Interactive simulation of anisotropic fracturing with position based composite materials

Jan Scheffczyk

BACHELORARBEIT

eingereicht am 06.08.2018

Fachhochschul-Bachelorstudiengang

Informatik

in Bingen

im August 2018

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Bingen, August 6, 2018

Jan Scheffczyk

Contents

Abstract

In this bachelor thesis we present a novel approach for simulating transverse-isotropic materials. In contrast to classical approaches that integrate the anisotropy into the strain-stress-constitutive model the presented approach is inspired by the natural structure of timber. A rod simulation that represents the fibers in timber is combined with a solid simulation that represents the cementing material to create an anisotropic composite material. For this purpose numerous approaches to calculate the dynamic response of rods and solids as well as their common theoretical background, continuum mechanics, are reviewed. By utilizing mesh-less solid simulation and scattered rods to discretize the object, our approach is not limited to a specific shape. Further by specifying the direction of anisotropy for each particle in the object results in a particular rod pattern. Uniform anisotropic directions create fibrous material like wood while random directions create locally anisotropic materials like press wood. Modulating the randomness allows to simulate a wide variety of materials. In addition, specifying the direction of anisotropy is more intuitive than parametrization of the stress-strain-constitutive model. The approach extends the recently popularized position based dynamics (PBD) allowing interactive frame rates at the cost of poor convergence making it difficult to achieve physically meaningful material parameters.

Kurzfassung

In der vorliegenden Bachelorarbeit wird ein neuartiger Ansatz zur Simulation transversal isotroper Materialien vorgestellt. Im Gegensatz zu klassischen Ansätzen, die die Anisotropie mit dem Deformations-Spannungs-Modell beschreiben, ist der vorgestellte Ansatz von der natürlichen Struktur des Holzes motiviert. Eine Stab-Simulation, welche die Fasern im Holz darstellt, wird mit einer Festkörper-Simulation kombiniert, die das hölzerne Verbindungsmaterial repräsentiert, um ein anisotropes Kompositmaterial zu bilden. Zu diesem Zweck werden zahlreiche Ansätze zur Berechnung der Dynamik von Stäben und Festkörpern sowie deren gemeinsamer theoretischer Hintergrund, die Kontinuumsmechanik, betrachtet. Die verwendete Festkörper-Simulation benötigt keine Verbindungen zwischen den Partikeln, außerdem werden kurze Stäbe abhängig von der spezifizierten Richtung im Objekt plaziert. Dies erlaubt eine Diskretisierung des Objekts, die nicht auf eine bestimmte Form beschränkt ist. Durch die Definition der Richtung der Anisotropie wird eine bestimmte Verteilung der Stäbe im Objekt erzeugt. Homogene anisotrope Richtungen erzeugen faserige Materialien wie Holz, während inhomogene Richtungen Materialien wie Pressholz erzeugen. Dadurch wird die Simulation von einer Vielzahl von Materialien möglich. Ein weiterer Vorteil ist, dass die Angabe der Richtung der Anisotropie intuitiver anwendbar ist als die Parametrisierung bei Verwendung des Spannungs-Dehnungs-Modells. Der Ansatz basiert auf der kürzlich vorgestellten Methode „position based dynamics“(PBD). Bei dieser können interaktive Bildraten erreicht werden, jedoch auf Kosten der Konvergenz und somit auch auf Kosten der physikalischen Materialparameter.

Index of Notation

p	Particle
$\mathbf{x}$	Position
$\mathbf{v}$	Velocity
$\mathbf{a}$	Acceleration
$\tilde{\mathbf{x}}$	Rest/Material Position
m	Mass
C	Constraints
n	Time-step variable
ρ	Density
$\mathbf{F}$	Deformation gradient with respect to $\tilde{\mathbf{x}}$
ε	Strain
ν	Strainrate
$\boldsymbol{\sigma}$	Stress
$\mathbf{W}$	Energy Potentials
$\boldsymbol{f}$	Forces
$\boldsymbol{\Omega}$	Darbaux Vector

Time-steps and Exponents: In context of simulation sometimes one needs to differentiate between information of the current time step n and next or previous one. Here we will overload the exponent to reference the timestep as well as the usual exponent. If the timestep variable n is used then the exponents references a position in time for example $\mathbf{x}_i^{\mathrm{n}+1}$ would reference the position of the i-th particle at the next time step. The rest-configuration is sometimes denoted as $\mathbf{x}^0$ however $n = 0$ does not necessarily define the rest-configuration. Instead parameters in rest configuration are denoted by $\tilde{\cdot}$, e.g. $\tilde{\mathbf{x}}$.

Domain and indices: Scalar values are denoted by lower-case italic letters, x, y, z, whereas vector are denoted by bold lower-case letters $\mathbf{x}, \tilde{\mathbf{x}}$ and second order or higher tensors by bold and upper-case letters $\mathbf{F}$. Unless stated otherwise indices on scalars denote correspondence to a particle, e.g.

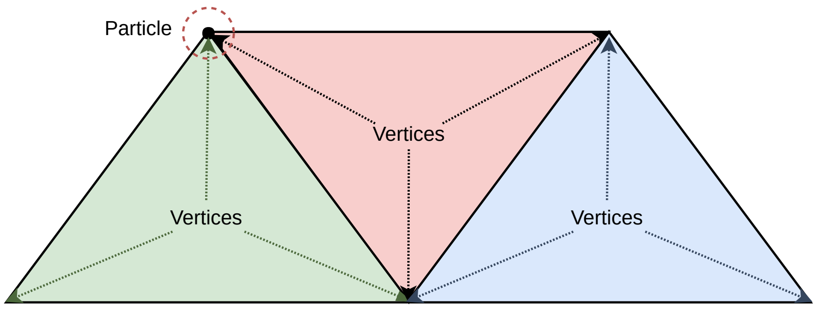


Figure 1: Each triangular elements has three vertices, each vertex is associated with a particle.

x_i might be the the x component of the i-th particle position. For vectors indices can denote either a particle correspondence or reference to their component x, y, z scalar component. The difference should be apparent from the the context as the component index is bound by the dimension of the vector where as the particle correspondence is bound by the, possibly infinite, number of particles. Finally single indices on matrices denote particle associations where as double indices denote a specific component of the matrix.

Discrete Entitles: Particles or points are abstract entities that associate a positions within the region of an object with other properties of the object like mass, temperature or even volumetric properties like pressure. In theory every object contains infinitely many particles or a continuum of particles however in practice we can only work with a finite amount. To approximate the continuum the object is decomposed into a set of simple shape[1] called elements. Each elements has a set of vertices, fig. 1, which can store element specific properties and but still reference to the corresponding particle. This way particle specific properties only need to be stored once and we are still able to store vertex-specific properties. A set of connected elements is called a mesh. Further we will use the same variables to denote continuous and discrete properties rather than introducing several variable for the same property in the discrete and continuous setting. As such $\mathbf{x}$ can either denote the continuous set of positions or a vector of a finite set of discrete positions $[\mathbf{x}_1^T, \ldots, \mathbf{x}_N^T]^T$.

[1]In one dimension possible elements are linear-lines, NURB-curvers or bezier-curver. For two dimensions triangle, quads or regular polygons are common and in three-dimensions their higher-order counterparts tetrahedral, cubes or platonic solids.

Eigenvalues and Eigenvectors: The i-th eigenvalue of a matrix $\boldsymbol{A}$ is denoted by $\lambda_i(\boldsymbol{A})$. If there is no index then $\lambda(\boldsymbol{A})$ denotes a diagonal matrix B with $B_{ii} = \lambda_i$ and 0 otherwise. The same notation is adopted for eigenvectors here $\varrho_i(\boldsymbol{A})$ denotes the i-th eigenvector of $\boldsymbol{A}$ and $\varrho(\boldsymbol{A})$ a matrix where the i-th column is the eigenvector $\varrho_i(\boldsymbol{A})$. It follows that the eigendecomposition of a matrix can then be written as $\varrho(\boldsymbol{A})\lambda(\boldsymbol{A})\varrho(\boldsymbol{A})^{-1}$ or if the context allows it simply as $\boldsymbol{\varrho\lambda\varrho^{-1}}$.

Chapter 1

Introduction

Mechanical simulations are essential to many industries as it allows to evaluate a products physical behavior, when subjected to load. The aerodynamics of an airplane-wing under extreme weather conditions, the stresses of heavy-duty trucks on a bridge or even the interaction of veins with surgical equipment need to be tested. Computer aided simulation allows to perform these tests digitally and thus, safely where conventional methods might fail due to moral, financial and physical constraints. Ultimately the ability to compute the behavior of complex mechanical systems is a corner stone of the rate of innovation and technological advances we had, and continue to experience. In order to be of any use in the engineering context the deviation of the simulation from an actual response must be constraint by a small and well understood numerical error. Much effort has been spent to minimize this error by sacrificing run-time-performance.

The visualization and simulation within the entertainment industry has almost the opposite requirements. Here visual plausibility is sufficiently accurate and instead, applications are optimized for run-time efficiency and ease of use. Run-time efficiency reduces the amount of hardware, power and, of course, time needed to complete the simulation. This also enables faster design iterations thus, reducing labor cost. To make the system usable for non-domain experts such as visual effects artists the parameter complexity that comes with a true physical model needs to be reduced to a manageable, ideally intuitive set of parameters. Another key distinction from other industries is the need for a unified simulation framework, i.e. a single framework that can simulate rods, sheets, solids, fluids and the interactions between them. While it is feasible or even reasonable for a cosmetic manufacturer to employ a simulation framework that only models hair(rods) and a separate framework to only simulate skin(sheets), it is not for content creation as the interaction between any kind of material is frequently needed. These alternative criteria are distinct enough to lead to a large body of novel methods yet similar enough that advancement in either field can potentially contribute to the other. Interactive-simulation is due to its real-time constraint a little further removed from the engineering methods than the offline-simulations used in feature-films.

It is apparent that all industries would benefit from the time and cost savings of better run-time efficiency. As such some of the advancements and best practices that emerged from this alternative set of criterias have been implemented into major production pipelines. For example Michels (2014) and Michels et al. (2014, 2015) proposed stiff integrators and partial analytic solutions for the elastic Kirchhoff-model which was adapted by P&G[1] for their hair-simulation. These new methods reduced simulation time from more than a week down to a few minutes. In computational fluid dynamics (CFD) software packages such as Ansys, now provide a GPU-acceleration that allows the technical designer to interactively get a rough approximation of their desired quantities before starting the more accurate and costly simulations. In this thesis we will focus on a specific aspect of interactive simulation that has gotten little coverage so far: the fracturing of anisotrophic materials.

1.1 Anisotopic Fracture

The elastic response of isotropic materials is independent of the direction external forces (or loads) are applied. For anisotropic material the direction of the force matters. Perhaps the most obvious example of such a material is wood. It can easily be split by an axe if chucked along the grain, however, it takes significantly more effort (force) to split it orthogonal to the fibre direction. Further wood makes it very clear as to why it has these anisotropic properties. The fiber structure within wood can be seen just by looking at it. As discribed in Dinwoodie (2000) at microscopic level one can see that wood is mostly made up of either 2 or 4 cell types for soft-and hardwood respectively. The fibre cells are the main supporting structure for hardwood, 1-2mm in length and have a length to diameter aspect ratio of roughly 1:100. In softwoods the tracheides cells are responsible for the supportive structure with a length of 2-4 mm. The strength of timber is significantly determined by the relative amount of supportive cells. Further different specimen can vary vastly in density from 120 kg/m^3 to 1200 kg/m^3. Hemicelluloses and lignin are the cementing materials that bind the structural components of the timber together and as such contribute to its strength and stiffness.

[1]Procter & Gamble

In most simulation methods isotropic and anisotropic materials only differ in the way the stress is computed from strain. The two Lamé constants λ, μ describe the stiffness and shear properties of isotropic materials, however for anisotropic materials 21 parameters are needed[2]. Not only is it an unwieldy number of parameters for both user and developer, they are also a lot less intuitive than the Lamé constants. The parameter μ for example describes the material resistance to shear-deformations. The key insight gained from the short timber discussion is that the combination of two isotropic materials can yield an anisotropic one. For our simulation we combine a rod framework, that represents the fibre-cells, and a solid simulation, that represents the cementing components of timber.

There is a wide range of applications for anisotropic materials. We already explained that most plants are anisotropic in nature. The muscle-tissue of most animals is also fibrous and thus, anisotrophic. For example to analyze the stresses within the heart-muscle one needs to employ an anisotropic material model and even change the direction of anisotrophy throughout the object. Our goal is to provide an anisotrophic material model for all these applications that allows an approximation of actual anisotrophic behavior with interactive frame-rates.

1.2 Structure of the Thesis

The majority of the thesis will cover and summarize previous works in the field of interactive dynamics and fracture. In chapter 2 a very short introduction into continuum mechanics is presented that covers the major theoretical aspects that many of the dynamic frameworks rely on. Based on these concepts a set of abstract building blocks is derived which are at least in part present in any dynamic model. Chapter 3 presents different methods to compute the elastic response of an object and special cases for rods, sheets and solids. Due to the limited scope of this thesis some methods are covered only very briefly others not at all. Instead large parts of the chapter are devoted to Position Based Dynamics(PBD) as our implementation is based on it. Chapter 4 then deals with the fracturing of objects. Therefore we discuss plastic deformations and fracture criteria and how these are realized in the different methods from previous chapter. The review of previous work is then concluded by a detailed description of the fracture process for oriented particles.

[2]Let $\boldsymbol{\varepsilon} \in \mathbb{R}^{3\times3}$ then the rank four tensor $\boldsymbol{C} \in \mathbb{R}^{3\times3\times3\times3}$ connects the strain $\boldsymbol{\varepsilon}$ to the stress $\boldsymbol{\sigma}$. Partly due to symmetry we actually only need 21 out of the 81 components to compute the remaining 60.

In chapter 5 we move on to our own method and discuss the different choices of rod-configuration as well as the choice of strain-measurement and it's correction. These choices are then compared to the approach that is closest to our own, Sutherland (2012). Chapter 6 discusses implementation specific details. As our models requires information that is not present in the usual object-formats the loading and model-preparation is shortly discussed. The current implementation of rod and particle placements employs simple greedy algorithms and as such improvements for group-clustering and rod-placements are suggested. The current capabilities and limits of the rendering are and not yet implemented meshing approach is schematically presented. At last chapter 7 discusses the successes and shortcomings of the presented method. The potentially most detrimental shortcomings, rate of convergence and level of detail, is discussed in more detail and suggestions which need to be investigated in future work are presented.

Chapter 2

Mechanical Simulations

Mechanics is the study of motion (kinematics), of forces which result in a change of motion (dynamics) and the interaction between forces such as the collision of different objects and the resulting deformations. This chapter gives a quick overview of the most essential aspects of continuum mechanics needed for interactive simulations. This chapter is based on the more complete and rigorous discussion in Y. Fung (1993), Abeyaratne (2012) and freely accessible resource provided by McGinty (2018).

2.1 Continuum Mechanics

The concept of a continuum is well understood from number sets such as the real or complex numbers. Between any two real numbers there is another real number in between them thus, between any two real numbers there are infinity many real numbers. Intuitively we can view both time and three-dimensional space around us as continuous. While the assumption of continuous matter obviously fails at atomic scale it is sufficient for a large range of engineering applications and even more so for computer graphics.

Since we assume space to be continuous we can describe its state by an infinite set of particles at every location in space. Particle might be associated with additional parameters such as mass, temperature or even extensive properties such as density. Also every natural object is bound by a particular *region* $\mathcal{R}$ in space. This implies that we can describe the state of natural objects with the particles within that region $\mathcal{R}$. The subset of particles is called *body* $\mathcal{B}$ and is an abstract representation of the real objects. This abstraction allows to apply mathematical operations and concepts to real objects. For example, viewing matter as a continuous distribution in space allows for a one-to-one correspondence between the particles of the body $p \in \mathcal{B}$ and positions within the region $\mathbf{x} \in \mathcal{R}$S:

$$\mathbf{x} = \mathcal{X}(p) \tag{2.1}$$

$$\mathcal{R} = \mathcal{X}(\mathcal{B}) \tag{2.2}$$

Other physical quantities for example temperature correspond directly to a particle and thus, a similar mapping can be defined for any such quantity θ:

$$\theta = \theta_*(p) \tag{2.3}$$

for $p \in \mathcal{B}$. While this definition is correct a particle is an abstract entity. As such it is not particularly convenient to reference to a particle to be evaluated. Instead, it is more intuitive to query the objects properties at a specific position in space. Thanks to the one-to-one correspondence this can be easily achieved:

$$\theta = \bar{\theta}(\mathbf{x}) \overset{\text{def}}{=} \theta_*(\mathcal{X}^{-1}(\mathbf{x})) \tag{2.4}$$

Extensive properties such as density are not directly associated with a particle but rather a region in the body. However, under the assumption of continuous matter the mapping can be extended to include such properties. To exemplify this point let us consider the concept of density more closely. If mass m_n is a measurement for the amount of matter in a corresponding region V_n then density ρ at a point $\mathbf{x} \in \mathcal{R}$ can be defined as the ratio of mass per volume:

$$\rho(\mathbf{x}) = \lim_{\substack{n \to \infty \\ V_n \to 0}} \frac{m_n}{V_n} \tag{2.5}$$

where the volume V_n is centered around $\mathbf{x}$. With the inverse mapping $\mathcal{X}^{-1}$ the extensive properties can now be mapped to a specific particle.

The most important implication of the continuity assumption is the ability to use rigorously defined mathematical tools such as differentiation and integration. As such continuity is the very foundation of classical mechanics which describes the behavior of natural objects such as rigid-bodies (Newton), fluids (Navier-Stokes) or electromagnetism (Maxwell) with differential equations.

2.1.1 Reference Configuration and Deformation

The abstraction of natural objects into a mathematical description of a body enables us to describe a single *state* or *configuration* of said objects. In order to capture changes in between different configurations, such as movements or deformations, time needs to be considered. To capture these changes the previous body-to-space-mapping needs to be extended by a time parameter:

$$\mathbf{x} = \mathcal{X}(p, t) \tag{2.6}$$

In eq. (2.4) we changed the parameter of the function to position as it gives a more intuitive mean of referencing particles. By introducing time to this mapping it is no longer clear how to label a particular particle as its position is subject to change with respect to time. However, as long as the time parameter is fixed the mapping is unique. By choosing a specific time or configuration we can reference a particular particle and use its index to track it in all configurations. This specific configuration is called material or rest configuration and we use a *material/rest* position $\tilde{\mathbf{x}}$ to reference particles $\tilde{\mathbf{x}} = \mathcal{X}_{ref}(p) \overset{\text{def}}{=} \mathcal{X}(p, t_{ref})$ in it. This allows to compute the position, or any other quantity, at the current time t based on the position in the rest configuration:

$$\mathbf{x} = \mathcal{D}(\tilde{\mathbf{x}}, t) \overset{\text{def}}{=} \mathcal{X}(\mathcal{X}_{ref}^{-1}(\tilde{\mathbf{x}}), t) \tag{2.7}$$

This mapping allows us to study geometric changes between configurations or, in other words, deformations of the body with respect to this reference configuration. Thus $\mathbf{x}$ is the deformation of $\tilde{\mathbf{x}}$ and $\mathcal{D}$ is the deformation mapping.

