Available online at www.sciencedirect.com
ScienceDirect
Mathematics and Computers in Simulation 170 (2020) 164–204
MATHEMATICS AND COMPUTERS IN SIMULATION
www.elsevier.com/locate/matcom

Original articles

Higher order finite elements in space and time for anisotropic simulations with variational integrators. Application of an efficient GPU implementation

M. Bartelt*, O. Klöckner, J. Dietzsch, M. Groß

Technische Universität Chemnitz, Professorship of Applied Mechanics and Dynamics, Reichenhainer Straße 70, D-09126 Chemnitz, Germany

Received 27 July 2018; received in revised form 26 June 2019; accepted 20 September 2019
Available online 4 October 2019

Abstract

The efficient use of resources is one of the most important things in the current development. In the field of lightweight constructions, rubber and reinforced polymer composite materials become more and more important. This constantly increases the demands on the simulation software. The development of robust and efficient algorithms for these simulations is the aim of this paper.

The application of higher order finite elements in space has a good influence on solution quality in space, but causes a higher computational cost compared to linear elements. We propose higher order finite elements in space and time in combination with a reinforced viscoelastic material formulation in a variational framework. Therefore, a higher order approximation in space and time is applied. The application of variational based time integrators guarantees the preservation of the total balance of linear momentum and the total balance of angular momentum. In order to fulfill the total balance of energy, an extension with discrete gradients is developed for the variational framework. The achieved time stepping scheme represents a very robust and consistent algorithm for the application of transient finite element simulations with reinforced viscoelastic materials and boundary conditions.

In an implementation, however, the higher order approximation in space and time combined with the viscoelastic material suffers from a high computational effort. Hence, an efficient implementation is required in order to reduce the computational time to a minimum. In our approach, we face this problem by using a GPU and the programming architecture Cuda from NVIDIA, which allows a massive parallelization of time-consuming parts of the simulation. We introduce a pipeline design for the GPU implementation, which provides multiple advantages. This design allows a simple porting of an already existing implementation by means of self-managing pipeline-stages. However, a significant speedup is still achieved due to further optimizations which exploit the architecture of GPUs. In addition, when combining both hardware resources GPU and CPU the computational time can be reduced significantly once more. Therefore, our GPU implementation easily allows a distribution of computational effort between both GPU and CPU. Finally, we show in numerical examples the reached speedup of this approach, and the impact of combining the GPU and the CPU is studied in detail.

Keywords: Energy conservation; Variational integrator; Higher order accuracy; Fiber reinforced viscoelasticity; CUDA; GPU

* Corresponding author.
E-mail addresses: matthias.bartelt.scientific@freenet.de (M. Bartelt), oliver.kloeckner@s2015.tu-chemnitz.de (O. Klöckner), julian.dietzsch@mb.tu-chemnitz.de (J. Dietzsch), michael.gross@mb.tu-chemnitz.de (M. Groß).

https://doi.org/10.1016/j.matcom.2019.09.018

1. Introduction

The increasing development of lightweight construction in the automotive and aircraft fields caused the application of more complex materials. Therefore, the use of viscoelastic materials is essential, because effects like relaxation and dumping cannot be reproduced by hyperelastic materials. Furthermore, different materials classes are combined to new materials that depend on several material properties like reinforced viscoelastic materials. To represent the physical properties of the real material in simulations the requirements increase continuously relating to the robustness and efficiency. Therefore, we present an application of different material formulations in one example to cover a large field of real materials. This includes the application of several formulations based on a NEO-HOOKIAN and SAINT VENANT-KIRCHHOFF model for the elastic and viscoelastic parts, further information are listed in Reference [11,12,21]. The extension to a reinforced viscoelastic material formulation is based on the work of [1].

Compared to linear elements in space, higher order elements in space require more computational effort, but in cases of very accurate solutions they can produce faster results, see References [3,15,18]. Motivated by this approach, we compare the use of different approximation orders in space to extend the work of [2].

Beside the approximation in space and the material formulation the time stepping scheme have to be kept in mind. The stability and the robustness strictly depend on the chosen approximation and the order of approximation in time. A very robust class of time stepping schemes are variational integrators, see [13]. A mechanical system can be formulated by a LAGRANGE function. Compared to [4], the approximation in time does not start with the equations of motion but with the approximation in the LAGRANGE function. The advantages of this formulation are the preservation of the total balance of linear momentum and the total balance of angular momentum. The linear approximation of the LAGRANGE function can be extended to a higher order approximation and can also be used to formulate constraint systems, see [16,20]. The disadvantage of this concept is that the error in the total balance of energy is only bounded but not exactly fulfilled. Therefore, we adapt the concept of discrete gradients from [9] in the same way like [12]. This leads to a time stepping scheme of higher order accuracy and preservation of the total balance of energy. In order to achieve an efficient implementation in addition to the physical accuracy, we show an opportunity for an implementation and parallelization on a GPU.

So-called general-purpose graphics processing units “GPGPU” are nowadays commonly used for accelerating various applications, e.g. high performance finite elements as in [8], where especially linear algebraic operations are accelerated. Compared to a CPU, a GPU is specialized in performing the same instruction on multiple data sets in parallel which leads to a high level of computational parallelism. In fact, linear algebraic operations like matrix–vector operations and matrix–matrix operations benefit from this feature, and with CUBLAS, see [5], a library already exists to accelerate such operations. However, even though those libraries are specialized on BLAS operations, they suffer from performance loss when e.g. a great number of matrix–vector operations with few entries is required. Furthermore, in cases where self-written code provides a significant speedup in comparison to an equivalent BLAS function, e.g. due to adaptions to the problem, it should be possible to port this code on the GPU. Therefore, we want to introduce an implementation in C and C++ for the GPU, which not only focuses on these BLAS operations. Our approach follows a pipeline design and uses the programming architecture Cuda in order to exploit the maximum possible performance of NVIDIA GPUs like the TESLA K20C. The pipeline consists of multiple stages, which either execute self-written and ported code or functionalities from libraries, and are optimized for the GPU based on [6,7,17,19]. These stages are connected with their predecessors and successor and the pipeline manages them and guarantees the right of execution and synchronization with the CPU. Additionally, the pipeline design allows further improvements, e.g. pipeline splitting in order to overcome the limited amount of memory space on the GPU and the parallel execution of stages based on the functionality of streams on Cuda. Another crucial advantage is the ability of distributing computational effort easily between the GPU and CPU to increase the speedup significantly. However, the impacts of the distribution on the performance have to be studied as well to assure the maximum utilization on both hardware resources, the GPU and CPU.

In this work Section 1 contains a brief summary of the developed time stepping schemes with a higher order approximation in time. The GALERKIN variational integrators have the advantages that, first, they are very robust and, second, preserve the balance of total linear momentum and the balance of total angular momentum. However, they cannot preserve the balance of total energy in the order of the NEWTON tolerance.

