{q}_1, \boldsymbol{\lambda}_1)$ are the same, except that $\nabla \boldsymbol{C}(\boldsymbol{q}_0)^T$ is replaced by $\nabla \boldsymbol{C}(\boldsymbol{q}_1)^T$. Thus, Macklin et al. [MMC16] claim that Equation 3.46 is close to zero as well if $\boldsymbol{g}(\boldsymbol{q}_0, \boldsymbol{\lambda}_0) = \mathbf{0}$, which is true by definition of $\boldsymbol{q}_0$ and $\boldsymbol{\lambda}_0$. However, this neglects the fact that $\nabla \boldsymbol{C}(\boldsymbol{q}_i)$ occurs in a product with $\boldsymbol{\lambda}_i$ in $\boldsymbol{g}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i)$. As mentioned in the discussion of assumption 1, $\boldsymbol{\lambda}_i$ gets very large for stiff constraints after a sufficient number of iterations. Thus, even small changes to $\nabla \boldsymbol{C}(\boldsymbol{q}_i)$ eventually drive the value of $\boldsymbol{g}$ away from zero significantly. As a result, assumption 2 becomes more aggressive with stiffer constraints and larger iteration counts, just like observed for assumption 1. Additionally, assuming that $\boldsymbol{g}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) = \mathbf{0}$ leads to

a change of the right side between the LSEs in Equation 3.41 and Equation 3.45. In contrast to the changes to the system matrix during assumption 1, this does affect the solution the solver converges to. Thus, due to assumption 2, the XPBD solver cannot be said to solve the original implicit equations of motion [MMC16].

When investigating the properties of PBD (Section 3.4.3), we mentioned that multiplying all particle masses $m_i$ with a constant positive factor $a \in \mathbb{R}^+$ does not impact the PBD update, highlighting that there is no punishment for moving individual particles from their inertial positions in PBD. The fact that both positions and Lagrange multipliers are updated during each iteration of the XPBD solver makes an analysis similar to Equation 3.32 for XPBD challenging for arbitrary iterations $i$. However, it is simple to look at the effect of scaling all particle masses by $a$ on the position update during the first iteration, assuming that $\boldsymbol{\lambda} = \mathbf{0}$. The resulting update of particle $i$ is given by

$$\Delta \boldsymbol{q}_i = -\frac{C(\boldsymbol{q})}{\sum_{j \in \{i_1,\dots,i_{n_c}\}} \frac{w_j}{w_i} |\nabla_{\boldsymbol{q}_j} C(\boldsymbol{q})|^2 + \tilde{\alpha} a m_i} \nabla_{\boldsymbol{q}_i} C(\boldsymbol{q}). \tag{3.55}$$

Note how the factor $a$ does occur in the second summand in the denominator in a way that decreases the size of the position update if $a$ is increased. However, it appears in a product with the small $\tilde{\alpha}$. Thus, the effect of increasing all particle masses on the position update is still rather small. It is worth pointing out that the inclusion of $a$ in the position update of the first iteration penalizes moving particles with large masses in all directions. Ideally, moving particles towards the inertial positions $\tilde{\boldsymbol{q}}$ should be encouraged whereas as moving particles away from $\tilde{\boldsymbol{q}}$ should be discouraged. This directionality is only possible if $\tilde{\boldsymbol{q}}$ is not simply used as an initial guess for the XPBD solver, but also used explicitly in the update equations. However, setting $\boldsymbol{g}(\boldsymbol{q}_i, \boldsymbol{\lambda}_i) = \mathbf{0}$ in assumption 2 removes all occurrences of the inertial masses from Equation 3.45.