2.1.2 Strain and Stress

In order to analyze the behavior of a material, the rigid body translation and rotation needs to be removed from the deformation function. This leaves only deformations that prompt a response from the material. The relative measure of these non-rigid-body-deformations is called *strain*. Strain describes how the extend of the object has changed compared to its rest extend and as such is a unit-less measurement. This also means that strain is independent of the particular extend of an object, thus, it can be used to define the general behavior of all materials. In order to derive strain we first calculate the deformation gradient:

$$\mathbf{F} = \frac{\partial \mathbf{x}_i}{\partial \tilde{\mathbf{x}}_j} = \begin{bmatrix} \frac{\partial \mathbf{x}_1}{\partial \tilde{\mathbf{x}}_1} & \frac{\partial \mathbf{x}_1}{\partial \tilde{\mathbf{x}}_2} & \frac{\partial \mathbf{x}_1}{\partial \tilde{\mathbf{x}}_3} \\ \frac{\partial \mathbf{x}_2}{\partial \tilde{\mathbf{x}}_1} & \frac{\partial \mathbf{x}_2}{\partial \tilde{\mathbf{x}}_2} & \frac{\partial \mathbf{x}_2}{\partial \tilde{\mathbf{x}}_3} \\ \frac{\partial \mathbf{x}_3}{\partial \tilde{\mathbf{x}}_1} & \frac{\partial \mathbf{x}_3}{\partial \tilde{\mathbf{x}}_2} & \frac{\partial \mathbf{x}_3}{\partial \tilde{\mathbf{x}}_3} \end{bmatrix} \tag{2.8}$$

Alternatively, the deformation gradient can be expressed in terms of *displacement* $\mathbf{k} = \mathbf{x} - \tilde{\mathbf{x}}$:

$$\mathbf{F} = \frac{\partial}{\partial \tilde{\mathbf{x}}}(\tilde{\mathbf{x}} + \mathbf{k}) = \frac{\partial \tilde{\mathbf{x}}}{\partial \tilde{\mathbf{x}}} + \frac{\partial \mathbf{k}}{\partial \tilde{\mathbf{x}}} = \mathbf{I} + \frac{\partial \mathbf{k}}{\partial \tilde{\mathbf{x}}} \tag{2.9}$$

Notice that the deformation gradient $\mathbf{F}$ is invariant with respect to rigid body translations as $\mathbf{k}$ stays constant under such translations. If only rigid body translations are applied to the body, the deformation gradient is the identity matrix, $\mathbf{F} = \mathbf{I}$. Unfortunately, rigid-body rotations are still present in the deformation gradient. There are multiple ways to decompose this matrix to remove its rotational components such as QR-decomposition. Here we will present the polar decomposition as it is also used in the approaches described in sections 3.3 and 4.3.

Polar Decomposition

The transformation gradient contains some rotation $\mathbf{R}$ and the non-rigid body transformation $\mathbf{U}$ that we are interested in. Thus, we should be able to decompose the deformation gradient into a multiplication of the two, $\mathbf{F} = \mathbf{RU}$. By definition[1] the multiplication of a matrix with it's transpose yields a symmetric matrix.

$$\mathbf{F}^T\mathbf{F} = (\mathbf{RU})^T(\mathbf{RU}) = \mathbf{U}^T\mathbf{R}^T\mathbf{RU} \tag{2.10}$$

As the matrix $\mathbf{R}$ is assumed to be a rotation matrix it has to be orthogonal and skew-symmetric. The transpose of an orthogonal matrix is equal to its inverse thus $\mathbf{R}^T\mathbf{R} = \mathbf{I}$. Further any matrix can be decomposed excatly into a symmetric and a skew-symmetric part. Since the rotation is known to be skew-symmetric, $\mathbf{U}$ has to be symmetric.[2] In other words this leaves us with the square $\mathbf{U}^T\mathbf{U} = \mathbf{U}^2$ of the matrix $\mathbf{U}$ that we are interested in. One way to take a square root from a matrix is to diagonolize it using its eigenwert decomposition.

$$(\mathbf{U}^2) = \mathbf{VDV}^{-1} \tag{2.11}$$

where D is a diagonal matrix containing the eigenvalues of $\mathbf{U}$ and the matrix $\mathbf{V}$ contains the corresponding eigenvectors. Now the square root of $\mathbf{D}$ can be taken by the square root of its separate entries to obtain $\mathbf{U}$:

$$\mathbf{U} = \mathbf{VD}^{\frac{1}{2}}\mathbf{V}^{-1} \tag{2.12}$$

This leaves us with the non-rigid body transformation $\mathbf{U}$. The corresponding rotation can now easily be calculated by $\mathbf{R} = \mathbf{FU}^{-1}$.

[1] A matrix is defined to be symmetric if $A = A^T$. Thus, the multiplication of any matrix with its transpose yields a symmetric matrix $(A^TA)^T = A^T(A^T)^T = A^TA = S$

[2] A matrix is symmetric if $\mathbf{A} = \mathbf{A}^T$ and anti-symmetric if $\mathbf{A} = -\mathbf{A}^T$. Thus, any matrix can be decomposed into a symmetric and an skew-symmetric matrix by:

$$\mathbf{A} = \frac{1}{2}(\mathbf{A} + \mathbf{A}^T) + \frac{1}{2}(\mathbf{A} - \mathbf{A}^T)$$

Strain

Finally, we can define strain as a deviation from pure rigid body transformation in which case $\boldsymbol{U}$ would be $\boldsymbol{I}$.

$$\boldsymbol{\varepsilon} = \mathbf{U} - \mathbf{I} \tag{2.13}$$

Since $\boldsymbol{U}$ is symmetric $\boldsymbol{\varepsilon}$ is implied to be symmetric as well. While this definition is quite useful it is computationally expensive, thus, other measurements of strain are commonly used, such as Almansi-strain, Cauchy-strain, Greens-strain or logarithmic strain. The strain-measure used most frequently in this thesis is the *Cauchy-Green* strain defined as:

$$\varepsilon_G = \frac{1}{2}(\mathbf{F}^T\mathbf{F}) - \mathbf{I} \tag{2.14}$$

Notice that the rigid-body rotations have already been removed as in equation 2.10, however, this definition refrains from calculating the actual tensor $\mathbf{U}$. By not taking the square root from the deformation gradient quadratic terms are introduced into the solution. Thus, this definition can only be used for small to modest amounts of deformation as its error is quadratically correlated with the amount of strain. Most simulation require small time-steps for the integration anyway and as such this is a popular approach.

Stress

The strain that was just calculated provides the raw information to compute the internal elastic and dampening forces[3]. However, it is computed independent of material properties. The *stress* measurement then correlates the strain to elastic and shear responses of a specific material. The total internal stress $\boldsymbol{\sigma}$ is simply the sum of the elastic stress due to strain $\boldsymbol{\varepsilon}$ and stress due to strain rate ν:

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}^{(\varepsilon)} + \boldsymbol{\sigma}^{(\nu)} \tag{2.15}$$

The most general linear constitutive model relates the strain $\boldsymbol{\varepsilon}$ to the stress with 81 elastic material properties stored in the rank four tensor $\mathbf{C}$:

$$\boldsymbol{\sigma}_{ij}^{(\varepsilon)} = \sum_{k=1}^{3}\sum_{l=1}^{3} \mathbf{C}_{ijkl}\, \varepsilon_{ij} \tag{2.16}$$

Obviously, for most applications this is an unpractical way to specify the material properties. By constraining ourself to isotropic materials the parameters can be reduced to only two independent variables λ and μ called

[3]For now a pure elastic material response is assumed, plastic deformation is covered in chapter 4

the Lamé coefficients. The first parameter λ describes the elasticity of the material or in the context of fluids the viscosity. The second parameter μ describes the materials response to shear strain. Many other elastic and shear moduli exist some notable examples include Young's modulus for elasticity, Possion's ratios or the bulk modulus. Any two of these moduli fully describe an isotropic material properties and, thus, can be converted to any other modulus representation.

To calculate the viscous stress $\boldsymbol{\sigma}^{(\nu)}$ we first need to differentiate the Greens-strain with respect to t. Since $\tilde{\mathbf{x}}$ is independent of t differentiation is straight forward:

$$\dot{\mathbf{F}} = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathbf{x}_i}{\partial \tilde{\mathbf{x}}_j}\right) = \frac{\partial \dot{\mathbf{x}}_i}{\partial \tilde{\mathbf{x}}_j} \tag{2.17}$$

Which can then be used to differentiate Greens-strain.

$$\nu_{ij} = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{1}{2}\left(\mathbf{F}^T\mathbf{F}\right) - \mathbf{I}\right) = \frac{1}{2}\left(\mathbf{F}\dot{\mathbf{F}}^T + \dot{\mathbf{F}}\mathbf{F}^T\right) \tag{2.18}$$

Which in turn can then be used to define the viscous stress with a four rank tensor $\mathbf{D}$ containing the dampening coefficients.

$$\boldsymbol{\sigma}_{ij}^{(\nu)} = \sum_{k=1}^{3}\sum_{l=1}^{3} \mathbf{C}_{ijkl}\, \nu_{ij} \tag{2.19}$$

Finally, we can apply Hooke's law which linearly corresponds elastic strain and stress to the elastic energy potentials η and the strain rate and viscous strain to a dampening potential κ at any point in the material.

$$\eta = \frac{1}{2}\sum_{i=1}^{3}\sum_{j=1}^{3} \boldsymbol{\sigma}_{ij}^{(\varepsilon)}\, \varepsilon_{ij} \tag{2.20}$$

$$\kappa = \frac{1}{2}\sum_{i=1}^{3}\sum_{j=1}^{3} \boldsymbol{\sigma}_{ij}^{(\nu}\, \nu_{ij} \tag{2.21}$$

These quantities can be integrated over the volume of the body to obtain the objects elastic and dampening potentials, respectively. By subtracting the applied external potentials from these body potential we get the total potential energy which needs to be minimized to find the objects equilibrium. Alternatively, we can use the stress to compute the internal force at any particular location. The force acting on a plane perpendicular to a unit normal $\hat{\mathbf{n}}$ in an infinitesimal volume centered at a specific point is defined as:

$$\mathbf{f} = \boldsymbol{\sigma}\, \hat{\mathbf{n}} \tag{2.22}$$

where $\mathbf{f}$ is the force per unit area.

$$\begin{tikzcd}
\text{Position} & \text{Strain} & \text{Stress} \\
\mathbf{x} \arrow[r, "\partial"] & \epsilon \arrow[r, "\text{Material}"] & \sigma \arrow[d, "\Delta"] \\
\mathbf{v} \arrow[u, "\int"] & \mathbf{a} \arrow[l, "\int"'] & \mathbf{f} \arrow[l, "\text{Mass}"'] \\
\text{Velocity} & \text{Acceleration} & \text{Force}
\end{tikzcd}$$

Figure 2.1: Summary of the connection between the different quantities discussed in this chapter.

For some materials, like fluids with low viscosity, the change in geometry is not needed to describe its behavior. It is independent of an initial configuration. However, most solid models require the strain or stress quantities we just defined.

2.2 Simulation Framework

Continuity is a nice theoretical tool, however, to perform computer aided simulation the notion of continuity needs to be abandoned due to limited storage and computational resources. This implies that both the space sampling i.e. the particle count and the time-step size between configurations needs to be finite. There is a multitude of numerical methods for solving PDEs. Some of them, such as smoothed particle hydrodynamics (SPH), operate directly and only on the particles of the object. Other, such as finite element method (FEM), decompose the region of the object into a mesh of primitive shapes called elements. The continuous function is approximated by interpolating the particle values across each element and as such across the object in a piece-wise continuous manner. Independent of the actual method, and weather it is mesh-less or not, it will always provide an approximation to the continuous function based on the values of discrete particles. This also allows to differentiate or integrate the function which in turn allows us to solve the PDE, at least locally. To summarize, the values at any position in the region of the object can be approximated from the discrete particle by a numerical methods.

Before we can discuss a simulation and utilize this abstract numerical approximation method the initial configuration needs to be specified. For this a finite amount of particles is scattered within the region of the object. Each particle is associated with a number of properties and usually includes at least position $\mathbf{x}$, velocity $\mathbf{v}$, acceleration $\mathbf{a}$, rest-position $\tilde{\mathbf{x}}$ and mass m. For every particle the property values need to be initialized. How the particles

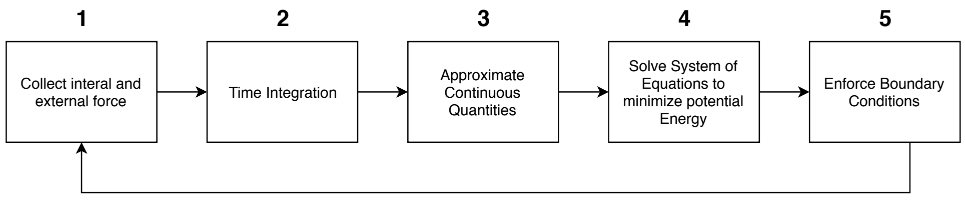

1
2
3
4
5
Collect interal and external force
Time Integration
Approximate Continuous Quantities
Solve System of Equations to minimize potential Energy
Enforce Boundary Conditions

Figure 2.2: Abstract building block for a physical simulation.

are scattered depends on the particular numerical method of choice. For FEM in three dimension tetrahedral meshes are popular leading to a regular grid structure for particle placement. The SPH method on the other hand requires uniformly scattered particles to cover the region as evenly as possible and thus, reduce the expected error. As the particles are usually not directly visible in the finial application grid structure can be used here as well. Often the initial configuration is also used as the rest-configuration but it could also be specified independently.

In most scenarios the user also wants to specify a container with which the object can collide, dissolve on impact or otherwise interact with, these are called *boundary conditions*. Finally external forces such as gravity, wind or other loads should be defined as without them the elastic response will always be zero. With an abstract idea of all aspects of a simulation in place fig. 2.2 outlines the execution of a single simulation step. It should be noted that particular methods may rearrange, skip or extend steps to this outline. None the less these building blocks provide a general high level overview of the most common components utilized in many simulation systems and as such provides a good ground for comparison. In the following the five steps of fig. 2.2 are discussed:

First: The defined external forces are applied to all applicable particles. Some forces like gravity affect all particles while other forces like point loads may only affect a single particle. Along with each particles mass the acceleration $\mathbf{a}$ can be computed according to Newtons second law of motion $\boldsymbol{f}_{ext} = m\mathbf{a}$. Then these additional external forces are added to possible correction forces from a previous time-step $\boldsymbol{f} = \boldsymbol{f}_{ext} + \boldsymbol{f}_{int}$.

Second: Either we start with a initial configuration or we have a valid configuration from the previous time-step. With a previous valid configuration we need to compute the current configuration. Suppose t describes the current point in time for which the quantity in question is still unknown. Unfortunately, there is general no function that describes the quantities values across time $q(t)$. For example, for computer-graphics one is only interested in the next positions of the particle, however, there is no function $\mathbf{x}(t)$ that returns a field of positions for the particles at that point in time. However, the current value of $q(t)$ can be described in terms of its derivative:

$$\frac{\partial q}{\partial t} = f(q) \tag{2.23}$$

Which can then be used to get the actual value of the quantity at any point in time without the actual function $q(t)$ by utilizing the previous values.

$$q(t) = \int_0^t f(q(\tau))\mathrm{d}\tau \tag{2.24}$$

Still we are dealing with a discrete position and also the time domain has to be discretized. Thus, in our simulation we start with a time-step $\mathrm{n} = 0$ where $\mathrm{n} \in \mathbb{N}$ and define our integration methods such that they approximate the integral as we go from time-step n to $\mathsf{n}+1$. One common and simple numerical integration method is the explicit Euler:

$$\mathbf{x}^{\mathrm{n}+1} = \mathbf{x}^{\mathrm{n}} + \Delta t f(q^{\mathrm{n}}) \tag{2.25}$$

where $f(q^{\mathrm{n}})$ describes the numerical differentiation with respect to t. This allows us to compute the position at the next time-step $\mathbf{x}^{\mathrm{n}+1}$. There are many other integration methods, many of which trade simplicity and computational expense for higher accuracy or better stability. Another popular approach for interactive applications is the leap-frog integration which interleaves update and derivative update steps. For higher accuracy implicit integration methods are preferred, these first compute the derivative at the current point in time and then calculate the current value based on it.

Third: Based on the new positions $\mathbf{x}^{\mathrm{n}+1}$ the chosen PDE solver can compute the current deformation function, locally approximate strain and stress and finally compute the potential energy at every particle. Chapter 3 covers different methods for this step in more detail.

Forth: The objects equilibrium or ideal particle position has been reached when the total potential energy function is minimized.

$$\mathrm{II} = \mathbf{W}(\mathbf{x}_{eq}) - \mathbf{W}_{\mathbf{ext}} \tag{2.26}$$

One way to solve for $\mathbf{x}_{eq}$ is to assume a system of only conservative forces and as such the total potential energy is 0. Equating body and external potentials create a system of equation that can be solved by linearizing with Taylor-expansion and then utilize widely available BLAS libraries to solve the system.

Fith: Solving the equation-system gives us a corrected position $\mathbf{x}^{n+1}$ to which we can apply the boundary conditions. This can either be done by simply projecting the position into a valid region and adjusting the velocity or by introducing penalty forces which will be applied at the beginning of the next simulation step. Finally, we increment the time-step variable n and can start the calculation of the next time-step by going back to step one.

Chapter 3

Methods of Interactive Simulation

A variety of models have been used to simulate the elastic behavior of objects. While most methods still reference back to continuum mechanics many of them significantly reduce the required computation by ignoring some physically relevant aspects. For example mass-spring system, as the name suggests, connect particles with spring forces. This make it simple to implement, as only a single force type has be be realized and and no additional quantities such as strain, stress or density need to be computed. However mass-spring-system do not treat the space as a continuum, as such no physical meaning-full material parameters can be derived. This makes it very difficult to parameterize such a system to correspond to a certain type of material as only the spring-stiffness between each particle-pair can be adjusted. Models that treat objects as a continuum benefit from physical meaningful and thus more intuitive parametrization. This chapter presents Finite Element method, Position-Based-Dynamics and Shape-Matching. Another common method SPH is not discussed in this chapter, however very briefly referenced to in section 6.1.

3.1 Finite Elements

The finite element method (FEM) requires the object in question to be decomposed into a connected mesh of elements. Potential elements in three dimensions are tetrahedrons, cubes or other platonic solids. For two dimension triangles, rectangles and other regular polygons respectively. This allows to interpolate the values of the mesh particles within each element. As such we have a continuous set of value within each element. Since the object is represented as a mesh of the elements all particle quantities can be interpolated across the object in a piece wise continuous matter.