In Section 2 we introduce a discrete gradient that bounds the balance of the total energy in the order of the Newton tolerance. This section also includes the theoretical conservation of the balance of total linear momentum,

the balance of total angular momentum, and the balance of total energy for fixed boundary nodes and external forces.

In the third section, our theoretical main results are approved by the numerical examples for a viscoelastic rotor with different material domains for the blades. To create a more complex example that underpins the robustness of the time stepping scheme we expand the different viscoelastic materials with an elastic fiber in various directions.

The higher order approximation in space and time combined with the discrete gradient and the viscoelastic material leads to a high computational effort. Therefore, we explain in the fourth section an efficient GPU implementation that supports the CPU. The implementation is done on a NVIDIA GPU with the programming language CUDA. For the numerical example with DIRICHLET and NEUMANN boundaries we show the reduction of the computational time using the GPU.

1.1. Approximation in time

Conservative mechanical systems can be formulated by two energy parts in the well known LAGRANGE function. This function represents the difference of the kinetic and potential energy. By applying the LAGRANGE formalism for these systems the results are the equations of motion in the continuous case. In this paper we introduce a time stepping scheme based on a LAGRANGE formalism in a discrete form, see [13] and the references therein. We extend the LAGRANGE formalism for a conservative system to the LAGRANGE-D'ALEMBERT principal for non-conservative systems. The LAGRANGE function is defined by the kinetic and potential energy of the system:

$$\mathcal{L} = T^{\mathrm{kin}} - V^{\mathrm{int}} \tag{1}$$

In contrast to the potential energy V^{int} the kinetic energy T^{kin} can be formulated as an integral over the body in the reference configuration $\mathcal{B}_0$. The potential energy is split into two parts — the internal material energy and the gravitational energy. The representation of the potential energy in a finite element framework is described as an integral for the reference domain $\mathcal{B}_0^e$ on one element. The material energy describes the energy which is saved in every element during the deformation process. In this paper we use a reinforced fiber material part that depends on an elastic formulation with the fiber direction $\mathbf{M}_{\mathrm{F}}$, see [1]. For rubber materials it is known that they have a dissipative part as well. This dissipative part $\mathcal{E}$ cannot be formulated as a potential energy, because it is a non-reversible process and it reduces the total energy of an isothermal system. As a result of the dissipative formulation the introduction of internal variables $\mathbf{C}_{\mathrm{i}}^e$ is necessary. The external forces $\mathcal{F}^{\mathrm{ext}}$ cannot be formulated as a potential either. For these two parts we require the virtual work $\delta W_{\mathrm{nc}}^{\mathrm{tot},e}$ and the second-kind LAGRANGE-equations.

$$V^{\mathrm{int},e} = \int_{\mathcal{B}_0^e} \rho_0 \psi^e(\mathbf{C}^e(\mathbf{q}^e), \mathbf{C}_{\mathrm{i}}^e, \mathbf{M}_{\mathrm{F}}) - \rho_0 \mathbf{g}^e \cdot \mathbf{q}^e \, \mathrm{d}V \tag{2}$$

$$T^{\mathrm{kin}} = \frac{1}{2} \int_{\mathcal{B}_0} \rho_0 \dot{\mathbf{q}} \cdot \dot{\mathbf{q}} \, \mathrm{d}V \tag{3}$$

$$\delta W_{\mathrm{nc}}^{\mathrm{tot},e} = \int_{\mathcal{B}_0^e} -\mathcal{E}(\mathbf{C}^e(\mathbf{q}^e), \mathbf{C}_{\mathrm{i}}^e, \dot{\mathbf{C}}_{\mathrm{i}}^e) \cdot \delta \mathbf{C}_{\mathrm{i}}^e \, \mathrm{d}V + \int_{\mathcal{B}_0^e} \mathcal{F}^{\mathrm{ext}}(\mathbf{q}^e) \cdot \delta \mathbf{q}^e \, \mathrm{d}V \tag{4}$$

$\mathcal{E}$ is described in [12] and represents a part of the nonlinear evolution equation for the internal variable $\mathbf{C}_{\mathrm{i}}^e$. The description of the dissipative part $\mathcal{E}$ and the free energy function φ is listed in Appendix C.1. In this general description of the conservative and non-conservative energies in the continuous case we introduce a discrete description. The first discretization will be done in time. Therefore, a discrete time line is used. Every time step h_n depends on values at the beginning and values at the end of this step $\mathbf{q}_k, \quad \forall k = 0, \ldots, n_{\mathrm{ti}}$. This can be interpreted as a macro time step. In the case of higher order discretization in time every macro time step is divided by a number of micro time steps. This leads to a classification in micro time steps and macro time steps, compare Fig. 1. One macro time step includes all micro time steps in the interval of the time steps size h_n, which means

$$\breve{\mathbf{q}}_{k+1} = \left[\mathbf{q}_k, \mathbf{q}_{k+\frac{1}{n_{\mathrm{pg}}}}, \mathbf{q}_{k+\frac{2}{n_{\mathrm{pg}}}}, \ldots, \mathbf{q}_{k+\frac{n_{\mathrm{pg}}-1}{n_{\mathrm{pg}}}}, \mathbf{q}_{k+1} \right]^{\mathrm{T}} \tag{5}$$

$$\breve{\mathbf{C}}_{\mathrm{i}k+1} = \left[\mathbf{C}_{\mathrm{i}k}, \mathbf{C}_{\mathrm{i}k+\frac{1}{n_{\mathrm{pg}}}}, \mathbf{C}_{\mathrm{i}k+\frac{2}{n_{\mathrm{pg}}}}, \ldots, \mathbf{C}_{\mathrm{i}k+\frac{n_{\mathrm{pg}}-1}{n_{\mathrm{pg}}}}, \mathbf{C}_{\mathrm{i}k+1} \right]^{\mathrm{T}} \tag{6}$$

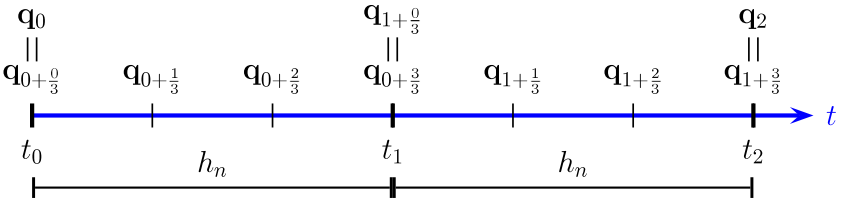

Fig. 1. Time line with both micro time steps and macro time steps for the position vector with an approximation of order $n_{\mathrm{pg}} = 3$.