Both assumptions 1 and 2 benefit from the choice $\boldsymbol{\lambda}_0 = \mathbf{0}$. It is worth mentioning that this precludes the use of the more natural initial guess given by $\boldsymbol{\lambda}_0 = -\tilde{\alpha}\boldsymbol{C}(\tilde{\boldsymbol{q}})$. This candidate results from plugging the inertial positions into the definition of the Lagrange multipliers in Equation 3.38. Picking an initial guess that is as close as possible to the true solution is important for keeping the linearization error during the fixed-point iteration based on Newton's method in the XPBD derivation small.

While motion due to external forces is handled by the symplectic Euler integration in lines 3-4 of Algorithm 7, the way elasticity is handled is derived from the implicit equations of motions (Eq. (3.33)). As a result elastic forces are subjected to numerical damping that gets more severe with growing time step sizes in XPBD (Section 3.1.2). For this reason, even though baking the time step into $\tilde{\alpha}$ in the compliant constraint formulation reduces coupling between time step and

stiffness, the perceived stiffness of simulated materials is still time step dependent to some extent.

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

## 3.5 Projective Dynamics

In the approaches to physical simulations via implicit time integration that we have encountered so far, a new linear system needs to be solved at every 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), Bouaziz et al. [BML+14] instead restrict energy potentials to a specific structure which allows for efficient implicit time integration via alternating steps of local and global optimization [BML+14]. The local optimization steps are comprised of per-constraint projections of particle positions onto constraint manifolds. The global optimization step combines the results from the individual local projection steps while taking into consideration global effects including inertia and external forces. This is achieved by solving a linear system of equations whose system matrix is constant across 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 $\boldsymbol{f} \coloneqq \mathbf{F}(\mathbf{q}) \in \mathbb{R}^{3\times 3}$. Then, energy potentials of the general form $\psi(\mathbf{E}(\boldsymbol{f}))$, where $\mathbf{E}(\boldsymbol{f})$ is a strain measure that depends on the deformation gradient of a discrete element, are frequently used in nonlinear continuum mechanics [SB12]. If $\mathbf{E}$ is Green's strain measure $\mathbf{E}_{\text{Green}}$ defined by

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

then $\mathbf{E}_{\text{Green}}(\boldsymbol{f}) = \mathbf{0}$ is equivalent to $\mathbf{F}^{\mathbf{T}}\mathbf{F} = \mathbf{I}$. Thus, $\mathbf{E}_{\text{Green}}(\boldsymbol{f}) = 0$ defines a constraint manifold of configuarion 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 $\psi(\mathbf{0}) = \rho$ for some $\rho \in \mathbb{R}$ and that $\psi$ reaches its minimum at the undeformed configuration, then

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

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 $\mathbf{E}_{\text{Green}}(\mathbf{f}) = 0$ – and a distance measure $d$ of the body's current configuration to that constraint manifold. Formally, this leads to energy potentials 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 (Fabian points out that it is quite surprising that $p$ and $q$ do not have same dimensions, i.e. that there is no particle to particle correspondence. This is because I use the notationa from [LBK17] to stay consistent. It is better to use the original notation from [BML+14] when introducing PD for the sake of clarity. Later write a paragraph to show that both formulations are equivalent and use the one from [LBK17]) 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 $\psi(\boldsymbol{q}) = d(\boldsymbol{q}, \boldsymbol{p_q}) + \delta_{\boldsymbol{C}}(\boldsymbol{p_q})$ (Fabian prefers argmin notation here.). 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

$$d(\boldsymbol{q}) = \frac{w}{2}\|\boldsymbol{G}\boldsymbol{q} - \boldsymbol{p}\|_F^2, \tag{3.56}$$

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

$$\psi(\boldsymbol{q}) = \min_{\boldsymbol{p}} \frac{w}{2} \|\boldsymbol{g}\boldsymbol{q} - \boldsymbol{p}\|_F^2 + \delta_{\boldsymbol{C}}(\boldsymbol{p}). \tag{3.57}$$