While the PDE can not directly be solved for the whole region of the object the simplicity of the chosen elements allows for analytical integration and thus the PDEs can be solved for each element. First the deformation is interpolated across the element allowing us to compute the strain. Applying the material properties to the constitutive model the stress is computed. Integrating the stress of the element yields the potential energy for the element. To get the physical behavior of the whole object the vertex solutions are combined at their corresponding particle under a continuity condition. This allows us to define the elastic potential density per element as:

$$\eta^{\text{element}} = \frac{1}{2}\int_V \eta \mathrm{d}V \tag{3.1}$$

where η is defined as in eq. (2.20). In order to minimize eq. (2.26) we still need the external forces that are applied to a specific element. Having already defined the displacement e this can easily be expressed as:

$$\mathbf{W}_{ext} = \int_V e(\mathbf{x})\boldsymbol{f}_{ext}\mathrm{d}V \tag{3.2}$$

where $\boldsymbol{f}$ denote the sum of all external forces acting on the particles corresponding to the elements vertices. Assuming that we only consider conservative forces in a closed system gives us the equation for total potential energy eq. (2.26), for each element. In order to minimize this equation its derivative has to be zero. For M elements and N particles we get an $M \times N$ sparse system of equations that needs to be solved for the position differential at the particle positions. Each equation depends only on the discrete deformed vertex positions.

3.1.1 Real Time FEM

The FEM approach is used extensively in the engineering community and also in the computer graphics community even for interactive simulation of solids and fractures, Matthias Müller et al. (2001) and O'brien et al. (2002, 1999). In order to achieve interactive frame rates most of these methods employ a few accuracy-speed trade offs. First they usually rely on linear shape functions of either of triangle (2D) or tetrahedral (3D) elements as these result in a constant deformation gradient across the element, as explained in section 3.2.3 page 22. This eliminates the need for numerical integration in eq. (3.1), as the volume of the element can be computed analytically and η is constant. This is precisely the reason why engineering applications rely on higher order interpolation or other element types as this reduces the accuracy of the results in an inconsistent manner. In order to maintain visual fidelity there are separate meshes for shading and physical simulations. For example, the convex hull of the object can be used to generate the tetrahedral mesh.

In recent years there have been a few notable advances in adapting FEM to allow for visually plausible results at interactive frame rates. The first papers on interactive fracture by Matthias Müller et al. (2001) and O'brien et al. (2002, 1999) are also based on FEM. The thesis Flores (2015) gives a more detailed overview of interactive use of FEM. Ngan et al. (2008) apply stiffness-warped small-strain linear elastic models to nonlinear elements and present an efficient implementation.

3.2 Position Based Dynamics

Position based dynamics (PBD) started as a non-physical approach which has found wide spread adoption[1] due its simplicity and stability. Instead of computing and integrating forces to correct the position, PBD computes the position corrections directly. This is achieved by constraining particle positions with respect to each other and then solving the system of constraints. It was originally introduced by Matthias Müller et al. (2007), then in the siggraph course notes on interactive dynamics Matthias Müller et al. (2008) a whole chapter was devoted to it and finally the survey paper Bender et al. (2014a), Bender et al. (2017) are entirely devoted to PBD and summarize the numerous extensions already published for this method. As these publications are either entirely or partially devoted to PBD some of the equations are identical across the different publications and will also be presented in this thesis.

3.2.1 Algorithm Overview

The object that is to be simulated is represented by a set of N particles and a set of M constraints. These constraints describe the behavior of the object and in full compliance with them the object would be completely stiff. By partial compliance, either through a stiffness-parameter for each constraint or by stopping the iterative solver before it converged, larger deformations that mimic elastic response can be created. The calculation of a time-step starts out very familiar to the pipeline in the section 2.2. Initially, all forces are summed and integrated to compute the velocity of the current time step.

[1]Implemented has been implemented in the open-source physics-engine bullet as well as in Nvidia PhysiX and Nvidia Flex.

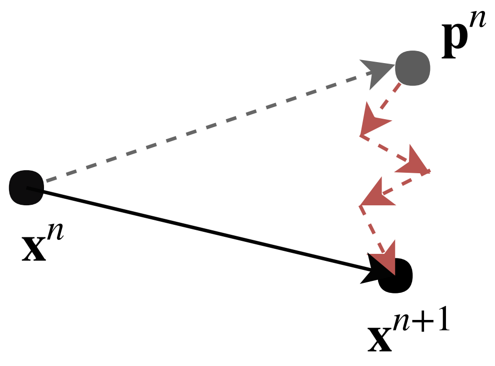


Figure 3.1: Overview of the PBD algorithm. For the time-step n the next position is predicted based on the current velocity (grey arrow). The prediction is corrected by the constraint solver (red arrows). Then the velocity is updated to account for the solver correction (black arrow).

Since the elastic response is not represented as a force the additional velocity is computed entirely from external forces $\boldsymbol{f}_{\text{ext}}$, line 2. The velocity can then be integrated to yield position. However, as only external forces where considered during integration the result does not represent the actual next position but rather a starting point which needs to be corrected to comply with the constraints. The particle position at the beginning of the time-step will be needed later so the computed position will be assigned to an additional property, the predicted position $\mathbf{b}$, line 3. Based on the prediction a collision detection dynamically generates a set of additional constraints that resolve these collisions, line 4. Then the solver, line 6, takes the set of predicted positions and projects them onto the individual constraint manifolds sequentially. The solver will be discussed in detail in the next section, for now we just assume that it has corrected the predicted positions to comply with the constraint set. Figure 3.1 depicts the process discussed so far. The grey dotted line describes the sum of the velocity of the previous time step and the additional velocity from the external forces. The red arrows describe the iterative correction from the constraint projection. The velocity will be used in the next time-step to compute the predicted position. However, the grey-line (velocity) does not yet account for the correction made by the solver. Since the position from the beginning of the time-step is still available as $\mathbf{x}$ this is easily rectified by finite difference approximation, line 10. This yields the actual velocity, black arrow. Finally, the predicted position can be assigned to the actual position, line 11, and then the next time-step can be computed.

Algorithm 1: Position based dynamics

```
1  Time integration with symplectic-Euler
2  forall particles $i$ do $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t m_i \boldsymbol{f}_{\text{ext}}(\mathbf{x}_i)$
3  forall particles $i$ do $\mathbf{b}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
4  forall particles $i$ do generateCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{b}_i$)
5  forall solverIteration do
6      projectConstraint($C_1, \dots, C_{M+M_{coll}}, \mathbf{b}_1, \dots, \mathbf{b}_N$)
7  end
8  Velocity correction
9  forall particles $i$ do
10     $\mathbf{v}_i \leftarrow (\mathbf{b}_i - \mathbf{x}_i)/\Delta t$
11     $\mathbf{x}_i \leftarrow \mathbf{b}_i$
12 end
```

3.2.2 Constraint System and Solver

The simplest constraint is the distance constraint. A distance constraint between two particles can be defined as:

$$C(\mathbf{x}_1, \mathbf{x}_2) = |\mathbf{x}_1 - \mathbf{x}_2|^2 - d^2 = 0 \tag{3.3}$$

where $\mathbf{x}_1, \mathbf{x}_2$ are the positions of two particles involved in the constraint, $|\cdot|$ is the euclidean norm and d describes the desired distance between the particles. In order to keep the constraint definition general and independent of PBD $\mathbf{x}$ is used to denote the position. The M constraints of the object yield a non-linear system of equation for the N particles. Since we need to solve for particles positions and every position has three values (x, y, z) the system is over-determined if $M > 3N$ or under determined if $M < 3N$. While in theory we could assign every particle exactly three constraints it would limit the use-cases and flexibility of the approach. However, constraining a under or over determined system to yield usable results is a difficult task and is further complicated by inequalities that result from collision constraints. A simple collision-constraint $C(\mathbf{x}_1, \mathbf{x}_2) = |\mathbf{x}_1 - \mathbf{x}_2|^2 - d^2 \geq 0$ keeps one particle *at least* a distance d from the other particle.

To make matters worse multiple constraints can create a system where the solution manifolds of the constraints do not overlap and as such no position that satisfies all constraints can be found[2]. Incompatible constraint sets can result from dynamic constraint creation, as in line 4. Some approaches even intentionally generate incompatible constraint sets that are

[2]The simplest case would connect two particle with two distance-constraints, instead of one, and different parameters d for each constraint.

supposed to complement each other e.g. bending and distance constraints. The numerical solver has to be able to return plausible approximation in all these conditions. PBD implementations usually use a Gauss-Seidel method or one of its derivatives to solve these systems. This approach rapidly returns rough approximations at the cost of poor convergence. Even if it convergences it usually takes a lot of iterations even for objects of moderate complexity.

We are going to derive the solver for a general constraint definition. This allows us to constraint the system of equations once for the general case which will then hold true for any particular constraint derived from the general one. Let $\mathbf{x}$ be the concatenation of all particle positions[3] $\mathbf{x} = [\mathbf{x}_1^T, \ldots, \mathbf{x}_N^T]^T$ then all particle positions can be passed to the general constraint $C(\mathbf{x})$ so that each particular constraint can choose which particle it actually operates on. The distance constraint, for example, would only reference to two out of the N positions. Additional constraint types can have an arbitrary amount of parameters without changing the following uniform constraint definition:

$$\begin{gathered} C_1(\mathbf{x}) \succeq 0 \\ \ldots \\ C_M(\mathbf{x}) \succeq 0 \end{gathered} \tag{3.4}$$

where $\succeq$ denotes either $=$ or $\geq$ depending on the particular constraint. The process starts by taking the current position as a guess for the solution of the system. Using Taylor-Expansion on each constraint individually linearlizes it with respect to its current neighborhood:

$$C(\mathbf{x} + \Delta\mathbf{x}) = C(\mathbf{x}) + \nabla C(\mathbf{x})\Delta\mathbf{x} + \mathcal{O}(|\Delta\mathbf{x}|^2) \tag{3.5}$$

where $\nabla C(\mathbf{x})$ is the derivative with respect to the N components of $\mathbf{x}$. Instead of computing a global solution we will solve every constraint individually. Each constraint will be solved for $\Delta\mathbf{x}$ which describes the correction needed to translate the current position such that the constraint is satisfied. Since every constraint is solved individually there is a single equation and three unknown scalars in $\Delta\mathbf{x}$ making the system highly underdetermined. As such the system needs to be constraint to a single unique solution first. A desirable property is to conserve linear and angular momentum. This is the case if $\Delta\mathbf{x}$ is parallel to ∇C. Applying this parallel condition reduces the system to a unique solution by $\Delta\mathbf{x} = \lambda\mathbf{M}^{-1}\nabla C^T$ where $\mathbf{M} = diag(m_1, \ldots, m_N)$. The equation with a single scalar unknown λ is given by:

$$C(\mathbf{x}) + \lambda\mathbf{M}^{-1}\nabla C^T \succeq 0 \tag{3.6}$$

[3]The actual input in the PBD algorithm is not the actual position but rather the predicted position.

For all particles j that are participating in the constraint C the correction for a particular particle i is:

$$\Delta \mathbf{x}_i = \underbrace{\frac{C(\mathbf{x})}{\sum_j w_j |\nabla_{x_j} C(\mathbf{x})|^2}}_{\lambda} w_i \nabla_{\mathbf{x}_i} C(\mathbf{x})^T \tag{3.7}$$

Now that every constraint can be solved in isolation their results can be combined to approximate the global solution. This is achieved by simply applying the correction of each constraint directly such that subsequent constraints work with already corrected positions. This is known as the Gauss-Seidel solver and is summarized in algorithm 2. Since λ is constant for a constraint it only needs to be computed once for every constraint, line 4. Next the corrections for each particle of the current constraint is calculated, lines 5 to 7 . The correction from one constraint are applied to the global $\mathbf{x}$ so that the next constraint works with the corrected positions, line 8. Inequalities are handled by a if-condition at the beginning of each constraint-projection, line 3.

Algorithm 2: Gauss-Seidel Solver

```
1  forall SolverIterations do
2      forall Constraints $i \in \{0, .., M\}$ do
3          if $C_i(\mathbf{x}) \leq 0$ then
4              $\lambda \leftarrow \frac{C_i(\mathbf{x})}{\sum_j w_j |\nabla_{x_j} C_i(\mathbf{x})|^2}$
5              forall particles $k$ of Constraint $C_i$ do
6                  $\Delta \mathbf{x}_k \leftarrow \lambda w_i \nabla_{\mathbf{x}_i} C_i(\mathbf{x})^T$
7              end
8              $\mathbf{x} \leftarrow \mathbf{x} + \Delta \mathbf{x}$
9          end
10     end
11 end
```

Notice that the correction and the rest of the Gauss-Seidel algorithm are independent of the chosen time-step making it stable regardless of the time-step size. This is especially useful for interactive simulation where large time steps are required to simulate objects behavior in real-time. A drawback of the local solver is that the stiffness of the object depends on the discretization and constraint ordering, figs. 3.2a and 3.2b. While the time-step does have no effect on the correction step itself it does influence how much change the external force introduces to the system and thus how large the corrections need to be. Some approaches to improve the convergence are discussed in section 3.2.6. But arguably the biggest downside is that the algorithm can not be parallelized since the results from previous constraint are used to compute the next $\Delta\mathbf{x}$.

This can be remedied by making a minor adjustment to the algorithm which is then called Jacobi-Method, algorithm 3. Instead of using the correction from previous constraints the correction $\Delta\mathbf{x}$ for each constraint is calculated based on the $\mathbf{x}$ vector at the beginning of the solver iteration, line 7. Since the summation is commutative[4] the inner loop, line 3 can be parallelized and their individual $\Delta\mathbf{x}$ vectors can then be summed after each solver-iteration and applied to the actual position vector, line 11. Also notice that the Jacobi solver converges to a solution independent of constraint ordering, which differs from the Gauss-Seidel solution in case of mutually exclusive constraint, fig. 3.2.

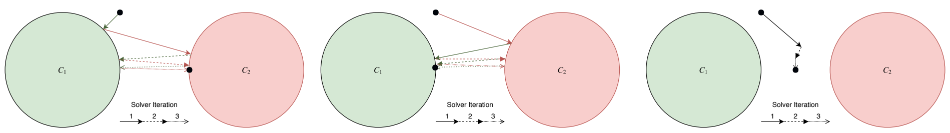
Algorithm 3: Jacobi solver

```
1  forall SolverIterations do
2      $\Delta\mathbf{x} \leftarrow \mathbf{0}$
3      forall Constraints $i \in \{0, .., M\}$ do
4          if $C_i(\mathbf{x}) \leq 0$ then
5              $\lambda \leftarrow \frac{C_i(\mathbf{x})}{\sum_j w_j |\nabla_{x_j} C_i(\mathbf{x})|^2}$
6              forall particles $k$ of Constraint $C_i$ do
7                  $\Delta\mathbf{x}_k \leftarrow \Delta\mathbf{x}_k + \lambda w_i \nabla_{\mathbf{x}_i} C_i(\mathbf{x})^T$
8              end
9          end
10     end
11     $\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}$
12 end
```

3.2.3 Strain Based Dynamics

Matthias Müller et al. (2014) derived PBD constraints based from continuum mechanics which allows to control stretch and shear deformation independently. This is achieved by deriving a PBD constraint from strain. Hence they named this approach strain based dynamics. The measurement of strain is done as for FEM simulations by decomposing the object into elements. They use tetrahedrons for three dimensional objects and triangles

[4]This may not be true for floating-point operations but since the solver itself is rather inaccurate approximation to the global solution to the system we can assume that the ordering of the $\Delta\mathbf{x}$ summation is irrelevant.



(a) Gauss-Seidel C_1 **(b)** Gauss-Seidel C_2 **(c)** Jacobi-method

Figure 3.2: In case of two mutually exclusive constraint C_1, C_2 the solution of the Gauss-Seidel method depends on the ordering of the constraints, a,b, also notice that the algorithm eventually alternates between two positions. The Jacobi method, c, on the other hand converges to a position where both constraints are equally satisfied or equally violated, thus, converges to the same solution independent of constraint ordering.

for two dimensional objects. Thus, we need to decompose the object that we wish to simulate into a tetrahedral-mesh[5].

Now we will derive the deformation-gradient for tetrahedrons and demonstrate that it is constant across the element for linear interpolation. Let k denote vertex 1 to 4 of a single tetrahedron. Then let $[u_k, v_k, w_k]^T$ be the position of the particle corresponding to the vertex k in rest configuration and let $[x_k, y_k, z_k]^T$ be position in the current configuration. In order to derive the deformation gradient for the element we need to interpolate the vertex position across the element. We can then interpolate vertex-values with the three linear weights r, s, t in both the rest and current configuration. A particular set of weights describes the same position in both configurations. The x-component $\mathbf{x}_1$ of the rest-position $\tilde{\mathbf{x}}$ can then be linearly interpolated by:

$$\tilde{\mathbf{x}}_1 = ru_1 + su_2 + tu_3 + (1 - r - s - t)u_4 \tag{3.8}$$

where x_k is the x-component of k-th vertex. The same set of weights can be used to compute the corresponding current position:

$$\mathbf{x}_1 = rx_1 + sx_2 + tx_3 + (1 - r - s - t)x_4 \tag{3.9}$$

The other two components for both the current and rest-position can be express equivalently. Now we can apply the definition of the deformation gradient from eq. (2.8) and rearrange it in terms of the linear interpolation weights:

$$\mathbf{F} = \frac{\partial \mathbf{x}}{\partial \tilde{\mathbf{x}}} = \begin{bmatrix} \frac{\partial \mathbf{x}_1}{\partial \tilde{\mathbf{x}}_1} & \frac{\partial \mathbf{x}_1}{\partial \tilde{\mathbf{x}}_2} & \frac{\partial \mathbf{x}_1}{\partial \tilde{\mathbf{x}}_3} \\ \frac{\partial \mathbf{x}_2}{\partial \tilde{\mathbf{x}}_1} & \frac{\partial \mathbf{x}_2}{\partial \tilde{\mathbf{x}}_2} & \frac{\partial \mathbf{x}_2}{\partial \tilde{\mathbf{x}}_3} \\ \frac{\partial \mathbf{x}_3}{\partial \tilde{\mathbf{x}}_1} & \frac{\partial \mathbf{x}_3}{\partial \tilde{\mathbf{x}}_2} & \frac{\partial \mathbf{x}_3}{\partial \tilde{\mathbf{x}}_3} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathbf{x}_1}{\partial r} & \frac{\partial \mathbf{x}_1}{\partial s} & \frac{\partial \mathbf{x}_1}{\partial t} \\ \frac{\partial \mathbf{x}_2}{\partial r} & \frac{\partial \mathbf{x}_2}{\partial s} & \frac{\partial \mathbf{x}_2}{\partial t} \\ \frac{\partial \mathbf{x}_3}{\partial r} & \frac{\partial \mathbf{x}_3}{\partial s} & \frac{\partial \mathbf{x}_3}{\partial t} \end{bmatrix} \begin{bmatrix} \frac{\partial \tilde{\mathbf{x}}_1}{\partial r} & \frac{\partial \tilde{\mathbf{x}}_1}{\partial s} & \frac{\partial \tilde{\mathbf{x}}_1}{\partial t} \\ \frac{\partial \tilde{\mathbf{x}}_2}{\partial r} & \frac{\partial \tilde{\mathbf{x}}_2}{\partial s} & \frac{\partial \tilde{\mathbf{x}}_2}{\partial t} \\ \frac{\partial \tilde{\mathbf{x}}_3}{\partial r} & \frac{\partial \tilde{\mathbf{x}}_3}{\partial s} & \frac{\partial \tilde{\mathbf{x}}_3}{\partial t} \end{bmatrix}^{-1} \tag{3.10}$$

[5] Open-source software-package like CGAL provide an optimized Delaunay-triangulation-algorithm which can tetrahedralize a point cloud in $\mathcal{O}(n(\log n))$.