$$\breve{\boldsymbol{\lambda}}_{k+1} = \left[\boldsymbol{\lambda}_k, \boldsymbol{\lambda}_{k+\frac{1}{n_{\mathrm{pg}}}}, \boldsymbol{\lambda}_{k+\frac{2}{n_{\mathrm{pg}}}}, \ldots, \boldsymbol{\lambda}_{k+\frac{n_{\mathrm{pg}}-1}{n_{\mathrm{pg}}}}, \boldsymbol{\lambda}_{k+1}\right]^{\mathrm{T}} \tag{7}$$

The whole time line for the position vector $\mathbf{q}$, the vector of the internal variables $\mathbf{C}_{\mathrm{i}}$, and the LAGRANGE multipliers $\boldsymbol{\lambda}$ are described as follows.

$$\breve{\mathbf{q}}^{\mathrm{tot}} = \left[\mathbf{q}_{0+\frac{0}{n_{\mathrm{pg}}}}, \mathbf{q}_{0+\frac{1}{n_{\mathrm{pg}}}}, \mathbf{q}_{0+\frac{2}{n_{\mathrm{pg}}}}, \ldots, \mathbf{q}_{0+\frac{n_{\mathrm{pg}}-1}{n_{\mathrm{pg}}}}, \mathbf{q}_{1+\frac{0}{n_{\mathrm{pg}}}}, \ldots, \mathbf{q}_{n_{\mathrm{ti}}-1+\frac{n_{\mathrm{pg}}-1}{n_{\mathrm{pg}}}}, \mathbf{q}_{n_{\mathrm{ti}}}\right]^{\mathrm{T}} \tag{8}$$

$$\breve{\mathbf{C}}_{\mathrm{i}}^{\mathrm{tot}} = \left[\mathbf{C}_{\mathrm{i}0+\frac{0}{n_{\mathrm{pg}}}}, \mathbf{C}_{\mathrm{i}0+\frac{1}{n_{\mathrm{pg}}}}, \mathbf{C}_{\mathrm{i}0+\frac{2}{n_{\mathrm{pg}}}}, \ldots, \mathbf{C}_{\mathrm{i}0+\frac{n_{\mathrm{pg}}-1}{n_{\mathrm{pg}}}}, \mathbf{C}_{\mathrm{i}1+\frac{0}{n_{\mathrm{pg}}}}, \ldots, \mathbf{C}_{\mathrm{i}n_{\mathrm{ti}}-1+\frac{n_{\mathrm{pg}}-1}{n_{\mathrm{pg}}}}, \mathbf{C}_{\mathrm{i}n_{\mathrm{ti}}}\right]^{\mathrm{T}} \tag{9}$$

$$\breve{\boldsymbol{\lambda}}^{\mathrm{tot}} = \left[\boldsymbol{\lambda}_{0+\frac{0}{n_{\mathrm{pg}}}}, \boldsymbol{\lambda}_{0+\frac{1}{n_{\mathrm{pg}}}}, \boldsymbol{\lambda}_{0+\frac{2}{n_{\mathrm{pg}}}}, \ldots, \boldsymbol{\lambda}_{0+\frac{n_{\mathrm{pg}}-1}{n_{\mathrm{pg}}}}, \boldsymbol{\lambda}_{1+\frac{0}{n_{\mathrm{pg}}}}, \ldots, \boldsymbol{\lambda}_{n_{\mathrm{ti}}-1+\frac{n_{\mathrm{pg}}-1}{n_{\mathrm{pg}}}}, \boldsymbol{\lambda}_{n_{\mathrm{ti}}}\right]^{\mathrm{T}} \tag{10}$$

For the evaluation of the discrete quantities and the construction of a time step method, shape functions in time are needed. In this paper we use only one class of shape functions in time for the approximation, the LAGRANGE polynomials with a number m of equidistant points ξ in the interval between zero and one. In the discrete case, derivatives of variables with respect to the time will be created using the shape functions $\dot{M}$. The shape functions M and M' for the polynomial degree one, two, and three are listed in Appendix D. The general form is given by:

$$M_{\frac{i}{m}}(\alpha) = \prod_{\substack{j=0 \\ j\neq i}}^{m} \frac{\alpha - \xi_j}{\xi_i - \xi_j} \qquad \dot{M}_{\frac{i}{m}}(\alpha) = \frac{\partial M_{\frac{i}{m}}(\alpha)}{\partial \alpha}\frac{1}{h_n} = M'_{\frac{i}{m}}(\alpha)\frac{1}{h_n} \tag{11}$$

There are several possibilities for shape functions M, for further details we refer the reader to [13,16,20]. In this paper we use a special collocation method. With a linear approximation in time $m = 1$ is needed in which $m = 2$ leads to a quadratic approximation in time with three shape functions M. The evaluation of the shape functions depends on quadrature points. The shape functions in time used in conjunction with the quadrature points α determine the maximum order of the time stepping method. The application of a quadrature rule with GAUSSIAN points leads to GALERKIN variational integrator. Therefore, the number of GAUSSIAN points α_g is coupled to the approximation order in time. In other words $n_{\mathrm{pg}} = 1$ describes a linear approximation in time with two shape functions and one quadrature point in the same way $n_{\mathrm{pg}} = 2$ describes a quadratic approximation in time with three shape functions and two quadrature points.

The position vector $\mathbf{q}^e$, the right CAUCHY–GREEN $\bar{\mathbf{C}}^e$, and the internal variable $\mathbf{C}_{\mathrm{i}}^e$ are discretized in the following way, compare [13,16]:

$$\mathbf{q}^{e,\mathrm{h}} = \sum_{i=0}^{n_{\mathrm{pg}}} M_{\frac{i}{n_{\mathrm{pg}}}}\, \mathbf{q}^e_{\frac{i}{n_{\mathrm{pg}}}} \qquad \mathbf{q}^{e,\mathrm{d}} = \sum_{i=0}^{n_{\mathrm{pg}}} \dot{M}_{\frac{i}{n_{\mathrm{pg}}}}\, \mathbf{q}^e_{\frac{i}{n_{\mathrm{pg}}}} \tag{12}$$

$$\bar{\mathbf{C}}^{e,\mathrm{h}} = \sum_{i=0}^{n_{\mathrm{pg}}} M_{\frac{i}{n_{\mathrm{pg}}}}\, \bar{\mathbf{C}}^e_{\frac{i}{n_{\mathrm{pg}}}} \qquad \bar{\mathbf{C}}^{e,\mathrm{d}} = \sum_{i=0}^{n_{\mathrm{pg}}} \dot{M}_{\frac{i}{n_{\mathrm{pg}}}}\, \bar{\mathbf{C}}^e_{\frac{i}{n_{\mathrm{pg}}}} \tag{13}$$

$$\mathbf{C}_{\mathrm{i}}^{e,\mathrm{h}} = \sum_{i=0}^{n_{\mathrm{pg}}} M_{\frac{i}{n_{\mathrm{pg}}}}\, \mathbf{C}_{\mathrm{i}}^e{}_{\frac{i}{n_{\mathrm{pg}}}} \qquad \mathbf{C}_{\mathrm{i}}^{e,\mathrm{d}} = \sum_{i=0}^{n_{\mathrm{pg}}} \dot{M}_{\frac{i}{n_{\mathrm{pg}}}}\, \mathbf{C}_{\mathrm{i}}^e{}_{\frac{i}{n_{\mathrm{pg}}}} \tag{14}$$