**Example: Strain Energies**

As an elucidating example, we briefly recap how to formulate strain energies in terms of PD energy potentials (Equation 3.57) according to Bouaziz et al. [BML+14]. Strain energies measure the change of local variation between the deformed and undeformed state. If the simulated body consists of linear tetrahedral elements, the continuous strain energy can be discretized across the tetrahedra according to (refer to appropriate section). The energy for a single tetrahedradron can be approximated in terms of PD energy potentials (Eq. (3.57)) by setting $\mathbf{G} \in \mathbb{R}^{3\times m}$ to the matrix $\boldsymbol{a}$ 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) (Don't forget to change this for the [BML+14] notation. In this setting, p is the set of particle positions for which the deformation gradient is a rotation.). 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:

$$\psi(\boldsymbol{q}) = \min_{\boldsymbol{p}} \frac{w}{2} \|\mathbf{A}\boldsymbol{q} - \boldsymbol{p}\|_F^2 + \delta_{\mathrm{SO}(3)}(\boldsymbol{p}) = \min_{\boldsymbol{p}} \frac{w}{2} \left\| \boldsymbol{f}^T - \boldsymbol{p} \right\|_F^2 + \delta_{\mathrm{SO}(3)}(\boldsymbol{p}). \tag{3.58}$$

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 $\boldsymbol{p} = \mathbf{U}\mathbf{I}\mathbf{V}^T$ where $\boldsymbol{f}^T = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ 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. (3.57)) into the variatonal form of implicit Euler integration (Eq. (3.7)). 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 $\boldsymbol{p}_j$

$$\min_{\boldsymbol{q},\boldsymbol{p}_j} \tilde{g}(\boldsymbol{q}, \boldsymbol{p}_j) = \min_{\boldsymbol{q},\boldsymbol{p}_j} \frac{1}{2h^2}$$

# 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 (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 (cited on pages 20, 21, 23).

[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 (cited on pages 5, 7, 8, 29, 31, 40–42).

[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).

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

[LBK17] Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. “Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials”. In: *ACM Trans. Graph.* 36.4 (July 2017). ISSN: 0730-0301 (cited on pages 17, 41).

[LLF+23] Andreas Longva, Fabian Löschner, José Antonio Fernández-Fernández, Egor Larionov, Uri M. Ascher, and Jan Bender. *Pitfalls of Projection: A study of Newton-type solvers for incremental potentials*. 2023 (cited on page 12).

[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 (cited on pages 5, 26, 27, 32, 34, 35, 37–39).

[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 (cited on pages 26–31).

[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 (cited on page 23).

[NMK+06] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. "Physically Based Deformable Models in Computer Graphics". In: *Computer Graphics Forum* 25.4 (2006), pp. 809–836 (cited on page 21).

[NW06] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. 2e. New York, NY, USA: Springer, 2006 (cited on pages 8, 9, 38).

[RU57] Hanan Rubin and Peter Ungar. "Motion under a strong constraining force". In: *Communications on Pure and Applied Mathematics* 10.1 (1957), pp. 65–87 (cited on page 21).

[SLM06] Martin Servin, Claude Lacoursière, and Niklas Melin. "Interactive Simulation of Elastic Deformable Materials". In: *Proc. SIGRAD* (Jan. 2006) (cited on pages 7, 20, 21, 23, 24, 36).

[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 (cited on page 40).

[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 (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 (cited on page 21).

[TNGF15] Maxime Tournier, Matthieu Nesme, Benjamin Gilles, and François Faure. “Stable Constrained Dynamics”. In: *ACM Trans. Graph.* 34.4 (July 2015). ISSN: 0730-0301 (cited on pages 19, 22, 23, 25, 34, 38).

[UR95] Linda R. Petzold Uri M. Ascher Hongsheng Chin and Sebastian Reich. “Stabilization of Constrained Mechanical Systems with DAEs and Invariant Manifolds”. In: *Mechanics of Structures and Machines* 23.2 (1995), pp. 135–157 (cited on pages 21–23).

# Index