Actually computing the partial derivatives results in the deformation gradient in terms of the four vertices:

$$\mathbf{F} = \begin{bmatrix} x_1 - x_4 & x_2 - x_4 & x_3 - x_4 \\ y_1 - y_4 & y_2 - y_4 & y_3 - y_4 \\ z_1 - z_4 & z_2 - z_4 & z_3 - z_4 \end{bmatrix} \begin{bmatrix} u_1 - u_4 & u_2 - u_4 & u_3 - u_4 \\ v_1 - v_4 & v_2 - v_4 & v_3 - v_4 \\ w_1 - w_4 & w_2 - w_4 & w_3 - w_4 \end{bmatrix}^{-1} \tag{3.11}$$

Note that the deformation gradient F is independent of the interpolation weights r, s, t and, thus, constant across the element. This is true of linear interpolation of tetrahedron and triangles. Since the deformation gradient is constant we can avoid the volume integration of the gradient over the elements volume and simply multiply by the analytically computable volume. Having derived the deformation gradient we can compute the greens-strain as in section 2.1.2. In order to incorporate the strain into the PBD framework we create a constraint that tries to reduce the strain to zero. Let $\mathbf{S} = \mathbf{F}^T\mathbf{F}$ then the corresponding stretch an shear constraints are:

$$C_{stretch}(\mathbf{x}) = \mathbf{S}_{ii} - 1 \tag{3.12}$$

$$C_{shear}(\mathbf{x}) = \mathbf{S}_{ij} \tag{3.13}$$

Since the matrix $\mathbf{S}$ is symmetric, section 2.1.2, we only need to consider three shear constraint, for $i < j$. This leaves us with three stretch and three shear constraints for each tetrahedral element.

In order to solve them with the position based method the constraints need to be linearized so that they can be solved in a single step. The stretch constraint can easily be linearized by taking the square root of the respective component:

$$C_{stretch}(\mathbf{x}) = \sqrt{\mathbf{S}_{ii}} - 1 \tag{3.14}$$

Finally to decouple the shear deformation from stretch deformation it needs to be normalized and, thus, made independent of the amount of stretching applied. Giving us the final shear constraint:

$$C_{shear}(\mathbf{x}) = \frac{\mathbf{f}_i \cdot \mathbf{f}_j}{|\mathbf{f}_i|\,|\mathbf{f}_j|} \tag{3.15}$$

where $\mathbf{f}_i$ is the column vector $[\mathbf{F}_{1i}, \mathbf{F}_{2i}, \mathbf{F}_{3i}]^T$ and $\mathbf{f}_j = [\mathbf{F}_{1j}, \mathbf{F}_{2j}, \mathbf{F}_{3j}]^T$. These constraints can now be integrated and simulated into the PBD framework described above.

Energy Minimization

Bender et al. (2014b) deploy a very similar approach but instead of minimizing the strain tensor they compute the elastic potential energy for every element and formulate a constraint to minimize this instead. They also use the greens-strain and then employ the Saint Venant-Kirchhoff model for isotropic materials to derive the elastic energy potentials for every element. While the Saint Venant-Kirchhoff model faithfully captures material properties the actually simulated materials may significantly differ from expected behavior due to the slow convergence of the Gauss-Seidel solver. Further their method also supports plastic deformation by splitting the strain into a plastic and elastic component and adjust the plastic component if the maximal elastic yield threshold is reached or by a user specified plastic flow rate, section 4.2. Finally they check for inverted elements by evaluating the sign of $det(\mathbf{F})$ and apply the singular value decomposition to remove the reflected components of $\mathbf{F}$, as originally proposed by Irving et al. (2004).

3.2.4 Elastic Rods

The previous section was concerned with simulation of solids. This section will cover the special case of a beam or a rod and how these objects can be described and simulated using PBD. A Cosserat rod describes a rod with finite extense by its continuous centerline curve $\mathbf{r}(s) : [0, L] \to \mathbb{R}^3$, where L is the initial length of the rod. Since the curve is a one-dimensional object the definition of infinitesimal strain from chapter 2 can not be applied directly. Instead we will derive a strain measurement of stretch and shear for rods, however, it is clear that a notion of twist can not be expressed only by the position of the centerline. In order to measure twist and shear an orthonormal frame with the *directors* $\{\mathbf{d}_1(s), \mathbf{d}_2(s), \mathbf{d}_3(s)\}$ is defined for each point on the centerline. We will define this frame such that $\mathbf{d}_1$ and $\mathbf{d}_2$ span the cross-section of the rod and $\mathbf{d}_3$ is parallel to the cross-section. This implies that $\mathbf{d}_3$ is parallel to the tangent in rest configuration. This $\mathbf{d}_3$ convention has been named adapted frame by Bergou et al. (2008). In a space with a fixed origin and orthonormal basis vectors $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ the adapted frame can be expressed as rotated basis vectors. This is achieved by quaternion rotation, $\mathbf{d}_k = \mathrm{q}(s)\mathbf{e}_k\bar{\mathrm{q}}(s)$, where $\mathrm{q} \in \mathbb{H}$ is a unit-length quaternion that describes the rotation of the frame and $\mathbf{e}_k$ is embedded into an quaternion to make the product well defined.

Stretch and shear: These can be measured by comparing the initial director $\mathbf{d}_3(s)$ to the tangent in the current configuration $\partial_s\mathbf{r}(s)$.

$$\Gamma(s) \overset{\text{def}}{=} \boldsymbol{\varepsilon}_{\text{rod}}(s) = \partial_s\mathbf{r}(s) - \mathbf{d}_3(s) \tag{3.16}$$

where we named this specific strain measure Γ to make indices more readable. In order for the strain Γ to vanish the tangent needs to be both unit-length and parallel to the initial tangent described by the director. The

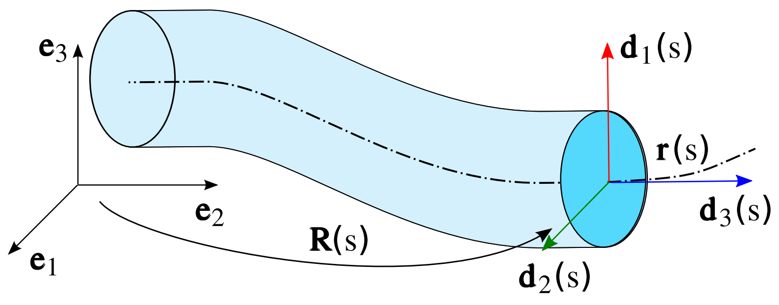


Figure 3.3: A Cosserat rod is described by it's centerline $\mathbf{r}$ and it's orthonormal frame $\{\mathbf{d}_1(s), \mathbf{d}_2(s), \mathbf{d}_3(s)\}$. Reprinted from Kugelstadt et al. 2016

tangent is unit-length if the rod is not stretched and the tangent is parallel to the initial tangent if the rod is not sheared. As the PBD-constraints are solved to vanish we can directly employ this to capture both stretch and shear.

Bend and Twist: Unlike continuous solids for rods we also need to consider the bending and twisting as the whole object is approximated by its centerline. Bending and twisting strain can be measured using the Darbaux vector $\mathbf{\Omega}$, which is defined as:

$$\mathbf{\Omega} = \frac{1}{2}\sum_{k=1}^{3} \mathbf{d}_k \times \mathbf{d}'_k \tag{3.17}$$

where $\mathbf{d}'_k$ is the derivative with respect to the initial curve length s. The Darbaux vector $\mathbf{\Omega}$ describes the rate of change of the directors as s is varied. Further we can express the Darbaux vector in terms of quaternions as:

$$\mathbf{\Omega} = 2\bar{\mathrm{q}}\mathrm{q}' \tag{3.18}$$

the differentiation of the quaternion with respect to s is derived in Kugelstadt et al. (2016).

Discritization of the Rod

Explicitly discretized Cosserat rods have been important for simulating dynamics of thin materials in both computer graphics, Selle et al. (2011) and Teschner (2007), as well as in the engineering comunities Lang et al. (2009, 2011). The specific approach outlined below is a summary of Kugelstadt et al. (2016). A similar approach was proposed by Umetani et al. (2014), however, their approach uses ghost-points instead of quaternions to measure bend and twist. This results in higher computational overhead.

Now that strain has been definied for continuous rods we are going to discretize the rod. Then we can formulate constraints on these particles that minimize both the strain measurements Γ and $\mathbf{\Omega}$. As a staggered grid provides higher accuracy for finite derivatives due to the central-difference, Bridson 2015, it is commonly used in fluid- and rod-dynamics. Here we measure positions on the particle positions and the orientations in between particles or in other words on the rod-elements, fig. 3.4. Thus, we will use half indices to emphasize that a staggered approach is used, e.g. $\mathrm{q}_{i+\frac{1}{2}}$. The actual implementation will, of course, work with integers and a respective offset rather than with floats. A single segment with two particles and an orientation is considered a rod-element. Thus, a curve is discretized into N particles and $N-1$ rod-elements and orientations.

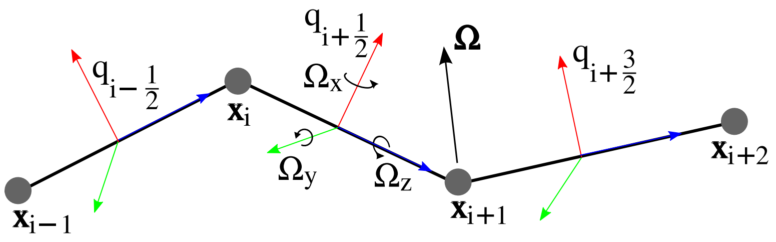

$\mathrm{q}_{\mathrm{i}-\frac{1}{2}}$
$\mathrm{q}_{\mathrm{i}+\frac{1}{2}}$
Ω_x
$\mathbf{\Omega}$
$\mathrm{q}_{\mathrm{i}+\frac{3}{2}}$
$\mathbf{x}_\mathrm{i}$
Ω_y
Ω_z
$\mathbf{x}_{\mathrm{i}+2}$
$\mathbf{x}_{\mathrm{i}-1}$
$\mathbf{x}_{\mathrm{i}+1}$

Figure 3.4: Staggered grid approach for rod-discretization. Reprinted from Kugelstadt et al. 2016

In order to calculate the strain Γ from eq. (3.16) we need both the tangent of the curve and its current frame. The tangent of the curve can be approximated by finite difference: $\partial\mathbf{r}(s) \approx \frac{1}{l}(\mathbf{x}_{i+1} - \mathbf{x}_i)$, where l denotes the initial length of the rod-element. Remember that the director $\mathbf{d}_3$ can be expressed with the orientations and as such the strain of a rod-element can be derived as:

$$\Gamma_{i+\frac{1}{2}} \approx \frac{1}{l}(\mathbf{x}_{i+1} - \mathbf{x}_i) - \Im(\mathrm{q}_{i+\frac{1}{2}}\mathbf{e}_3\mathrm{q}_{i-\frac{1}{2}}) \tag{3.19}$$

where $\Im()$ denotes the imaginary part of the product. This allows us to construct a constraint that minimizes shear and stretch strain similar to the approach in section 3.2.3. Unlike the originally abstracted version of a constraint, which only included all particle positions, this constraint also needs access to the orientations. The shear-stretch-constraint only needs access to the components of a rod-element, thus two positions and an orientation:

$$C_s(\mathbf{x}_1, \mathbf{x}_2, \mathrm{q}) = \frac{1}{l}(\mathbf{x}_{i+1} - \mathbf{x}_i) - \mathbf{R}(\mathrm{q})\mathbf{e}_3 = 0 \tag{3.20}$$

Even though we change the abstract notion of a constraint defined in section 3.2.2 we can still solve the constraint by linearization.

Algorithm 4: Cosserat-rods for position based dynamics

```
1  Time integration velocity of $\mathbf{v}$ and predicted position $\mathbf{b}$
2  forall particles $i$ do $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t m_i \boldsymbol{f}_{\text{ext}}(\mathbf{x}_i)$
3  forall particles $i$ do $\mathbf{b}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
4  Integration angular velocity $\boldsymbol{\omega}$ and predicted orientation $u \in \mathbb{H}$
5  forall orientations $j$ do $\boldsymbol{\omega_j} \leftarrow \boldsymbol{\omega_j} + \Delta t \mathbf{I}_j^{-1}\left(\boldsymbol{\tau_j} - \boldsymbol{\omega}_j \times (\mathbf{I}\boldsymbol{\omega_j})\right)$
6  forall orientations $j$ do
7      $u \leftarrow \mathrm{q}_j + 0.5\Delta t \mathrm{q}_j \boldsymbol{\omega_j}$
8      $u \leftarrow u_j / \|u_j\|$
9  end
10 forall particles $i$ do generateCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{b}_i$)
11 forall solverIteration do
12     projectConstraint($C_1, \ldots, C_{M+M_{coll}}, \mathbf{b}_1, \ldots, \mathbf{b}_N, u_1, \ldots, u_{N-1}$)
13 end
14 Velocity correction
15 forall particles $i$ do
16     $\mathbf{v}_i \leftarrow (\mathbf{b}_i - \mathbf{x}_i)/\Delta t$
17     $\mathbf{x}_i \leftarrow \mathbf{b}_i$
18 end
19 Orientation correction
20 forall particles $i$ do
21     $\boldsymbol{\omega_j} \leftarrow \Im(2\bar{\mathrm{q}}_{\mathrm{j}} u/\Delta t)$
22     $\mathrm{q}_j \leftarrow u$
23 end
```

To define a bend and twist constraint we need to evaluate the orientation at a particle position. Since orientations are only stored in between particles, fig. 3.4, the orientations need to be interpolated onto the particles from the adjacent orientations. The computationally simplest approach is to take the arithmetic mean:

$$\boldsymbol{\Omega} = \frac{1}{l}\Im\left((\bar{\mathrm{q}}_{i+\frac{1}{2}} + \bar{\mathrm{q}}_{i-\frac{1}{2}})(\mathrm{q}_{i+\frac{1}{2}} - \mathrm{q}_{i-\frac{1}{2}})\right) = \frac{2}{l}\Im\left(\bar{\mathrm{q}}_{i-\frac{1}{2}}\mathrm{q}_{i-\frac{1}{2}}\right) \tag{3.21}$$

In the discrete case the Darbaux vector is the imaginary part of the quaternion that realizes the rotation from discrete frame $i-\frac{1}{2}$ to $i+\frac{1}{2}$. Similar to the shear-stretch-constraint we want this constraint to reduce difference to the rest configuration. In this case we want the difference between the current Darbaux vector $\boldsymbol{\Omega}_i$ and the initial Darbaux vector $\boldsymbol{\Omega}_i^0$ to be zero:

$$C_b(\mathrm{q}, u) = \Im(\bar{\mathrm{q}}u - \bar{\mathrm{q}}^0 u^0) = \boldsymbol{\Omega} - s\boldsymbol{\Omega}^0 \tag{3.22}$$

where $\mathrm{q}, u \in \mathbb{H}$. The factor s is introduced to make the choice of the rest orientation unique as $+q$ and $-q$ represents the same rotation. Thus, for

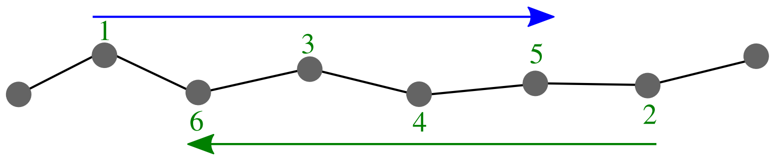


Figure 3.5: Bilateral interleaved constraint ordering. Reprinted from Kugelstadt et al. 2016

every bend constraint there would be two valid rest positions. The system is made unique by choosing s such that:

$$s = \begin{cases} +1 & \text{for} |\mathbf{\Omega} - \mathbf{\Omega}^0|^2 < |\mathbf{\Omega} + \mathbf{\Omega}^0|^2 \\ -1 & \text{for} |\mathbf{\Omega} - \mathbf{\Omega}^0|^2 > |\mathbf{\Omega} + \mathbf{\Omega}^0|^2 \end{cases} \tag{3.23}$$

The gradients of both constraints can be derived as presented in Kugelstadt et al. (2016) and thus, the constraint can be solved with a slightly modified version of the PBD algorithm in section 3.2. It has been extended, algorithm 4, to predict an orientation u based on the angular velocity, lines 4 to 9. The constraint projection, line 12, has been extended to allow constraints to access predicted orientations. And finally the orientations are updated based on the applied correction in lines 19 to 21.

As discussed in section 3.2.2 the ordering of constraints can change the result of the Gauss-Seidel solver. As pointed out in Umetani et al. (2014) solving the rod constraints in sequential order from one end to the other can lead to instabilities. They propose a bilateral interleaved constraint ordering as depicted in fig. 3.5.

3.2.5 Stiff Rods

Deul et al. (2018) employ a similar approach, however, they combine zero-stretch, bending and twisting conditions into a single constraint. In order to incorporate meaningful physical quantities they derive a compliance property α for the constraint as proposed in Macklin et al. (2016). This allows to define Youngs-modulus for bending and a torsion modulus. Usually, the shear-modulus is used to describe the torsion. However, since the constraint enforces zero-stretch and shear is independent of user material parameters notion of torsion-modulus emphasizes its independence from shear. As noted in section 3.2.6 these compliance values are only meaningful if the system can get reasonably close to convergence, which requires very high iteration counts with the non-linear Gauss-Seidel approach. In order to alleviate this issue they use a direct solver to compute the solution to the linear KKT[6]

[6]Karush–Kuhn–Tucker conditions

subproblem. Baraff (1996) describes how to get the perfect elimination order for an LDL^T decomposition. The elimination order only changes when the topology of the object is changed and can thus, be pre-computed before the simulation. This allows to directly [7] solve the system in $\mathcal{O}(n)$. However, this approach only works for acyclic constraints. To solve inequalities and cyclic constraints such as a rope fixed on both ends the usual Gauss-Seidel solver is used. Thus, for all compatible constraints they first compute the λ and the corresponding deltas, apply them to the predictions of position and orientations as well as to update the compliance value. Then collisions and closed looped constraints are solved using the Gauss-Seidel approach. This allows their method to significantly reduce iterations needed even for large error-thresholds thus, making up for the higher cost of the KKT-solver compared to the non-linear Gauss-Seidel.