The continuous LAGRANGE function from Eq. (1) will be discretized in the following way as well as the evolution equation from the LAGRANGE–D'ALEMBERT term:

$$\mathcal{L}_{\mathrm{d},k+1}(\breve{\mathbf{q}}_{k+1}, \breve{\mathbf{C}}_{\mathrm{i}k+1}) = h_n \sum_{g=1}^{n_{\mathrm{pg}}} w_g \mathcal{L}(\mathbf{q}_{k+1}, \dot{\mathbf{q}}_{k+1}, \mathbf{C}_{\mathrm{i}k+1}, \mathbf{M}_{\mathrm{F}}) \tag{15}$$

$$\mathcal{E}_{\mathrm{d},k+1}(\breve{\mathbf{q}}_{k+1}, \breve{\mathbf{C}}_{\mathrm{i}k+1}) = h_n \sum_{g=1}^{n_{\mathrm{pg}}} w_g \mathcal{E}(\mathbf{q}_{k+1}, \mathbf{C}_{\mathrm{i}k+1}, \dot{\mathbf{C}}_{\mathrm{i}k+1}) \tag{16}$$

Since the fiber directions are only parameters, these can be neglected in the functions arguments of the discrete LAGRANGIAN. The discrete LAGRANGIAN including the LAGRANGE multiplier and the constraints function $\boldsymbol{\phi}$ leads to:

$$\breve{\mathcal{L}}_{\mathrm{d},k+1}(\breve{\mathbf{q}}_{k+1}, \breve{\mathbf{C}}_{\mathrm{i}k+1}, \breve{\boldsymbol{\lambda}}_{k+1}) = \mathcal{L}_{\mathrm{d},k+1}(\breve{\mathbf{q}}_{k+1}, \breve{\mathbf{C}}_{\mathrm{i}k+1}) - h_n \sum_{i=0}^{n_{\mathrm{pg}}} \boldsymbol{\lambda}_{k+\frac{i}{n_{\mathrm{pg}}}} \cdot \boldsymbol{\phi}\left(\mathbf{q}_{k+\frac{i}{n_{\mathrm{pg}}}}\right). \tag{17}$$

The construction of a variational integrator depends on the discretization of the state variables and the variation of the discrete LAGRANGIAN. The variation of the action sum with the derivatives with respect to the state variables leads to:

$$\begin{aligned}\delta \mathcal{S}_{\mathrm{d}} &= \delta \sum_{k=0}^{n_{\mathrm{ti}}-1} \breve{\mathcal{L}}_{\mathrm{d},k+1}\left(\breve{\mathbf{q}}^{\mathrm{tot}}, \breve{\mathbf{C}}_{\mathrm{i}}^{\mathrm{tot}}, \breve{\boldsymbol{\lambda}}^{\mathrm{tot}}\right) - \sum_{k=0}^{n_{\mathrm{ti}}-1} \mathcal{E}_{\mathrm{d},k+1} \cdot \delta \breve{\mathbf{C}}_{\mathrm{i}}^{\mathrm{tot}} \\ &= \frac{\partial}{\partial \breve{\mathbf{q}}^{\mathrm{tot}}}\left(\sum_{k=0}^{n_{\mathrm{ti}}-1} \breve{\mathcal{L}}_{\mathrm{d},k+1}\right) \cdot \delta \breve{\mathbf{q}}^{\mathrm{tot}} + \frac{\partial}{\partial \breve{\mathbf{C}}_{\mathrm{i}}^{\mathrm{tot}}}\left(\sum_{k=0}^{n_{\mathrm{ti}}-1} \breve{\mathcal{L}}_{\mathrm{d},k+1}\right) \cdot \delta \breve{\mathbf{C}}_{\mathrm{i}}^{\mathrm{tot}} + \frac{\partial}{\partial \breve{\boldsymbol{\lambda}}^{\mathrm{tot}}}\left(\sum_{k=0}^{n_{\mathrm{ti}}-1} \breve{\mathcal{L}}_{\mathrm{d},k+1}\right) \cdot \delta \breve{\boldsymbol{\lambda}}^{\mathrm{tot}}\end{aligned} \tag{18}$$

The first discrete variation with respect to $\breve{\mathbf{q}}^{\mathrm{tot}}$ and the discrete integration by parts leads to:

$$\begin{aligned}&\frac{\partial}{\partial \breve{\mathbf{q}}^{\mathrm{tot}}}\left(\sum_{k=0}^{n_{\mathrm{ti}}-1} \breve{\mathcal{L}}_{\mathrm{d},k+1}\right) \cdot \delta \breve{\mathbf{q}}^{\mathrm{tot}} \\ &= \sum_{k=0}^{n_{\mathrm{ti}}-1}\left[\mathrm{D}_1 \breve{\mathcal{L}}_{\mathrm{d},k+1} \cdot \delta \mathbf{q}_k + \sum_{i=1}^{n_{\mathrm{pg}}-1}\left(\mathrm{D}_{1+i} \breve{\mathcal{L}}_{\mathrm{d},k+1} \cdot \delta \mathbf{q}_{k+\frac{i}{n_{\mathrm{pg}}}}\right) + \mathrm{D}_{1+n_{\mathrm{pg}}} \breve{\mathcal{L}}_{\mathrm{d},k+1} \cdot \delta \mathbf{q}_{k+1}\right] \\ &= \sum_{k=0}^{n_{\mathrm{ti}}-1} \sum_{i=1}^{n_{\mathrm{pg}}-1}\left(\mathrm{D}_{1+i} \breve{\mathcal{L}}_{\mathrm{d},k+1} \cdot \delta \mathbf{q}_{k+\frac{i}{n_{\mathrm{pg}}}}\right) + \sum_{k=1}^{n_{\mathrm{ti}}-1}\left(\mathrm{D}_1 \breve{\mathcal{L}}_{\mathrm{d},k+1} + \mathrm{D}_{1+n_{\mathrm{pg}}} \breve{\mathcal{L}}_{\mathrm{d},k}\right) \cdot \delta \mathbf{q}_k \\ &\quad + \mathrm{D}_1 \breve{\mathcal{L}}_{\mathrm{d},1} \cdot \delta \mathbf{q}_0 + \mathrm{D}_{1+n_{\mathrm{pg}}} \breve{\mathcal{L}}_{\mathrm{d},n_{\mathrm{ti}}} \cdot \delta \mathbf{q}_{n_{\mathrm{ti}}}\end{aligned} \tag{19}$$

The boundary terms $\delta \mathbf{q}_0$ and $\delta \mathbf{q}_{n_{\mathrm{ti}}} = \mathbf{0}$ are defined as zeros. The derivative of the discrete LAGRANGIAN with respect to the position vector for all time steps will be substituted for receiving a short notation by:

$$\mathrm{D}_{1+i} \breve{\mathcal{L}}_{\mathrm{d},k+1} := \frac{\partial \breve{\mathcal{L}}_{\mathrm{d},k+1}}{\partial \mathbf{q}_{k+\frac{i}{n_{\mathrm{pg}}}}} \quad \text{with} \quad i \in \{0, \dots n_{\mathrm{pg}}\} \tag{20}$$

The collection of terms leads to $n_{\mathrm{pg}} + 1$ equations of the following form:

$$\mathbf{0} = \mathrm{D}_{1+n_{\mathrm{pg}}} \breve{\mathcal{L}}_{\mathrm{d},k} + \mathrm{D}_1 \breve{\mathcal{L}}_{\mathrm{d},k+1} \tag{21}$$

$$\mathbf{0} = \mathrm{D}_{1+i} \breve{\mathcal{L}}_{\mathrm{d},k+1} \quad \forall\, i = 1, \dots, n_{\mathrm{pg}} - 1 \tag{22}$$

The linear momentum can be introduced for every macro time step. That means that there is no information about the linear momentum at each micro time step except the first one and the last one, as these points are similar to the macro time step.

$$\breve{\mathbf{p}}^{\mathrm{tot}} = \left[\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_{n_{\mathrm{ti}}-1}, \mathbf{p}_{n_{\mathrm{ti}}}\right]^{\mathrm{T}} \tag{23}$$

We use the definition of the linear momentum and the variation of the discrete LAGRANGIAN to receive the final discrete EULER–LAGRANGE equations. These equations are invariant up to rigid body motions and are called the

position-momentum equation form (compare Ref. [13]):

$$\mathbf{p}_k = \mathrm{D}_{1+n_{\mathrm{pg}}} \check{\mathcal{L}}_{\mathrm{d},k} = -\mathrm{D}_1 \check{\mathcal{L}}_{\mathrm{d},k+1} \tag{24}$$

$$-\mathbf{p}_k = \mathrm{D}_1 \check{\mathcal{L}}_{\mathrm{d},k+1} \tag{25}$$

$$\mathbf{0} = \mathrm{D}_{1+i} \check{\mathcal{L}}_{\mathrm{d},k+1} \qquad \forall\ i = 1, \ldots, n_{\mathrm{pg}} - 1 \tag{26}$$

$$\mathbf{p}_{k+1} = \mathrm{D}_{1+n_{\mathrm{pg}}} \check{\mathcal{L}}_{\mathrm{d},k+1} \tag{27}$$

In the case of a higher order approximation in time, Eq. (26) has to be regarded. For a linear approximation in time only Eqs. (25) and (27) are required.

For the approximation of the LAGRANGE multipliers we use LAGRANGE polynomials. Instead of using GAUSSIAN quadrature we here use a NEWTON–COTES quadrature. Further information can be found in [2]. This special quadrature rule in combination with the LAGRANGE polynomials leads to shape functions in time in which the result is one, evaluated at the quadrature point, and for all other points the results are zero. The discrete variation with respect to the unknown LAGRANGE multiplier vector $\check{\boldsymbol{\lambda}}^{\mathrm{tot}}$ leads to:

$$\frac{\partial}{\partial \check{\boldsymbol{\lambda}}^{\mathrm{tot}}} \left(\sum_{k=0}^{n_{\mathrm{ti}}-1} \check{\mathcal{L}}_{\mathrm{d},k+1} \right) \cdot \delta \check{\boldsymbol{\lambda}}^{\mathrm{tot}} = - \sum_{k=0}^{n_{\mathrm{ti}}-1} h_n \sum_{i=0}^{n_{\mathrm{pg}}} \boldsymbol{\phi} \left(\mathbf{q}_{k+\frac{i}{n_{\mathrm{pg}}}} \right) \cdot \delta \check{\boldsymbol{\lambda}}_{k+\frac{i}{n_{\mathrm{pg}}}} \tag{28}$$

The aim here is to fulfill the constraints $\boldsymbol{\phi}$ at every micro time step. This is explicitly formulated by:

$$\boldsymbol{\phi} \left(\mathbf{q}_{k+\frac{i}{n_{\mathrm{pg}}}} \right) \Big|_{i=0}^{n_{\mathrm{pg}}} = \mathbf{0} \tag{29}$$

Note that there is no approximation of the position vector in the constraints. This fulfills the constraints at every micro time step on the position level. To fulfill the constraints on velocity level an additional equation is required. The following condition has to be implemented in order to guarantee the correct construction for our time stepping scheme at the last micro time step in every macro time step, and to stabilize the algorithm, compare [2,10,13].

$$\frac{\mathrm{d}\boldsymbol{\phi}\left(\mathbf{q}_{k+1}\right)}{\mathrm{d}t} = \boldsymbol{\chi}\left(\mathbf{q}_{k+1}, \mathbf{p}_{k+1}\right) = \mathbf{0} \tag{30}$$

Based on Eqs. (25) to (27), and Eqs. (29) and (30) we reach to a time stepping scheme in a general form. These equations are the following discrete EULER–LAGRANGE equations in the position-momentum form of a constraint system:

$$-\mathbf{p}_k = \frac{\partial \mathcal{L}_{\mathrm{d},k+1}}{\partial \mathbf{q}_k} - h_n \left(\frac{\partial \boldsymbol{\phi}(\mathbf{q}_k)}{\partial \mathbf{q}_k} \right)^{\mathrm{T}} \cdot \boldsymbol{\lambda}_k \tag{31}$$

$$\forall\ n_{\mathrm{pg}} > 1 \qquad \left\{ \begin{array}{l} \mathbf{0} = \dfrac{\partial \mathcal{L}_{\mathrm{d},k+1}}{\partial \mathbf{q}_{k+\frac{i}{n_{\mathrm{pg}}}}} - h_n \left(\dfrac{\partial \boldsymbol{\phi}\left(\mathbf{q}_{k+\frac{i}{n_{\mathrm{pg}}}}\right)}{\partial \mathbf{q}_{k+\frac{i}{n_{\mathrm{pg}}}}} \right)^{\mathrm{T}} \cdot \boldsymbol{\lambda}_{k+\frac{i}{n_{\mathrm{pg}}}} \\ \mathbf{0} = \boldsymbol{\phi}\left(\mathbf{q}_{k+\frac{i}{n_{\mathrm{pg}}}}\right) \end{array} \right|_{i=1}^{n_{\mathrm{pg}}-1} \tag{32}$$