Algorithm 5: KKT/GS solver

```
1 while  residual error > $\eta$ do
2     forall rods do
3         direct KKT Solver
4     end
5     forall collsion do non-linear Gauss-Seidel
6 end
```

3.2.6 Convergence Rate

While PBD is well suited for simulating soft elastic materials and fluids like jelly, honey or cloth the slow convergence of the Gauss-Seidel solver makes it nearly impossible to simulate stiff materials like wood or metal. Exchanging the solver to a global one, as in Goldenthal et al. (2007) allows good approximation of stiff materials and helps to connect PBD to implicit integration as further discussed in section 3.2.7. The alternative is to introduce constraint-hierarchies. Solving a set of coarse constraints first to propagate changes quickly through the object and then simulate more detailed response with a finer set of constraints. Consider a rod as in fig. 3.6, assuming that we use the approach Cosserat approach from section 3.2.4 the minimal rod that is still bendable contains three particles and three constraints: two stretch-shear-constraints and a bending constraint between the two rod-elements. As such the first hierarchy will correct particles $\{0, 2, 4\}$ in fig. 3.6. The next hierarchy will then work with the corrected position of these particles. Alternatively we may also define the positions of higher hierarchies relative to their parents. These concepts are further explored and generalized to trian-

[7]Direct in opposition to an iterative approach.

gle and tetrahedral meshes by Matthias Müller (2008), Schmitt et al. (2013), and Wang et al. (2010),

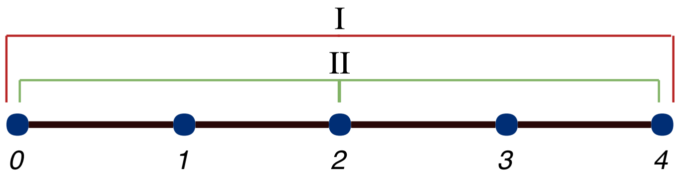


Figure 3.6: The first hierarchy I only corrects particles $\{0, 2, 4\}$ and thereby rapidly propagates corrections through the whole length of the rod. The second hierarchy II then works with the corrected particle positions. This structure is repeated until all particles in the rod are covered.

By considering only fixed cloth, like flags attached to a pole Kim et al. (2012) used long-distance-attachments (LDA) to rapidly propagate corrections from the fixed particles to the rest of the cloth. A similar approach was used in Müller et al. (2012) to simulate in-extensible attached rods, like hair or fur, where the constraints are ordered such that the rod is corrected from the fixed particle onward. This combined with the original Gauss-Seidel solver, which uses the corrected position from the previously solved constraint and thus, guarantees no stretch in a single iteration. As all particles in a rod are corrected based on the movement of the fixed particle in the rod their method is called Follow-The-Leader, fig. 3.7.

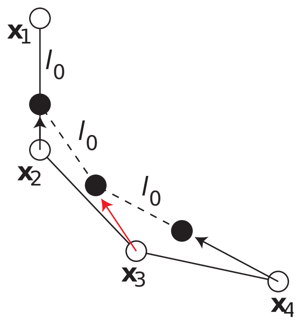


Figure 3.7: The Follow-The-Leader method improves convergence by ordering constraints to be solved from the fixed end to the free. Reprinted from Müller et al. (2012)

Another limitation of PBD discussed so far is that the stiffness of the object depends on the iteration count. It would be preferred if the stiffness would only depend on the user specified stiffness values. This addressed by Macklin et al. (2016) by introducing an compliance multiplier with each constraint. Each calculation of λ can be seen as an incremental change to a total multiplier and thus, effectively limiting how much force a single constraint can apply. While this approach does manage to fix the convergence of the solver to a user specified stiffness value it reduces the convergence rate even further. As most PBD applications run in an interactive environment their iteration count is limited and thus, the solver is stopped before convergence[8] is reached. As such the default PBD implementation described above rarely benefits from this approach.

3.2.7 Projective Dynamics

Projective dynamics, Bouaziz et al. (2014), builds upon the strain based dynamic approach and connects it to implicit Euler integration. Their key insight is that non-linearity of the constraints can be decoupled from the elastic response. This is done by defining a constraint manifold $\boldsymbol{\varepsilon} = 0$. As it is based on strain it is independent of the material response. This allows us to measure the potential energy, i.e. the distance from this manifold in a linear manner while keeping the non-linearity from the constraint manifold itself. As described in section 3.2.3 we can define strain constraints on a discrete set of particles in both two and three dimensions with triangles and tetrahedral, respectively. We can then project the prediction position $\mathbf{x}$ onto this constraint manifold and storing the projection as $\mathbf{b}$. The potential energy can now be solved for as the distance between the position $\mathbf{x}$ and projection $\mathbf{b}$. This results in the minimization of a quadratic system which is equivalent to the solution of a linear system. Assuming the constraints stay constant the global solution to this system can efficiently computed by factorizing the system at start up.

Projective dynamics is similar to section 3.2.5 as it also combines local (projection) and global solvers to improve convergence. Further it shows that PBD is a special case of the well known implicit Euler integration. Projective dynamics is extended by Narain et al. 2016 to improve convergence by applying alternating direction method of the optimization algorithms by Boyd (2010).

3.3 Shape Matching

Unlike previously presented methods shape matching is meshless and as such operates without elements. It allows unconditionally stable simulation of elastic and plastic behavior and is easy to implement. So far the other methods have computed the strain and derived an elastic response from that. Shape matching on the other hand estimates the particle positions as

[8]Or rather before a epsilon criterion with sufficient accuracy would terminate the solver.

if the object would be rigid. By moving the actual positions to these goal-position the elastic response is realized. To discretize the body of interest, points are scattered within its region. As with PBD the particles are first integrated in time by considering external forces to predict new positions. To calculate the rigid-body positions called goal positions $\boldsymbol{g}$ we need the optimal rigid-body transformation. It can be computed from the difference between the current and the material configuration. Then by applying the optimal transformation to all material coordinates $\mathbf{u_i}$ the goal positions $\mathbf{g_i}$ are computed. After each timestep every particle position $\mathbf{p_i}$ is adjusted towards its corresponding goal position $\mathbf{g_i}$. Given these two sets of positions $\mathbf{u_i}$ and $\mathbf{p_i}$ the optimal rotation $\mathbf{R}$ and translation $\mathbf{t}$ between them can be found by minimizing:

$$\sum_i m_i(\mathbf{R}(\mathbf{u}_i - \tilde{\boldsymbol{t}}) + \mathbf{t} - \mathbf{p}_i)^2 \tag{3.24}$$

The optimal translation vectors $\tilde{\boldsymbol{t}}$ and $\boldsymbol{t}$ turn out to be the center of mass in the rest and current configuration, respectively which intuitively makes sense as we are looking for a rigid-body translation.

$$\tilde{\boldsymbol{t}} = \tilde{\mathbf{x}}_{com} = \frac{\sum_i m_i \tilde{\mathbf{x}}_i}{m_i}\ , \quad \boldsymbol{t} = \mathbf{x}_{com} = \frac{\sum_i m_i \mathbf{x}_i}{m_i} \tag{3.25}$$

The center of mass can be used to define all particle positions relative to it, let $\mathbf{q}_i = \tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_{com}$ and let $\mathbf{p}_i = \mathbf{x}_i - \mathbf{x}_{com}$. By minimizing the term $\sum_i(\boldsymbol{A}\mathbf{q}_i - \mathbf{p}_i)^2$ we obtain the optimal transformation $\boldsymbol{A}$ in the least-squares sense. To minimize this function we set all its derivatives with respect to $\boldsymbol{A}$ to zero which yields:

$$\boldsymbol{A} = (\sum_i m_i \mathbf{p}_i \mathbf{q}_i^T)(\sum_i m_i \mathbf{q}_i \mathbf{q}_i^T)^{-1} = \boldsymbol{A}_r \boldsymbol{A}_s \tag{3.26}$$

It is apparent that A_s is symmetric and as such does not contain any rotation. Thus, we can extract the optimal rotation $\mathbf{R}$ from the moment matrix A_r. Note that the moment matrix can be computed directly as:

$$\boldsymbol{A}_r = \sum_i m_i \mathbf{q}\mathbf{q}^T \tag{3.27}$$

Polar decomposition, section 2.1.2, allows us to extract the rotational components from the remaining deformations $\mathbf{A}_r = \boldsymbol{R}\boldsymbol{S}_{\mathbf{qp}}$. The goal position can now be computed as:

$$\boldsymbol{g}_i = \boldsymbol{R}(\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_{com}) + \mathbf{x}_{com} \tag{3.28}$$

With the goal positions for all particles computed a simple distance constraint can be employed to pull the current particles to their respective goal

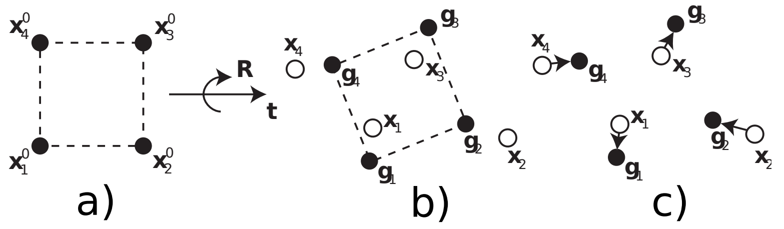


Figure 3.8: The initial configuration with positions $\mathbf{x}_i^0$ is matched to the deformed configuration with an ideal rigid translation t and rotation R. To simulate elastic behavior (or the reduction of strain) the positions $\mathbf{x}_i$ are pulled towards the goal positions g_i. Reprinted from Matthias Müller et al. (2005)

positions. As such this approach can be integrated seamlessly into the PBD-framework.

The method described above is based on Matthias Müller et al. (2005) and as described only allows for small deformations. To allow for larger deformation one can either introduce a projection that does not only depend on $\boldsymbol{R}$ but also on the linear transformation $\boldsymbol{A}$ or define multiple overlapping shape-matching regions. The different shape-matching regions contain a set of particles on which the shape-matching is applied, overlap between regions connects them to each other. The amount of overlap determines the stiffness but also significantly increases computational overhead. Matthias Müller et al. (2008), introduced a fast summation technique for regular grids to reduce the redundant computations. Doug et al. (2007), extends this grid-summation approach to allow for irregular shape combinations which allows for inhomogenius materials and also makes handling of boundary conditions more efficient. Diziol et al. (2011) build upon the fast summation approach. In contrast to previous methods they only use the surface mesh of a volumetric object to compute goal-positions.

3.3.1 Oriented Particles

While shape-matching works well for evenly distributed particle sets the matrix $\mathbf{A}_r$ eq. (3.26) becomes ill-conditioned for co-planar particle sets, which makes simulation of thin materials like sheets and rods difficult. Matthias Müller et al. (2011) addresses this issue by storing an orientation information along with the position for each particle. This allows us to reformulate the computation of the optimal rotation such that it is also stable for co-planar particles and even shape matching groups with only a single particle. To compute the moment matrix $\boldsymbol{A}_i$ for each particle i we associate a small sphere with each particle. By definition of the moment matrix is then given by:

$$\begin{aligned}\boldsymbol{A}_i &= \int_{V_r} \rho \boldsymbol{R} \mathbf{x}\mathbf{x}^T dV_r = \rho \boldsymbol{R} \int_{V_r} \mathbf{x}\mathbf{x}^T dV_r \\ &= \rho \boldsymbol{R} \int_0^{2\pi} \int_0^{\pi} \int_0^{r} (r sin \vartheta)^2 r^2 dr \, d\vartheta \, d\varphi \\ &= \frac{4}{15}\pi r^5 \rho \boldsymbol{R} = \frac{4}{15}\pi r^5 \frac{m}{V_r} \boldsymbol{R} = \frac{1}{5} m r^2 \boldsymbol{R}\end{aligned} \tag{3.29}$$

where V_r is the volume of a sphere with radius r. Having defined the moment matrix for every particle we can proceed to combine them to a moment matrix for a set of particles. However, as each moment matrix A_i has been defined with its own particle position as center we need to adjust the individual moment matrices to fit the center of mass of the object, as described in Doug et al. (2007).

$$\boldsymbol{A}_r = \sum_i \left(\boldsymbol{A}_i + m_i \mathbf{x}_i \tilde{\mathbf{x}}_i^T \right) - \boldsymbol{M} \mathbf{x}_{com} \tilde{\mathbf{x}}_{com}^T \tag{3.30}$$

This allows us to compute the momentum matrices for different shape-matching regions while making use of the orientation information. Jones et al. (2015) describe an efficient algorithm to cluster particles and perform collision checks on them and as such is well suited for creating shape-matching groups.

3.3.2 Related Methods

Instead of defining goal positions, Sorkine et al. (2007), derive a rigidity measurement which defines a linear equation system that can be solved directly. As they optimize for rigidity their approach is called as-rigid-as-possible. Choi et al. (2018), extend oriented particle to allow for faster convergences with a partial global solver similar to projective dynamics in the case of PBD. Another meshless method is based on a sparse discretization of the body with frames that carry both position and orientation information similarly to oriented particles. Unlike oriented particles and other PBD methods they actually derive the elastic energy potentials and integrate them similar to an FEM simulation. The resulting deformations are then applied to the visual representation with dual-quaternion skinning. This method was introduced by Gilles et al. (2011) and allows interactive frame rate with very sparse sampling of the object. A unfied solver for rods, sheets and solids was proposed by Martin et al. (2010) which also operates on frames, however requires a denser sampling. Finally Faure et al. (2011) extend this formulation for heterogeneous materials.

Chapter 4

Fracture

The approaches in the previous chapter only considered elastic deformation this will be extended here to include plastic deformation as well. Plastic deformation is required to model ductile fracture, without it only brittle fracture can be simulation. For this purpose we will define a elastic yield criterion and a fracture condition which represents the maximum strength material can withstand. These can be integrated into the methods discussed in the previous chapter and is exemplified on oriented particles. For more complete overview of fracture methods we will start off with a short discussion of non-physical fracturing.

4.1 Non-Physical Fracture

Muguercia et al. (2014) categorize interactive fracturing into either non-physical methods such as image-based or procedural fractures and physical based fracturing. Non-physical methods are popular in video games as they have a guaranteed run time cost that is usually fairly low. The most common method is to pre-fracture the geometry. In this method the whole body is first decomposed into small components. Fracturing then separates these components from each other but no additional splits are introduced. This allows for fine grained artistic control and the decomposition can be done in advance. The connected object can be simulated as a rigid body until a fracture condition is met. As the fracture pieces are decided before the simulation started it can not respond dynamically to user input. Matthias Müller et al. (2013) propose a convex decomposition to dynamically apply a fracture pattern to the geometry based on the impact position. This decomposes the object dynamically into a set of convex pieces. Since all the fractured pieces are convex the rigid-body simulation and collision response can be computed for a large number of bodies in real-time. A sightly more

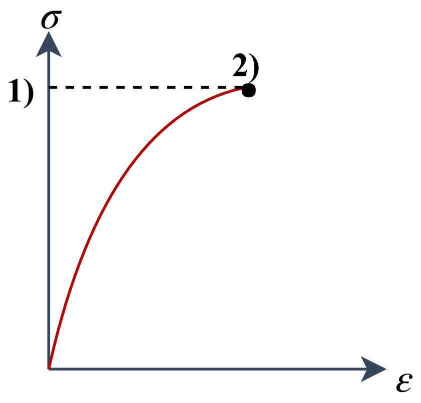


Figure 4.1:
1 Ultimate strength
2 Material failure/fracture

Reprinted from Richfield 2009

sophisticated approach is to connect the individual parts of the object with glue-constraints. These constraints can break under tension, however this approach requires to dynamically detect islands that then need to be simulated as rigid bodies. This method is commonly used in software packages such as Blender or Houdini due to its high versatility, artistic control and ease of use.

4.2 Plastic Deformations

So far we have only considered the elastic response of each method. Which means that if the load is removed the object returns to its initial configuration. However, not all materials respond only in an elastic manner to strain. Some also exhibit plastic deformation. Unlike elastic deformation plastic deformation dissipates the load as heat and actual material flow resulting in a permanent deformation. Materials that only exhibit elastic behavior before their failure are considered brittle. Examples of brittle materials are glass, certain types of steel or ceramics. One distinctive feature of brittle fracture is that once fracturing occurs the fracture front continues to propagate through the material even if the load has been removed. Brittle fracture also tends to propagate at high velocities making them instant for most choices of time-step. We have and continue to assume that the elastic response is linear to strain, however, as can be seen in fig. 4.1 this is not necessarily the case.

Other materials like aluminum or structural steel exhibit plastic deformation before they fracture. Consider the stress-strain curve in fig. 4.2, note that only small strains yield elastic response, however, here the elastic response is nearly linear. As plastic deformation changes the structure of the material it also changes the rest-configuration that the elastic response will correct towards. This also has an effect on how the strain/stress response is measured.

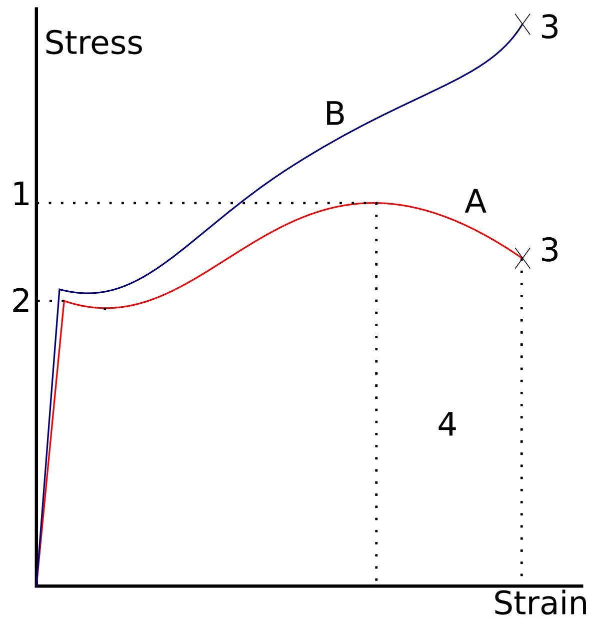


Figure 4.2:

1	Ultimate strength
2	Elastic limit/yield point
3	Material failure/fracture
4	Necking region
A	Apparent stress $\frac{F}{A^0}$
B	True stress $\frac{F}{A}$

Reprinted from Richfield 2009

As such the cross-section of the object changes during plastic deformation. The apparent stress $\frac{F}{A^0}$ that is measured with respect to the initial cross-section, thus, does not account for plastic deformations. The true stress $\frac{F}{A}$ is difficult to measure since the cross section changes non uniformly. Necking region is the region where the apparent stress of the object drops due the reduction of the cross-section. The plastic deformation further depends on the duration of loading which is measured by the plastic flow rate. Unlike brittle materials, materials with strong plastic deformation exhibit ductile fracture. Here the material is pulled apart rather than cracked. The crack propagation is slow when compared to brittle material and induces further plastic deformations to the materials. In contrast to brittle fracture the crack does not propagate through the material if the load is not increased. Thus, in order to model ductile fracture the methods above need to be extended to handle plastic deformations.

4.2.1 Yield Criterion

Strain itself is a pure geometric measurement of non-rigid deformations. By splitting the total strain ε into an elastic ε^e and a plastic ε^p components we can introduce plastic deformation into any of the strain based dynamic approaches. In chapter 3 we introduced a number of methods to compute the elastic response of the material, to introduce plasticity into these models the response needs to be computed based on the elastic strain rather than the total strain. So any non-rigid deformation will first contribute to elastic strain and once the elastic limit is reached additional deformation will contribute towards plastic strain. This requires to define the elastic limit. O'brien et al. (2002) proposed to employ the *Mises yield criterion* to define the elastic limit and is based on the deviation of elastic strain:

$$\boldsymbol{\varepsilon}' = \boldsymbol{\varepsilon}^e - \frac{\mathrm{Tr}(\boldsymbol{\varepsilon}^e)}{3}\boldsymbol{I} \tag{4.1}$$

where Tr() is the trace of a matrix. By subtracting the average of the trace we remove uniform scaling from the strain. Making the plastic deformation independent of uniform stretch, prevents the object from changing its volume. We can then compute the elastic limit for a material constant γ_1 as:

$$\gamma_1 < \|\boldsymbol{\varepsilon}'\| \tag{4.2}$$

where $\|\cdot\|$ is the Frobenius norm. Equation (4.2) defines Mises yield criterion as in Y. C. Fung 1965. Next we can extract the plastic component of the strain by:

$$\Delta\boldsymbol{\varepsilon}^p = \frac{\|\boldsymbol{\varepsilon}'\| - \gamma_1}{\|\boldsymbol{\varepsilon}'\|}\boldsymbol{\varepsilon}' \tag{4.3}$$

This allows us to update plastic deformation in every time-step as $\boldsymbol{\varepsilon}^e = \boldsymbol{\varepsilon}^p + \Delta\boldsymbol{\varepsilon}^p$.

4.2.2 Fracture Criterion

The previous section 4.2.1 gave us a simple way to compute the elastic limit on the stress-strain-curve. This allows to split strain into an elastic and plastic component. Plastic deformation or the total deformation also has a limit. This limit describes how much stress the material can withstand before it fractures. As with elastic limit a simple approach to determine the material fracture point is needed. A popular approach is to decompose the stress matrix $\boldsymbol{\sigma}$ into its eigenvalues $\lambda_i(\boldsymbol{\sigma})$ and eigenvectors $\varrho_i(\boldsymbol{\sigma})$ called principal stresses. Then the user can set a uniaxial-material strength s which is then compared to the largest stress eigenvalue $\max_i(\lambda_i(\boldsymbol{\sigma})) > s$ to decide if a fracture occurs. Then the fracture plane is set to be perpendicular to the corresponding eigenvector $\varrho_i(\boldsymbol{\sigma})$. Depending on the simulation method we can then proceed to split or remove the elements or constraints that pass through this fracture plane.

A slightly more complex, yet more accurate fracture criterion is defined in O'brien et al. (1999). They propose to decompose the stress into compressive $\boldsymbol{\sigma}^-$ and tensile components $\boldsymbol{\sigma}^+$. This is also based on the eigenvalue decomposition as :

$$\boldsymbol{\sigma}^+ = \sum_i^3 \max_i(0, \lambda_i(\boldsymbol{\sigma}))\boldsymbol{m}(\varrho_i(\boldsymbol{\sigma})) \tag{4.4}$$

$$\boldsymbol{\sigma}^{-} = \sum_{i}^{3} \min_{i}(0, \lambda_i(\boldsymbol{\sigma}))\boldsymbol{m}(\varrho_i(\boldsymbol{\sigma})) \tag{4.5}$$

where $\boldsymbol{m}(\cdot)$ is a matrix that collects the principal stress as:

$$\boldsymbol{m}(\boldsymbol{a}) = \begin{cases} \boldsymbol{a}\boldsymbol{a}^T/|\boldsymbol{a}| & \boldsymbol{a} \neq \mathbf{0} \\ \mathbf{0} & \boldsymbol{a} = \mathbf{0} \end{cases} \tag{4.6}$$

Note that $\boldsymbol{m}$ has $|\boldsymbol{a}|$ as eigenvalues and $\boldsymbol{a}$ as eigenvectors. By polar decomposition we can compute the highest tensile and compressive principal direction and value on the tensile and compressive matrices, respectively. As before a simple threshold criterion can now be applied to the maximum value.

4.3 Oriented Particle Fracture

In this section we will summarize how plastic deformation and a fracture criterion can be integrated into the oriented particle approach. This integration is based on Choi (2014). As we have seen in section 3.3 shape-matching does not compute the strain thus, the Mises yield criterion can not be applied directly to determine the elastic limit. First we must derive an optimal stretch measurement which we will use instead of strain to determine material failure.The optimal stretch can be derived similarly to the optimal rotation. If we look at the strain derivation with polar decomposition in section 2.1.2 we notice that the linear transformation can be decomposed into a rotational matrix and an additional part that contains the stretch. We already have the rotation $\boldsymbol{R}$ computed with the shape-matching approach, therefore we can proceed to formulate the optimal stretch based on the now fixed rotation by minimizing:

$$\sum_{i} m_i \|\boldsymbol{R}\boldsymbol{S}\mathbf{q}_i - \mathbf{p}_i^T\|^2 \tag{4.7}$$

where $\mathbf{q}_i = \tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_{com}$ and $\mathbf{p}_i = \mathbf{x}_i - \mathbf{x}_{com}$ as in section 3.3. By introducing an additional constraint that ensures that the stretch-matrix $\boldsymbol{S}$ is symmetric to the above equation Choi (2014) derived the optimal stretch as:

$$\boldsymbol{S} = \varrho(\boldsymbol{A}_s)\left[\left(\varrho(\boldsymbol{A}_s)^T \lambda(\boldsymbol{S}_{\mathbf{pq}})\varrho(\boldsymbol{A}_s)\right)\right) \circ \Lambda\right] \varrho(\boldsymbol{A}_s)^T \tag{4.8}$$

where $\boldsymbol{A}_s = (\sum_i m_i \mathbf{q}\mathbf{q}^T)$ as defined in eq. (3.26), $\circ$ is the Hadamard product that produces the element-wise multiplication of its operand matrices and $\Lambda ij = 2/\left(\lambda_i(\boldsymbol{A}_s) + \lambda_j(\boldsymbol{A}_s)\right)$. By using eigenvalue decomposition we can define the Greens-strain based on this optimal stretch as $\varepsilon_G = \frac{1}{2}\boldsymbol{\varrho}(\boldsymbol{\lambda}^2 - \boldsymbol{I})\boldsymbol{\varrho}^T$. Now we can use Mises yield criterion or another yield criterion to separate the total strain into an elastic and a plastic component. In our case we are interested in the plastic deformation rather than the plastic strain, this can be computed by applying the yield criterion to the stretch matrix we just computed. Given the plastic deformation $\mathbf{D}^p$ we replace the relative rest position with: $\mathbf{q}_i = \mathbf{D}^p(\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_{com})$ to apply the plastic deformation to the calculation of the goal position.

Chapter 5

Composite Material

As touched on in the introduction our main idea is to combine a solid simulation with a rod simulation into a composite material. Combintation of solid and rod simulations is not a new idea, Liu et al. (2012) propose a force based method that allows artists to paint in fibers into a material to define its response in that direction. More recently Cai et al. (2016) combined the fibers directly into the constitutive model of the solid simulation. This allows them to reduce computational expense as they do not need to compute the elastic response of the rod model. All these methods have in common that they are unable to simulate full anisotrophy. Rather they define a fiber-direction that has a different, usually higher, stiffness than to the orthogonal plane and, therefore, it is called transverse isotrophy. Our method will adopt the former methodology and simulate a rod and a solid in sequence, effectively summing their respective elastic correction. The idea of rod-solid-composite materials is not new for the computer-graphic community. However, to our knowledge this thesis is the first to formulate this kind of material for PBD-solver.

Due to its ease of implementation, computational efficiency and in part due to previous experience with PBD we chose to test the approach with a PBD-solver. The original PBD-approach has been extended to simulate rods, sheets and solids as discussed and demonstrated in section 3.2. Clavet et al. (2005) showed that PBD combines well with ideas from SPH to simulate both viscous and non viscous fluids which was further developed by Macklin et al. (2013) and Takahashi et al. (2016). As such we can consider PBD to be a unified solver. By basing our method on PBD it can easily be extended to interact with any kind of material. Further we will use the Cosserat approach presented by Kugelstadt et al. (2016). With these two options fixed we will discuss first how to discretize the objects for the rod simulation and then proceed with the choice and coupling of solid simulation.

5.1 Rod Configuration

Before the choice of the solid simulation is discussed we will consider the rod discretization. This will yield a set of particles which can be used and, if necessary, extended by the solid simulation.

5.1.1 Central Continuous Rod

For simulating whole tree structures it is common to approximate their branches as rods and simulate the trees behavior solely based on rod-dynamics, as done by Pirk et al. (2014, 2017) . In case of tree-dynamics, the rod-simulation has multiple advantages, first and most obviously, only a fraction of the particles is needed to approximate the branches. This will significantly reduce computation. For trees the interaction of roots with the soil is usually not visible and thus neglected. For this reason there is only a single fixed point in the tree-graph. This allows to employ approaches like Follow-The-Leader or Long-Range-Attachment as covered in section 3.2.6. Both these methods work if and only if there is a single fixed point for each graph. Follow-The-Leader biases the direction corrections propagate by ordering and directly applying the deltas such that subsequent constraints operate with the already corrected predictions. However, the single-fixed point assumption falls apart as soon as we apply a significant load to another particle in the graph. The collision forces the corresponding particles to certain positions thus, fixing them violating the initial assumption. Hence, these approaches fail instead we employ hierarchical constraints to increase convergence. As we saw in section 3.2.4 we can calculate stretch, shear, bending and twisting strain along the rod. Based on these strain-measurements we can not only describe the elastic response but also formulate a fracture-criterion. However, as the center line represents the whole material rather than approximate the specific region around a particle, as with FEM or SPH, we can only tell that the rod is going to fracture and that it's going to fracture in a radial, longitudinal or tangential direction. Yet, we will not be able to identify where the fracture starts or how it is going to propagate through the material, fig. 5.1.

It might be possible to employ a heuristic that captures the fracture pattern of a specific material. However, in order to compute a physical based fracture response we need to fully discretize the rod and employ a solid simulation, chapter 3, on these particles, fig. 5.2a. The particular choice of solid-simulation method will be discussed later in this chapter, for now we will just assume that we can evaluate the strain and elastic response at the discrete particle positions. With these strain measurements we can measure where exactly and in what direction the rod will fracture. The pur-

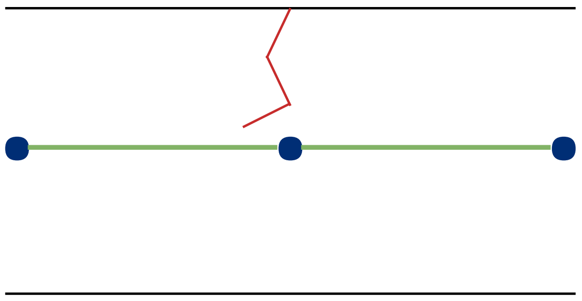


Figure 5.1: We can simulate the dynamics of a branch (in black) with a single centered rod (in green) and even formulate a fracture criterion. However, we only know how much deformation the rod can handle, but not where and in what direction the fracture (in red) will propagate.

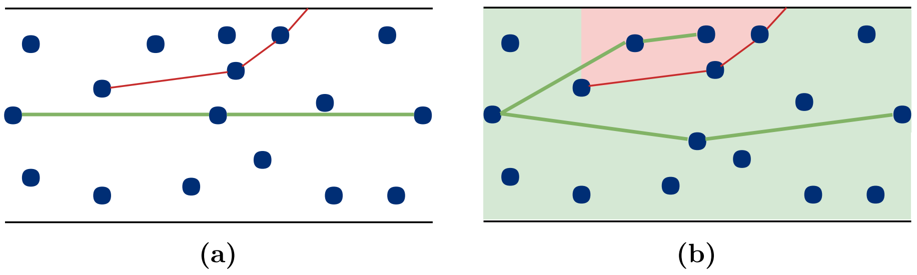


Figure 5.2: By adding additional particles we are able to measure the strain within the object. This allows us to determine a fracture direction as seen in subfigure **a**. Fracture parallel to the center line as they are common in anisotropic materials need their own center line as the elastic response is now independent of the previous center line. The stiffness of the center line would need to be adjusted depending on how the cylinder is fractured.

pose of the central rod configuration is to reduce the computational cost over a complete discretization. In order to keep this benefit we calculate the main elastic response of the whole object based on the central rod. As we already have quaternion computed for every rod-particle[1] we can apply the rod-corrections to their surrounding particles with dual-quaterion skinning. Then we can evaluate the local response with only a few Gauss-Seidel iterations. In order to summarize local responses back to the central rod one could employ shape-matching groups as discussed in more detail in section 5.2.3. In most cases we might not even need to evaluate the local-strains as the strain on the central-rod is small enough that a fracture can not occur.

[1]We assume that we are using the Cosserat model discussed in section 3.2.4.

The real issue occurs when we consider how the fracture needs to change the dynamic response of the system. It is obvious that a fracture as in fig. 5.1 needs to reduce the stiffness of the central rod yet, it is not apparent how the stiffness will change. We could approximate the stiffness reduction by computing the minimum relative cross-section of the still connected part, the minimum of the green area in fig. 5.2b. As the dynamics of the system depend on the center rod we would also need to add an additional rod to the fractured part. Changing the resolution during the simulation is a challenge as the total-dynamics should not vary due to the change in resolution. Though in this case small error may be masked due to the violent nature of a fracture. Adaptive resolution poses many more challenges first it needs to be evaluated when to change the resolution and then prevent the system from recursively refining the discretization. As the resolution changes are dynamic it becomes difficult or even impossible to predict the computational cost of the system making this ill suited for video games with a strict cost budget. Perhaps the largest drawback of this approach would be the limitation to cylindrical objects or objects composed of connected cylinders, like trees.

5.1.2 Parallel Continuous Rods

In the central rod approach the fracture and the dynamics are computed on different domains making the coupling between them difficult. By using many parallel rods as proposed by Sutherland (2012) the fracturing is computed in the same domain as the dynamics fig. 5.3. In this approach the rods are continued through the object making its way back to a fixed region. To connect the rods to each other we can employ any solid simulation on the already existing rod particles. However, it is unclear how to address junctions or even the simple reduction of diameter as common among tree-branches. As seen in fig. 5.3b the naive approach would lead to very high particle densities in thin branches and coarse sampling for the trunk. To address this the number of rods needs to depend on the area of the cross section. To maintain continuous rods this implies that we need to merge and combine rods. This is not addressed by Sutherland (2012) as they assume perfectly cylindrical objects.

5.1.3 Scattered Rod Segments

Finally we will present the rod-configuration that solves most of the issues of the previous configurations though also increasing computational cost. Instead of using continuous rods that run through the whole length of the branch we only employ shorter rod-segments that are not fixed at any point,

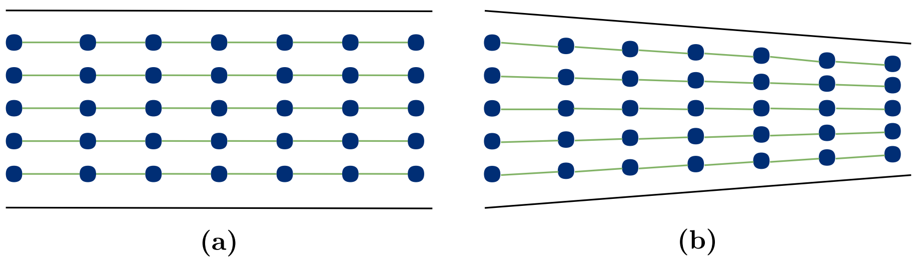


Figure 5.3: Parallel rods

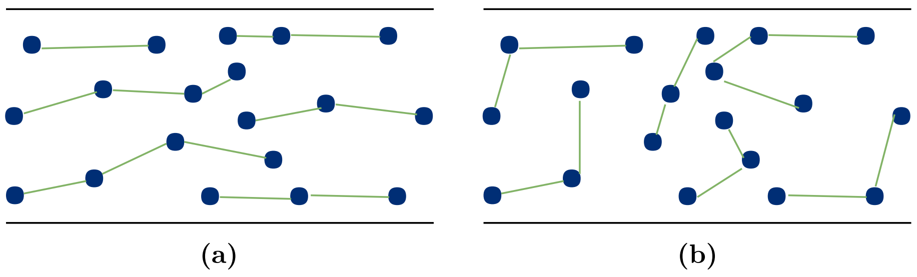


Figure 5.4: Using many short rods

fig. 5.4a. The rods depend on the solid simulation that operates on the rod-particles to keep the rods connected as one object until it fractures. This configuration is very similar to microscopic observations of timber as we noted in the introduction. The rods will give additional strength and help to propagate corrections. By aligning the rods with the imaginary central axis we get anisotropic behavior. We could also randomize the direction of the rods, fig. 5.4b, to get a behavior similar to presswood.

When we fracture this material we can define a fracture criterion for both, the rods and the solid simulation. Then we can proceed with the fracture as if we would compute them in isolation. For example, when a rod fractures at a certain position there will be less correction in that area thus more strain in the solid simulation making it more likely that the solid simulation will also fracture in that region. The same is true for the opposite fracture order. This works since both the rod and solid simulation work on the same position information and so we get implicit fracture coupling. Further as the fracture plane can run through rods we also get the correct dynamic response by just removing a few rod-constraints.

Using rod-segments also allows to simulate arbitrary shapes. By specifying a direction of anisotropy, in case of timber it is the fiber direction, we can make any shape anisotropic.

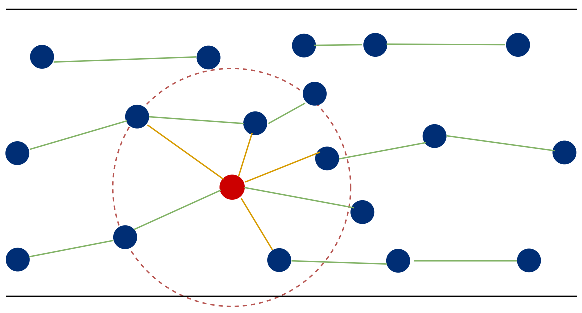


Figure 5.5: For every particle the particles within a radius r that are not part of the same rod-segment are connected by distance constraints. These constraints are then weighted in an SPH fashion using a kernel function that receives the relative distance $h = d/r$ as an argument. Cosserat constraint in green, distance constraint in orange, current particle and it's radius in red.

5.2 Solid Configuration

Here we discuss the choice of solid simulation that connects the scattered cossarat rods to a single object. In order to couple it with the rods the solid simulation needs to be PBD based and provide some sort of orientation information.