$$\mathbf{0} = \boldsymbol{\phi}\left(\mathbf{q}_{k+1}\right) \tag{33}$$

$$\mathbf{p}_{k+1} = \frac{\partial \mathcal{L}_{\mathrm{d},k+1}}{\partial \mathbf{q}_{k+1}} - h_n \left(\frac{\partial \boldsymbol{\phi}(\mathbf{q}_{k+1})}{\partial \mathbf{q}_{k+1}} \right)^{\mathrm{T}} \cdot \boldsymbol{\lambda}_{k+1} \tag{34}$$

$$\mathbf{0} = \boldsymbol{\chi}\left(\mathbf{q}_{k+1}, \mathbf{p}_{k+1}\right) \tag{35}$$

Eqs. (31) to (33) have to be solved implicitly for the unknown vector of $\check{\mathbf{q}}_{k+1}$ and $\boldsymbol{\lambda}_{k+\frac{i}{n_{\mathrm{pg}}}}, \quad \forall i = 0, \ldots, n_{\mathrm{pg}} - 1$. Eqs. (34) and (35) are to be solved in the "update"-procedure. Here, both the vector of linear momentum $\mathbf{p}_{k+1}$ and LAGRANGE multiplier vector $\boldsymbol{\lambda}_{k+1}$ are finally calculated. For a linear approximation in time with $n_{\mathrm{pg}} = 1$, this leads to a second order accurate time stepping scheme. For higher order approximation $n_{\mathrm{pg}} > 1$ higher order accurate time stepping schemes are achieved.

The next step is to expand the time stepping scheme by the variation with respect to the internal variables. Therefore, we use once more the definition of the action sum from Eq. (18).

$$\begin{aligned}
&\frac{\partial}{\partial \check{\mathbf{C}}_{\mathrm{i}}^{\mathrm{tot}}}\left(\sum_{k=0}^{n_{\mathrm{ti}}-1}\check{\mathcal{L}}_{\mathrm{d},k+1}\right)\cdot\delta\check{\mathbf{C}}_{\mathrm{i}}^{\mathrm{tot}}-\sum_{k=0}^{n_{\mathrm{ti}}-1}\mathcal{E}_{\mathrm{d},k+1}\cdot\delta\check{\mathbf{C}}_{\mathrm{i}}^{\mathrm{tot}}\\
&=\sum_{k=0}^{n_{\mathrm{ti}}-1}\left[\mathrm{D}_1^{\mathrm{i}}\check{\mathcal{L}}_{\mathrm{d},k+1}\cdot\delta\mathbf{C}_{\mathrm{i}k}+\sum_{i=1}^{n_{\mathrm{pg}}-1}\left(\mathrm{D}_{1+i}^{\mathrm{i}}\check{\mathcal{L}}_{\mathrm{d},k+1}\cdot\delta\mathbf{C}_{\mathrm{i}k+\frac{i}{n_{\mathrm{pg}}}}\right)+\mathrm{D}_{1+n_{\mathrm{pg}}}^{\mathrm{i}}\check{\mathcal{L}}_{\mathrm{d},k+1}\cdot\delta\mathbf{C}_{\mathrm{i}k+1}\right. && (36)\\
&\quad\left.-\mathcal{E}_{\mathrm{d},k}\cdot\delta\mathbf{C}_{\mathrm{i}k}-\sum_{i=1}^{n_{\mathrm{pg}}-1}\left(\mathcal{E}_{\mathrm{d},k+\frac{i}{n_{\mathrm{pg}}}}\cdot\delta\mathbf{C}_{\mathrm{i}k+\frac{i}{n_{\mathrm{pg}}}}\right)-\mathcal{E}_{\mathrm{d},k+1}\cdot\delta\mathbf{C}_{\mathrm{i}k+1}\right] && (37)\\
&=\sum_{k=0}^{n_{\mathrm{ti}}-1}\sum_{i=1}^{n_{\mathrm{pg}}-1}\left(\left[\mathrm{D}_{1+i}^{\mathrm{i}}\check{\mathcal{L}}_{\mathrm{d},k+1}-\mathcal{E}_{\mathrm{d},k+\frac{i}{n_{\mathrm{pg}}}}\right]\cdot\delta\mathbf{C}_{\mathrm{i}k+\frac{i}{n_{\mathrm{pg}}}}\right) && (38)\\
&\quad+\sum_{k=1}^{n_{\mathrm{ti}}-1}\left(\mathrm{D}_1^{\mathrm{i}}\check{\mathcal{L}}_{\mathrm{d},k+1}+\mathrm{D}_{1+n_{\mathrm{pg}}}^{\mathrm{i}}\check{\mathcal{L}}_{\mathrm{d},k}-\mathcal{E}_{\mathrm{d},k}-\mathcal{E}_{\mathrm{d},k}\right)\cdot\delta\mathbf{C}_{\mathrm{i}k}\\
&\quad+\left(\mathrm{D}_1^{\mathrm{i}}\check{\mathcal{L}}_{\mathrm{d},1}-\mathcal{E}_{\mathrm{d},0}\right)\cdot\delta\mathbf{C}_{\mathrm{i}0}+\left(\mathrm{D}_{1+n_{\mathrm{pg}}}^{\mathrm{i}}\check{\mathcal{L}}_{\mathrm{d},n_{\mathrm{ti}}}-\mathcal{E}_{\mathrm{d},n_{\mathrm{ti}}}\right)\cdot\delta\mathbf{C}_{\mathrm{i}n_{\mathrm{ti}}}
\end{aligned}$$

The variation terms $\delta\mathbf{C}_{\mathrm{i}0}$ and $\delta\mathbf{C}_{\mathrm{i}n_{\mathrm{ti}}}=\mathbf{0}$ are defined in the same way as for the position variation as zero. The derivative of the LAGRANGIAN with respect to the internal variable for all micro time steps will be defined in a shorter notation as follows.

$$\mathrm{D}_{1+i}^{\mathrm{i}}\check{\mathcal{L}}_{\mathrm{d},k+1}:=\frac{\partial\check{\mathcal{L}}_{\mathrm{d},k+1}}{\partial\mathbf{C}_{\mathrm{i}k+\frac{i}{n_{\mathrm{pg}}}}}\qquad\text{with}\qquad i\in\{0,\dots n_{\mathrm{pg}}\} \tag{39}$$