5.2.1 Strain Based Dynamics

The first approach we tried was using the strain based dynamics from section 3.2.3. Before the object can be simulated it needs to be decomposed into tetrahedral elements. This decomposition can yield ill-shaped elements especially for a tree-model, since it includes thin branches which are impossible to capture with a coarse tetrahedron decomposition. Where as a fine grained decomposition does introduce too much computational expense. However the approach does provide us with strain measure for every element. When working with strain directly as is the case here we could even employ an anisotropic elastic response. As we do not use a constitutive strain stress model we only require 3 stretch and 3 shear parameters to capture the anisotropy. As with all mesh-based approaches large deformations can result in element inversion, as covered in section 3.2.3. To couple the approach to the rods the orientation for each tetrahedral-element needs to be computed and interpolated onto the corresponding rod-elements.

5.2.2 Distance Constraints

Next we tried simple distance constraints between particles of different rods. For every particle we must first find all its neighbors within a certain radius r that are not connected by Cosserat constraints. As we are simulating solids rather than fluids the neighborhood of each particle does not change until a fracture occurs. This means that we only have to do the neighborhood search once at the start of the simulation. Even though a distance constraint does not consider angles if a particle is connected to other rods by at least three unique distance constraints, it will correct itself to the initial configuration.

While we do avoid the decomposition that plagued the strain based approach we can not properly couple the distance constraints with the Cosserat constraints. As distance constraints only change the position of the particles, but not the orientation of the rods, we are unable to correct for torsion within the object.

5.2.3 Oriented Particles Coupling

The previous approach is already very similar to oriented particles. It is also mesh-less and corrects particles within a certain radius. Oriented particles extend this approach by adding an orientation attribute to each particle. This allows us to couple the solid portion of the simulation to both the strain-shear and the bend-twist constraints. To couple oriented particle corrections to the rods they need to (1) operate on the same particle, fig. 5.6 and (2) operate on the same attribute for position and orientation. For simplicity we define the shape-matching groups for every particle. Remember that the Cosserat approach used a staggered grid to store orientations. Oriented particles on the other hand store the orientation directly within the particle. As both approaches work on the same position information these are implicitly coupled as the rod projections will already be applied when we start with the shape matching. First we apply the Cosserat projection step, line 10 which will update the predicted particle position and the predicted orientation on the rod-elements. For the oriented particle update we need the current orientation on the particles themselves rather then on the rod-elements. Spherical linear interpolation (slrep) allows us to average the rod-element orientations onto the particles, line 11. In line 12 we perform the shape-matching for each group, compute the goal positions and apply the correction to each particle, as described in section 3.3.1. This also update the orientation on the particles. To ensure that the Cosserat correction in the next solver-iteration works with the current orientations we need to interpolate the orientations back onto the rods, line 13. By using a delta

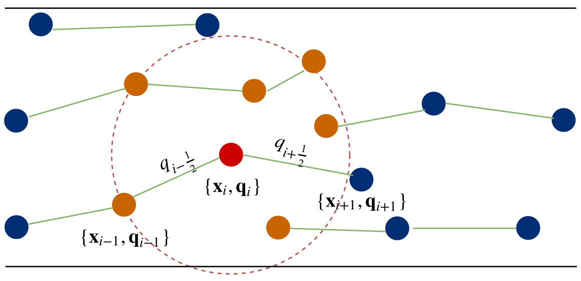

$q_{i-\frac{1}{2}}$
$q_{i+\frac{1}{2}}$
$\{\mathbf{x}_i, \mathbf{q}_i\}$
$\{\mathbf{x}_{i+1}, \mathbf{q}_{i+1}\}$
$\{\mathbf{x}_{i-1}, \mathbf{q}_{i-1}\}$

Figure 5.6: The red particle forms with the orange particles that are within a radius r around it a shape-matching group. The oriented particles store position and orientation information on the particles. The Cosserat rods, drawn in green, instead store the orientations on the rod-elements as noted with the half indices.

summation for the constraint projection we can run the projection step concurrently. The deltas need to be applied before we can run the oriented particle update. As such this behaves like a Gauss-Seidel approach between the different methods but as a Jacobi solver within each method. Thus the ordering of the methods matters. As such we should apply collision constraints at the last correction to ensure that it is fully satisfied and ensure that other updates can not correct the particles back into a collision, line 19.

The remaining code is equivalent to algorithm 4 and left in to make the code more readable and independent.

Algorithm 6: Oriented Particle to Cosserat Rod Coupling

```
forall particles $i$ do $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t m_i \boldsymbol{f}_{\text{ext}}(\mathbf{x}_i)$
forall particles $i$ do $\mathbf{b}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
forall orientations $j$ do $\boldsymbol{\omega_j} \leftarrow \boldsymbol{\omega_j} + \Delta t \mathbf{I}_j^{-1} (\boldsymbol{\tau_j} - \boldsymbol{\omega}_j \times (\mathbf{I}\boldsymbol{\omega_j}))$
forall orientations $j$ do
    $u \leftarrow \mathrm{q}_j + 0.5\Delta t \mathrm{q}_j \boldsymbol{\omega_j}$
    $u \leftarrow u_j / \|u_j\|$
end
forall particles $i$ do generateCollisionConstraints$(\mathbf{x}_i \rightarrow \mathbf{b}_i)$
forall solverIteration do
    projectConstraint$(C_1, \ldots, C_{M+M_{coll}}, \mathbf{b}_1, \ldots, \mathbf{b}_N, u_1, \ldots, u_{n-1})$
    forall particles $i$ do $u_i \leftarrow \text{slerp}(u_{i-\frac{1}{2}}, u_{i+\frac{1}{2}})$
    orientedParticleCorrection$(\mathbf{b}_1, \ldots, \mathbf{b}_n, u_1, \ldots, u_{n-1})$
    forall particles $i$ where $i > 0$ do $u_{i-\frac{1}{2}} \leftarrow \text{slerp}(u_{i-1}, u_i)$
end
  Velocity correction
forall particles $i$ do
    $\mathbf{v}_i \leftarrow (\mathbf{b}_i - \mathbf{x}_i)/\Delta t$
    $\mathbf{x}_i \leftarrow \mathbf{b}_i$
    projectCollisionConstraint$(C_1, \ldots, C_{M+M_{coll}}, \mathbf{b}_1, \ldots, \mathbf{b}_N, u_1, \ldots, u_{n-1})$
end
```

5.3 Compared to Sutherland

To our knowledge the only other publication within the computer graphics community that specifically covers anisotrophic fracture is Sutherland (2012). They assume cylindrical objects which are discretized by parallel rods. As discussed in section 5.1.2 this rod configuration is limited to cylindrical shaped objects. Rods are simulated by computing elastic bending and stretching potentials based on the Kirchoff-model, the torque potentials are not considered. The inter-rod connection is realized by string forces. As such the object is only represented by its discrete points making it difficult to derive consistent material properties and to generate a fracture surface. Our approach solves this problem by connecting the rods with a solid simulation thus, providing a continuum approximation and an element based fracture surface. Using a global solver simulation results are more compliant with material properties. However, force based methods are prone to overshooting due to integration. Overall our method significantly reduces computational costs, is independent of the objects shape, however, reducing material parameter compliance.

Chapter 6

Implementation

The majority of the implementation is written with C++. The solver and projection steps are implemented as described in the previous chapters and parallelised using OpenMp. The project is built with the CMAKE utility and has been configured to compile on both Windows and Linux. This gives us the option to easily switch to a different compiler such as PGI-compiler which supports the latest OpenAcc version. OpenAcc like OpenMp uses pre-processor directives to create kernel programs usually from for loops which can then be executed concurrently. Unlike OpenMp OpenAcc generates CUDA-binaries and as such they can be executed on the graphics processor instead of the CPU.

6.1 Model Handeling

For any set of particle we can generate shape-matching groups based on proximity and coverage, or simply create a shape-matching group per particle. However, in order to place the rods we need a direction of anisotrophy defined for each particle. We do this by manually inserting a spine into the material. This spine can either be a line or a tree-graph which means that every point on the line except the root has a parent. For any particle we can now define its direction of anisotrophy as the average direction of the spine within a certain radius r. For a particle at the position $\mathbf{x}$ we define its relative distance κ_i to a spine-particle i as:

$$\kappa_i = \frac{\|\mathbf{x} - \mathbf{x}_i\|_2}{r} \tag{6.1}$$

Based on the relative distance we can define how much weight w the direction that is associated with this particular particle should carry. We define weight as $w_i = \boldsymbol{K}(\kappa_i)$, where $\boldsymbol{K}$ is a kernel function, currently we use a simple cubic kernel function $\boldsymbol{K}(\kappa) = (1-\kappa)^3$. The direction of the spine-particle i is linearly approximated by subtracting the position of the parent particle, $par(i)$:

$$\boldsymbol{d}_i = w_i \frac{(\mathbf{x}_i - \mathbf{x}_{par(i)})}{\|(\mathbf{x}_i - \mathbf{x}_{par(i)})\|_2} \tag{6.2}$$

Then we can define the anisotrophy direction of any particle as the average of the weighted direction of all spine-particles within the radius r:

$$\boldsymbol{d}(\mathbf{x}) = \frac{1}{|\boldsymbol{S}|} \sum_{i \in \boldsymbol{S}} \boldsymbol{d}_i \tag{6.3}$$

where $\boldsymbol{S}$ is the set of all spine-particles that are within a radius, $\boldsymbol{S} = \{i : \|\mathbf{x}_i - \mathbf{x}\| < r \wedge p_i \in \mathbb{S}\}$ and $\mathbb{S}$ is the set of all spine-particles. Of course, this requires that a spine particle is within the radius of every object-particle.

We load and compute the anisotrophy direction along with other properties such as particle-mass, normals, tangents and bitangents and fixed-position-flag in Houdini. Some objects like tree-models often already include spine information thus we can set up a template-project that makes use of this information. Then during runtime we can use the Python-API to load different objects directly from our application. For objects that do not already provide spine information the user needs to edit this object once. In order to call into Pyhton code from C++we utilize the default Python bindings for C++to start a Python interpreter when we need it. This allows us not only to load different objects directly from within our application but also make use of Houdinis vast function library. For example, we can create and query signed distance fields then change the sampling density dynamically without adding any library dependencies into C++.

6.2 Rod and Particle Group Placement

Currently we employ a simple greedy strategy to place the rods along the previously defined anisotrophy direction. As can be seen in the screen-shots, fig. 6.2a, this creates rod-structure that covers only about 80% of all the particles. Due to the large radii of the shape-matching the rods are still held together however a higher rod-coverage might further improve stiffness and other placement strategies should be investigated.

For the oriented-particle we create a shape-matching group for every particle. This yield to significant amount of redundant computation which can be reduced by fast-summation techniques as described in section 3.3.2. Another approach is create more efficient clusters that are based on coverage

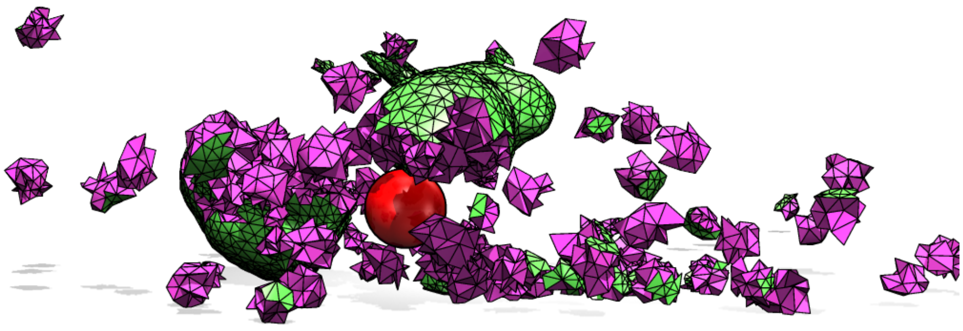
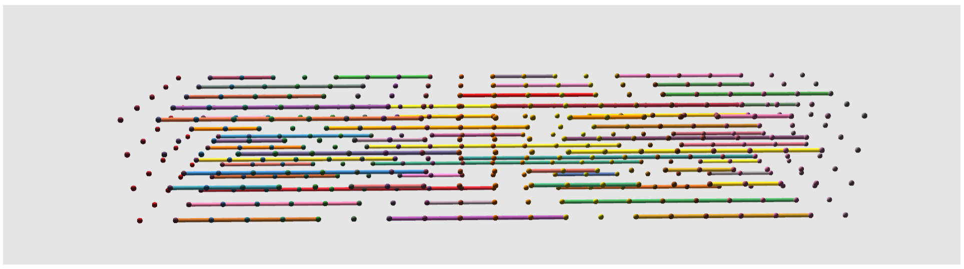


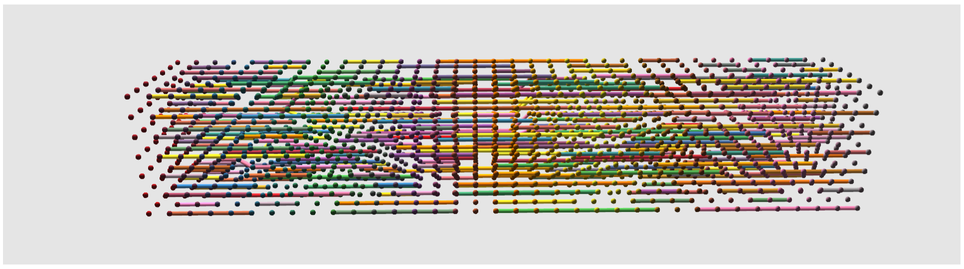
Figure 6.1: Meshing based on shape-matching groups provides enough details for a coarse mesh-representation. Reprinted from Choi (2014).

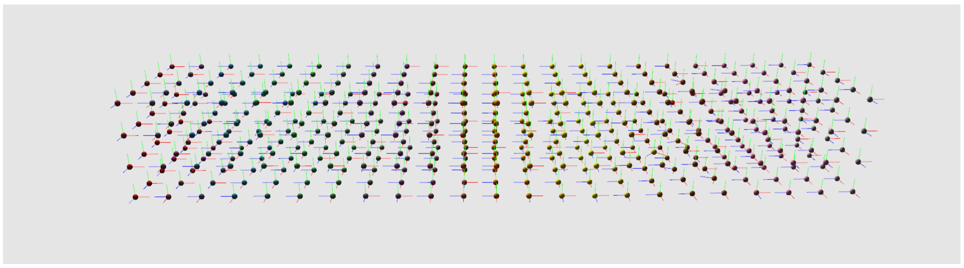
and density. As described in Jones et al. (2015) clustering can not only significantly reduce computational expense by reducing the total number of shape-matching groups but can also be used for collision detection.

6.3 Visulization

We extended an OpenGL-Framework written by Torsten Hädrich. The framework provides a basic user interface as well as common real-time shading techniques such as Phong-shading, shadow-maps, point and spot lighting. This allows us to adjust parameters like, stiffness, solver iterations or time step size during the simulation. Figure 6.2 shows a few parameters that can be visualized for testing and debugging. Other parameters include the principal-strain or fracture direction. Visualization of a mesh is not yet supported but will be based on the shape-matching groups. Every shape-matching group then has its own mesh that can be fixed to it by skinning. This approach was already proven to be sufficient by Choi (2014), fig. 6.1.


(a) Low particle density


(b) High particle density


(c) Particles and their Orientations

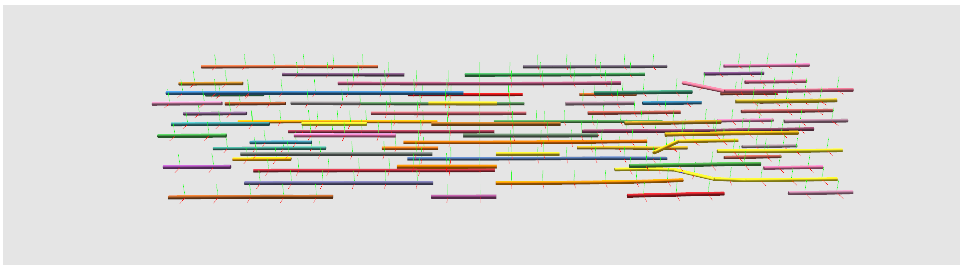

(d) Rods and their orientations

Figure 6.2: Visualization of different parameters along the object.

Chapter 7

Results & Future Work

Figure 7.1 shows an image sequence of a simulation. Due to limited time we used a simple maximum stretch fracture criterion and then removed all constraints that violate the criterion. Using principal strain (or stress) fracture criterion as described in section 4.2.2 would allow for a more fine-tuned constraint-removal as this criterion also provides a fracture direction which can be used to split oriented particle groups. As also evident from the image sequence our implementation is still unrefined so the difference would probably not yet be visible.

7.1 Adaptive Fracture refinement

Note that the coarse fracture in fig. 7.1 already exhibits a jagged fracture one would expect from bending timber beyond its failure point. The missing mesh makes it difficult to see the fracture-plane however, it is save to assume that it is rather crude due to the coarse sampling. The sampling density needed for a refined fracture surface is far beyond the sampling density viable for interactive simulation. This is true for any interactive fracture model and thus there have been two options to address this issue (1) adaptive simulation resolution and (2) procedural fracture refinement.

Adaptive resolutions are difficult as the change in resolution should not change the dynamic response of the overall object. For mesh based approaches like FEM this requires to split existing elements into multiple smaller elements. For example, Bender et al. 2013 extend the $\sqrt{3}$-refinement strategy to perform a 1-3-split of triangle meshes by inserting additional vertices in the center of a triangle element. Further they derive a join operation to allow for mesh coarsening as well. Pfaff et al. (2014) also work on thin sheets but they refine only the region around the fracture tip and only work

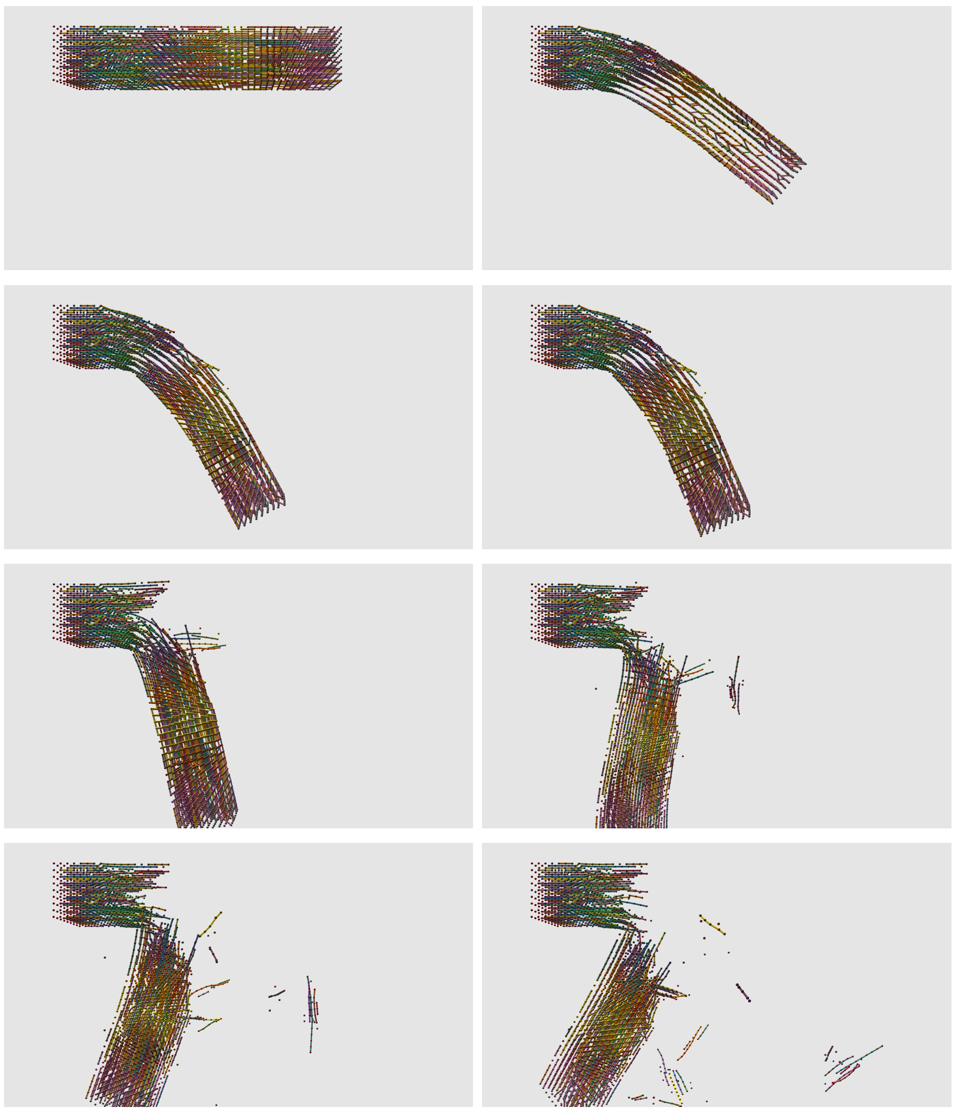


Figure 7.1: Image sequence taken from the simulation of a beam shaped object.

with use edge collapses, edge flips and edge splits. This allows them to keep most of the model at a coarse sampling-density and only refine the region that is of visual interest. Lipponer et al. (2014) describe an adaptive fracture resolution for tetrahedral elements. However, due to the Gauss-Seidel fashion solver the convergence rate and thus elastic response of a PBD-system heavily depends on the particle resolution. Any increase in resolution will reduce the stiffness of the model further, making these fracture refinement methods impractical.

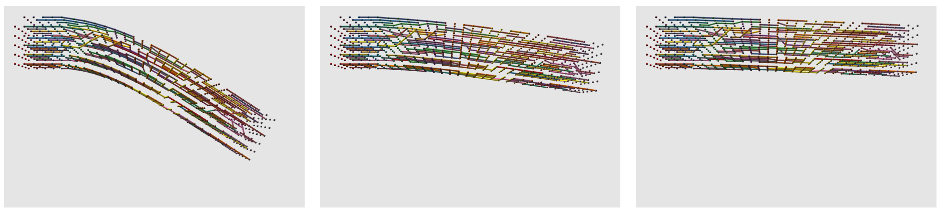

(a) 10 iterations **(b)** 50 iterations **(c)** 250 iterations

Figure 7.2: Effect of iteration count on objects' stiffness.

Chen et al. (2014) proposed to refine the coarse fracture created by an interactive simulation with procedural or texture driven noise. This allows for a more lightweight approach and might be a good match for our simulation approach.

7.2 Global Position Solver

The only external force that is applied for the simulation in fig. 7.1 is gravity. The beam simply bends under its own weight exhibiting very little stiffness. As we are trying to simulate wood which can usually hold not only its own weight but also a significant amount of addition load this stiffness is insufficient. Figure 7.2 shows how iteration count effects the stiffness of the object. This clearly illustrates that the chosen Gauss-Seidel solver is ill suited to simulate stiff materials. Further iterations on this approach should explore global or partial-global solver as introduced in projective dynamics, section 3.2.7, or used in most force based approaches, as this evaluation is beyond the scope of the thesis.

7.3 Conclusion

In this bachelor thesis an overview of continuum mechanics was given and numerous recent methods of interactive dynamic simulation and fracturing were summarized. Among these the PBD based Cosserat rod simulation, which provides time-step independent and robust results, and a solid simulation based on oriented particles, which can be utilized even for co-planar particle sets, were presented. By combining these two approaches we were able to create an anisotropic material model inspired by the natural structure of wood. By employing scattered rods our approach can reproduce varying degrees of anisotropy and is independent of the objects shape. First prototypes verify that the resulting fracture exhibits jagged fracture planes as expected from these kind of materials. However, they also illustrate that the chosen PBD solver unable to provide materials stiffnesses one would expect from wood within reasonable iteration counts. Overall we believe this to be a strong foundation for the interactive simulation of anisotropic materials and encourage future works to improve on the convergence problem.

References

Literature

Abeyaratne, Rohan (2012). “Continuum Mechanics, Volume II of Lecture Notes on The Mechanics of Elastic Solids”. 00000. Quentin Berg Professor of Mechanics MIT Department of Mechanical Engineering (cit. on p. 5).

Baraff, David (1996). “Linear-Time Dynamics Using Lagrange Multipliers”. en. In: 00440. ACM Press, pp. 137–146 (cit. on p. 30).

Bender, Jan and Crispin Deul (2013). “Adaptive Cloth Simulation Using Corotational Finite Elements”. en. In: *Computers & Graphics* 37.7. 00010, pp. 820–829 (cit. on p. 56).

Bender, Jan, Matthias Müller, and Miles Macklin (2017). “A Survey on Position Based Dynamics, 2017”. en. In: 00000, p. 31 (cit. on p. 17).

Bender, Jan et al. (2014a). “A Survey on Position-Based Simulation Methods in Computer Graphics: A Survey on Position-Based Simulation Methods in Computer Graphics”. en. In: *Computer Graphics Forum* 33.6. 00078, pp. 228–251 (cit. on p. 17).

Bender, Jan et al. (2014b). “Position-Based Simulation of Continuous Materials”. en. In: *Computers & Graphics* 44. 00043, pp. 1–10 (cit. on p. 24).

Bergou, Miklós et al. (2008). “Discrete Elastic Rods”. In: *ACM Transactions on Graphics (SIGGRAPH)* 27.3. 00359, 63:1–63:12 (cit. on p. 25).

Bouaziz, Sofien et al. (2014). “Projective Dynamics: Fusing Constraint Projections for Fast Simulation”. en. In: *ACM Transactions on Graphics* 33.4. 00109, pp. 1–11 (cit. on p. 32).

Boyd, Stephen (2010). “Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers”. en. In: *Foundations and Trends® in Machine Learning* 3.1. 07334, pp. 1–122 (cit. on p. 32).

Bridson, Robert (2015). *Fluid Simulation for Computer Graphics.* 00394. AK Peters/CRC Press (cit. on p. 27).

Cai, Jianping, Feng Lin, and Hock Soon Seah (2016). “Fiber Controls in FEM Model for Transversely Isotropic Materials”. en. In: *Graphical Sim-*

ulation of Deformable Models. 00000. Cham: Springer International Publishing, pp. 49–65 (cit. on p. 42).

Chen, Zhili et al. (2014). "Physics-Inspired Adaptive Fracture Refinement". en. In: *ACM Transactions on Graphics* 33.4. 00015, pp. 1–7 (cit. on p. 58).

Choi, Min Gyu (2014). "Real-Time Simulation of Ductile Fracture with Oriented Particles: Real-Time Simulation of Ductile Fracture with Oriented Particles". en. In: *Computer Animation and Virtual Worlds* 25.3-4. 00006, pp. 455–463 (cit. on pp. 40, 54).

Choi, Min Gyu and Jehee Lee (2018). "As-Rigid-as-Possible Solid Simulation with Oriented Particles". en. In: *Computers & Graphics* 70. 00002, pp. 1–7 (cit. on p. 35).

Clavet, Simon, Philippe Beaudoin, and Pierre Poulin (2005). "Particle-Based Viscoelastic Fluid Simulation". en. In: 00284. ACM Press, p. 219 (cit. on p. 42).

Deul, Crispin et al. (2018). "Direct Position-Based Solver for Stiff Rods: Direct Position-Based Solver for Stiff Rods". en. In: *Computer Graphics Forum* 37.6. 00001, pp. 313–324 (cit. on p. 29).

Dinwoodie, J. M (2000). *Timber: Its Nature and Behavior*. English. 00006 OCLC: 664111149. New York; Florence: Spon Press [Imprint] Routledge Taylor & Francis Group [distributor (cit. on p. 2).

Diziol, R., J. Bender, and D. Bayer (2011). "Robust Real-Time Deformation of Incompressible Surface Meshes". en. In: 00029. ACM Press, p. 237 (cit. on p. 34).

Doug, Alec R Rivers and L James (2007). "FastLSM: Fast Lattice Shape Matching for Robust Real-Time Deformation". en. In: 00205, p. 6 (cit. on pp. 34, 35).

Faure, François et al. (2011). "Sparse Meshless Models of Complex Deformable Solids". en. In: 00084, p. 10 (cit. on p. 35).

Flores, Oscar (2015). "Robust Interactive Simulation of Deformable Solids with Detailed Geometry Using Corotational FEM". 00000. PhD thesis. Barcelona, (cit. on p. 17).

Fung, Y. C. (1965). *Foundations of Solid Mechanics*. 04166. Prentice-Hall (cit. on p. 39).

Fung, Y.C. (1993). *A First Course in Continuum Mechanics for Physica and Biological Engineers and Scientists*. 3rd ed. 00000. Pearson (cit. on p. 5).

Gilles, Benjamin et al. (2011). "Frame-Based Elastic Models". en. In: *ACM Transactions on Graphics* 30.2. 00063, pp. 1–12 (cit. on p. 35).

Goldenthal, Rony et al. (2007). "Efficient Simulation of Inextensible Cloth". en. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*. 00296, p. 7 (cit. on p. 30).

Irving, G., J. Teran, and R. Fedkiw (2004). “Invertible Finite Elements for Robust Simulation of Large Deformation”. en. In: 00356. ACM Press, p. 131 (cit. on p. 25).

Jones, Ben et al. (2015). “Clustering and Collision Detection for Clustered Shape Matching”. en. In: 00004. ACM Press, pp. 199–204 (cit. on pp. 35, 54).

Kim, Tae-Yong, Nuttapong Chentanez, and Matthias Müller-Fischer (2012). “Long Range Attachments - A Method to Simulate Inextensible Clothing in Computer Games”. en. In: *Eurographics/ ACM SIGGRAPH*. 00028, p. 6 (cit. on p. 31).

Kugelstadt, Tassilo and Elmar Schömer (2016). “Position and Orientation Based Cosserat Rods.” In: *Symposium on Computer Animation*. 00011, pp. 169–178 (cit. on pp. 26, 27, 29, 42).

Lang, Holger and Joachim Linn (2009). “Lagrangian field Theory in Space and Time for Geometrically Exact Cosserat Rods”. en. In: 00000, p. 21 (cit. on p. 26).

Lang, Holger, Joachim Linn, and Martin Arnold (2011). “Multi-Body Dynamics Simulation of Geometrically Exact Cosserat Rods”. en. In: *Multibody System Dynamics* 25.3. 00102, pp. 285–312 (cit. on p. 26).

Lipponer, Dan Koschier Sebastian and Jan Bender (2014). “Adaptive Tetrahedral Meshes for Brittle Fracture Simulation”. en. In: 00023, p. 10 (cit. on p. 57).

Liu, Ning et al. (2012). “Physical Material Editing with Structure Embedding for Animated Solid”. en. In: 00004, p. 8 (cit. on p. 42).

Macklin, Miles and Matthias Müller (2013). “Position Based Fluids”. en. In: *ACM Transactions on Graphics* 32.4. 00230, p. 1 (cit. on p. 42).

Macklin, Miles, Matthias Müller, and Nuttapong Chentanez (2016). “XPBD: Position-Based Simulation of Compliant Constrained Dynamics”. en. In: 00023. ACM Press, pp. 49–54 (cit. on pp. 29, 31).

Martin, Sebastian et al. (2010). “Unified Simulation of Elastic Rods, Shells, and Solids”. en. In: *ACM Transactions on Graphics* 29.4. 00069, p. 1 (cit. on p. 35).

McGinty, Bob (2018). *Continuum Mechanics*. http://www.continuummechanics.org/. 03639 (cit. on p. 5).

Michels, Dominik L. (2014). “Solving Stiff Cauchy Problems in Scientific Computing – An Analytic-Numeric Approach”. 00000. PhD thesis. University of Bonn (cit. on p. 2).

Michels, Dominik L., Gerrit A. Sobottka, and Andreas G. Weber (2014). “Exponential Integrators for Stiff Elastodynamic Problems”. In: *ACM Trans. Graph.* 33.1. 00024, 7:1–7:20 (cit. on p. 2).

Michels, Dominik L. et al. (2015). “On the Partial Analytical Solution to the Kirchhoff Equation”. In: *Computer Algebra in Scientific Computing – CASC 2015*. Vol. 9301. Lecture Notes in Computer Science. 00000. Berlin, Heidelberg: Springer, pp. 320–331 (cit. on p. 2).

Muguercia, Lien, Carles Bosch, and Gustavo Patow (2014). “Fracture Modeling in Computer Graphics”. en. In: *Computers & Graphics* 45. 00011, pp. 86–100 (cit. on p. 36).

Müller, M, T Y Kim, and N Chentanez (2012). “Fast Simulation of Inextensible Hair and Fur”. en. In: *VRIPHYS*. 00036, p. 6 (cit. on p. 31).

Müller, Matthias (2008). “Hierarchical Position Based Dynamics”. en. In: *The Eurographics Association*. 00102, p. 10 (cit. on p. 30).

Müller, Matthias and Nuttapong Chentanez (2011). “Solid Simulation with Oriented Particles”. en. In: *ACM Transactions on Graphics* 30.4. 00096, p. 1 (cit. on p. 34).

Müller, Matthias, Nuttapong Chentanez, and Tae-Yong Kim (2013). “Real Time Dynamic Fracture with Volumetric Approximate Convex Decompositions”. en. In: *ACM Transactions on Graphics* 32.4. 00043, p. 1 (cit. on p. 36).

Müller, Matthias et al. (2001). “Real-Time Simulation of Deformation and Fracture of Stiff Materials”. en. In: *Computer Animation and Simulation 2001*. Ed. by W. Hansmann et al. 00000. Vienna: Springer Vienna, pp. 113–124 (cit. on pp. 16, 17).

Müller, Matthias et al. (2005). “Meshless Deformations Based on Shape Matching”. In: *ACM SIGGRAPH 2005 Papers*. SIGGRAPH ’05. 00568. Los Angeles, California: ACM, pp. 471–478 (cit. on p. 34).

Müller, Matthias et al. (2007). “Position Based Dynamics”. In: *The Eurographics Association*. 00457 (cit. on p. 17).

Müller, Matthias et al. (2008). “Real Time Physics: Class Notes”. en. In: 00095. ACM Press, p. 1 (cit. on pp. 17, 34).

Müller, Matthias et al. (2014). “Strain Based Dynamics”. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 00033. Eurographics Association, pp. 149–157 (cit. on p. 22).

Narain, Rahul, Matthew Overby, and George E. Brown (2016). “ADMM $\supseteq$ Projective Dynamics: Fast Simulation of General Constitutive Models”. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’16. 00028. Zurich, Switzerland: Eurographics Association, pp. 21–28 (cit. on p. 32).

Ngan, Wai-Hin Wayne and John E Lloyd (2008). “Efficient Deformable Body Simulation Using Stiffness-Warped Nonlinear Finite Elements”. en. In: 00004, p. 8 (cit. on p. 17).

O’brien, James F., Adam W. Bargteil, and Jessica K. Hodgins (2002). “Graphical Modeling and Animation of Ductile Fracture”. In: *ACM transactions on graphics (TOG)* 21.3. 00328, pp. 291–294 (cit. on pp. 16, 17, 39).

O’brien, James F. and Jessica K. Hodgins (1999). “Graphical Modeling and Animation of Brittle Fracture”. In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. 00568. ACM

Press/Addison-Wesley Publishing Co., pp. 137–146 (cit. on pp. 16, 17, 39).

Pfaff, Tobias et al. (2014). "Adaptive Tearing and Cracking of Thin Sheets". en. In: *ACM Transactions on Graphics* 33.4. 00047, pp. 1–9 (cit. on p. 56).

Pirk, Sören et al. (2014). "Windy Trees: Computing Stress Response for Developmental Tree Models". en. In: *ACM Transactions on Graphics* 33.6, pp. 1–11 (cit. on p. 43).

Pirk, Sören et al. (2017). "Interactive Wood Combustion for Botanical Tree Models". en. In: *ACM Transactions on Graphics* 36.6. 00001, pp. 1–12 (cit. on p. 43).

Richfield, David (2009). *Stress Strain Curve.* https://en.wikipedia.org/wiki/File:Stress_v_strain_A36_2.svg. 00000 (cit. on pp. 37, 38).

Schmitt, Nikolas et al. (2013). *Multilevel Cloth Simulation Using GPU Surface Sampling.* en. 00008 (cit. on p. 30).

Selle, Andrew, Michael Lentine, and Ronald Fedkiw (2011). "A Mass Spring Model for Hair Simulation". en. In: 00194, p. 11 (cit. on p. 26).

Sorkine, Olga and Marc Alexa (2007). "As-Rigid-As-Possible Surface Modeling". en. In: 00574, p. 8 (cit. on p. 35).

Sutherland, Sean Meiji (2012). "Fibre Based Modeling of Wood Dynamics and Fracture". 00001. PhD Thesis. University of British Columbia (cit. on pp. 4, 45, 50).

Takahashi, Tetsuya et al. (2016). "Volume Preserving Viscoelastic Fluids with Large Deformations Using Position-Based Velocity Corrections". en. In: *The Visual Computer* 32.1. 00005, pp. 57–66 (cit. on p. 42).

Teschner, J Spillmann M (2007). "CORDE: Cosserat Rod Elements for the Dynamic Simulation of One-Dimensional Elastic Objects". en. In: 00002, p. 10 (cit. on p. 26).

Umetani, Nobuyuki, Ryan Schmidt, and Jos Stam (2014). "Position-Based Elastic Rods". In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* 00037. Eurographics Association, pp. 21–30 (cit. on pp. 26, 29).

Wang, Huamin, James O'Brien, and Ravi Ramamoorthi (2010). "Multi-Resolution Isotropic Strain Limiting". en. In: *Siggraph.* 00083, p. 10 (cit. on p. 30).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —

Breite = 100 mm
Höhe = 50 mm

— Diese Seite nach dem Druck entfernen! —