In comparison to the variation of the position vector a D'ALEMBERT term is strictly required. Hence:

$$\mathbf{0}=\mathrm{D}_{1+n_{\mathrm{pg}}}^{\mathrm{i}}\check{\mathcal{L}}_{\mathrm{d},k}-\mathcal{E}_{\mathrm{d},k}+\mathrm{D}_1^{\mathrm{i}}\check{\mathcal{L}}_{\mathrm{d},k+1}-\mathcal{E}_{\mathrm{d},k} \tag{40}$$

$$\mathbf{0}=\mathrm{D}_{1+i}^{\mathrm{i}}\check{\mathcal{L}}_{\mathrm{d},k+1}-\mathcal{E}_{\mathrm{d},k+\frac{i}{n_{\mathrm{pg}}}}\qquad\forall\ i=1,\dots,n_{\mathrm{pg}}-1 \tag{41}$$

It is possible to introduce a viscous linear momentum for the internal variable which is similar to the linear momentum. This can be interpreted as viscous linear momentum $\mathbf{p}^{\mathrm{i}}$. Analogous to the linear momentum of the position there exists the viscous linear momentum only at the beginning and at the end of each micro time step.

$$\check{\mathbf{p}}^{\mathrm{i,tot}}=\left[\mathbf{p}_0^{\mathrm{i}},\mathbf{p}_1^{\mathrm{i}},\mathbf{p}_2^{\mathrm{i}},\dots,\mathbf{p}_{n_{\mathrm{ti}}-1}^{\mathrm{i}},\mathbf{p}_{n_{\mathrm{ti}}}^{\mathrm{i}}\right]^{\mathrm{T}} \tag{42}$$

Compared to Eqs. (25) to (27) the parts of the equations are identical except the D'ALEMBERT terms:

$$\mathbf{p}_k^{\mathrm{i}}=\mathrm{D}_{1+n_{\mathrm{pg}}}^{\mathrm{i}}\check{\mathcal{L}}_{\mathrm{d},k}-\mathcal{E}_{\mathrm{d},k}=-\mathrm{D}_1^{\mathrm{i}}\check{\mathcal{L}}_{\mathrm{d},k+1}+\mathcal{E}_{\mathrm{d},k} \tag{43}$$

$$-\mathbf{p}_k^{\mathrm{i}}=\mathrm{D}_1^{\mathrm{i}}\check{\mathcal{L}}_{\mathrm{d},k+1}-\mathcal{E}_{\mathrm{d},k} \tag{44}$$

$$\mathbf{0}=\mathrm{D}_{1+i}^{\mathrm{i}}\check{\mathcal{L}}_{\mathrm{d},k+1}-\mathcal{E}_{\mathrm{d},k+\frac{i}{n_{\mathrm{pg}}}}\qquad\forall\ i=1,\dots,n_{\mathrm{pg}}-1 \tag{45}$$

$$\mathbf{p}_{k+1}^{\mathrm{i}}=\mathrm{D}_{1+n_{\mathrm{pg}}}^{\mathrm{i}}\check{\mathcal{L}}_{\mathrm{d},k+1}-\mathcal{E}_{\mathrm{d},k+1} \tag{46}$$

The structure of these equations is known as evolution equation for the internal variable $\mathbf{C}_{\mathrm{i}}$.

$$f(\mathbf{C},\mathbf{C}_{\mathrm{i}},\dot{\mathbf{C}}_{\mathrm{i}}):=\mathcal{E}(\mathbf{C}(\mathbf{q}),\mathbf{C}_{\mathrm{i}},\dot{\mathbf{C}}_{\mathrm{i}})=-\frac{\partial\psi(\mathbf{C},\mathbf{C}_{\mathrm{i}})}{\partial\mathbf{C}_{\mathrm{i}}}=:\hat{\mathbf{S}} \tag{47}$$

Since the viscous linear momentum is always zero, the structure can be directly generated. The evolution equations for the internal variable have to be solved implicitly. Therefore, a NEWTON method is used. This leads to a multi-level NEWTON method.

1.2. Local residual

By definition of Eq. (44) to (46) it can be seen that the "update equation" is not necessary, because the system of equations can be solved for the unknown internal variables. The "update equation" only delivers $\mathbf{p}^{\mathrm{i}}_{k+1} = \mathbf{0}$, so Eq. (50) can be skipped. The discretized variables have to be evaluated at every quadrature point α_g with its corresponding weights w_g, compare (11).

$$-\mathbf{p}^{\mathrm{i}}_{k} = \mathbf{0} = -\sum_{g=1}^{n_{\mathrm{pg}}} w_g h_n \frac{\partial \psi(\bar{\mathbf{C}}^{e,\mathrm{h}}, \mathbf{C}^{e,\mathrm{h}}_{\mathrm{i}})}{\partial \mathbf{C}_{\mathrm{i}_{k+\frac{0}{n_{\mathrm{pg}}}}}} - \sum_{g=1}^{n_{\mathrm{pg}}} w_g h_n \mathcal{E}_{\mathrm{d},k+\frac{0}{n_{\mathrm{pg}}}} \left(\bar{\mathbf{C}}^{e,\mathrm{h}}, \mathbf{C}^{e,\mathrm{h}}_{\mathrm{i}}, \mathbf{C}^{e,\mathrm{d}}_{\mathrm{i}}\right) \tag{48}$$

$$\mathbf{0} = -\sum_{g=1}^{n_{\mathrm{pg}}} w_g h_n \frac{\partial \psi(\bar{\mathbf{C}}^{e,\mathrm{h}}, \mathbf{C}^{e,\mathrm{h}}_{\mathrm{i}})}{\partial \mathbf{C}_{\mathrm{i}_{k+\frac{i}{n_{\mathrm{pg}}}}}} - \sum_{g=1}^{n_{\mathrm{pg}}} w_g h_n \mathcal{E}_{\mathrm{d},k+\frac{i}{n_{\mathrm{pg}}}} \left(\bar{\mathbf{C}}^{e,\mathrm{h}}, \mathbf{C}^{e,\mathrm{h}}_{\mathrm{i}}, \mathbf{C}^{e,\mathrm{d}}_{\mathrm{i}}\right) \quad \forall i = 1, \ldots, n_{\mathrm{pg}} - 1 \tag{49}$$

$$\mathbf{p}^{\mathrm{i}}_{k+1} = \mathbf{0} = -\sum_{g=1}^{n_{\mathrm{pg}}} w_g h_n \frac{\partial \psi(\bar{\mathbf{C}}^{e,\mathrm{h}}, \mathbf{C}^{e,\mathrm{h}}_{\mathrm{i}})}{\partial \mathbf{C}_{\mathrm{i}k+1}} - \sum_{g=1}^{n_{\mathrm{pg}}} w_g h_n \mathcal{E}_{\mathrm{d},k+1} \left(\bar{\mathbf{C}}^{e,\mathrm{h}}, \mathbf{C}^{e,\mathrm{h}}_{\mathrm{i}}, \mathbf{C}^{e,\mathrm{d}}_{\mathrm{i}}\right) \tag{50}$$

As a result, the local residual is achieved. The unknown internal variables for one time step will be summarized by $\mathbf{C}_{\mathrm{i}_\iota} \quad \forall \iota = k + \frac{1}{n_{\mathrm{pg}}}, \ldots, k + \frac{n_{\mathrm{pg}}}{n_{\mathrm{pg}}}$. In every time step the $\mathbf{C}_{\mathrm{i}k+0}$ is known as initial condition.

$$\mathbf{R}^{\mathrm{loc}}_{\mathrm{Ci}}(\mathbf{C}_{\mathrm{i}_\iota}, \mathbf{q}_\iota) := \begin{cases} \displaystyle\sum_{g=1}^{n_{\mathrm{pg}}} w_g h_n \frac{\partial \psi(\bar{\mathbf{C}}^{e,\mathrm{h}}, \mathbf{C}^{e,\mathrm{h}}_{\mathrm{i}})}{\partial \mathbf{C}_{\mathrm{i}_{k+\frac{0}{n_{\mathrm{pg}}}}}} + \sum_{g=1}^{n_{\mathrm{pg}}} w_g h_n \mathcal{E}(\bar{\mathbf{C}}^{e,\mathrm{h}}, \mathbf{C}^{e,\mathrm{h}}_{\mathrm{i}}, \mathbf{C}^{e,\mathrm{d}}_{\mathrm{i}})_{\mathrm{d},k+\frac{0}{n_{\mathrm{pg}}}} \\ \vdots \\ \displaystyle\sum_{g=1}^{n_{\mathrm{pg}}} w_g h_n \frac{\partial \psi(\bar{\mathbf{C}}^{e,\mathrm{h}}, \mathbf{C}^{e,\mathrm{h}}_{\mathrm{i}})}{\partial \mathbf{C}_{\mathrm{i}_{k+\frac{n_{\mathrm{pg}}-1}{n_{\mathrm{pg}}}}}} + \sum_{g=1}^{n_{\mathrm{pg}}} w_g h_n \mathcal{E}(\bar{\mathbf{C}}^{e,\mathrm{h}}, \mathbf{C}^{e,\mathrm{h}}_{\mathrm{i}}, \mathbf{C}^{e,\mathrm{d}}_{\mathrm{i}})_{\mathrm{d},k+\frac{n_{\mathrm{pg}}-1}{n_{\mathrm{pg}}}} \end{cases} \tag{51}$$

The derivatives are built by TAYLOR series with respect to the unknown.

$$\mathbf{R}^{\mathrm{loc}}_{\mathrm{Ci}}(\mathbf{C}_{\mathrm{i}_\iota} + \Delta\mathbf{C}_{\mathrm{i}}, \mathbf{q}_\iota + \Delta\mathbf{q}) = \mathbf{R}^{\mathrm{loc}}_{\mathrm{Ci}}(\mathbf{C}_{\mathrm{i}_\iota}, \mathbf{q}_\iota) + \frac{\partial \mathbf{R}^{\mathrm{loc}}_{\mathrm{Ci}}(\mathbf{C}_{\mathrm{i}_\iota}, \mathbf{q}_\iota)}{\partial \mathbf{C}_{\mathrm{i}_\iota}} \Delta\mathbf{C}_{\mathrm{i}_\iota} + \frac{\partial \mathbf{R}^{\mathrm{loc}}_{\mathrm{Ci}}(\mathbf{C}_{\mathrm{i}_\iota}, \mathbf{q}_\iota)}{\partial \mathbf{q}_\iota} \Delta\mathbf{q} = \mathbf{0} \tag{52}$$

$$-\mathbf{R}^{\mathrm{loc}}_{\mathrm{Ci}}(\mathbf{C}_{\mathrm{i}_\iota}, \mathbf{q}_\iota) = \mathbf{T}^{\mathrm{loc}}_{\mathrm{Ci}}(\mathbf{C}_{\mathrm{i}_\iota}, \mathbf{q}_\iota)\Delta\mathbf{C}_{\mathrm{i}} + \mathbf{T}^{\mathrm{loc}}_{\mathrm{q}}(\mathbf{C}_{\mathrm{i}_\iota}, \mathbf{q}_\iota)\Delta\mathbf{q} \tag{53}$$

Only in the local residual the unknown positions $\mathbf{q}_\iota$ are set as known. As a result, we obtain the increment $\Delta\mathbf{q} = \mathbf{0}$. The update of the internal variables follows by:

$$\Delta\mathbf{C}_{\mathrm{i}} = -\left(\mathbf{T}^{\mathrm{loc}}_{\mathrm{Ci}}(\mathbf{C}_{\mathrm{i}_\iota}, \mathbf{q}_\iota)\right)^{-1} \mathbf{R}^{\mathrm{loc}}_{\mathrm{Ci}}(\mathbf{C}_{\mathrm{i}_\iota}, \mathbf{q}_\iota). \tag{54}$$

This is done iteratively by checking $\|\mathbf{R}^{\mathrm{loc}}_{\mathrm{Ci}}(\mathbf{C}^{m}_{\mathrm{i}_\iota}, \mathbf{q}_\iota)\|$ so that there is $\mathbf{C}^{m+1}_{\mathrm{i}_\iota} \approx \mathbf{C}^{m}_{\mathrm{i}_\iota}$ for m NEWTON iterations.

$$\mathbf{C}^{m+1}_{\mathrm{i}_\iota} = \mathbf{C}^{m}_{\mathrm{i}_\iota} + \Delta\mathbf{C}^{m}_{\mathrm{i}} \tag{55}$$

$$\mathbf{C}^{m+1}_{\mathrm{i}_\iota} = \mathbf{C}^{m}_{\mathrm{i}_\iota} - \left(\mathbf{T}^{\mathrm{loc}}_{\mathrm{Ci}}(\mathbf{C}^{m}_{\mathrm{i}_\iota}, \mathbf{q}_\iota)\right)^{-1} \mathbf{R}^{\mathrm{loc}}_{\mathrm{Ci}}(\mathbf{C}^{m}_{\mathrm{i}_\iota}, \mathbf{q}_\iota) \tag{56}$$

The general way of the construction for a variational integrator has to be shown including internal variables and LAGRANGE multipliers. The goal of this paper is to extend this principal to an integrator that preserves the balance of total energy, too. Therefore, a discrete gradient will be applied.

2. Extension to energy conservation

Relating to the previous algorithm, two important things have to be changed. The first change is the introduction of an assumed strain definition for the right CAUCHY–GREEN $\mathbf{C}$, see [10] for the complete bibliography. For the discrete gradient in this paper two different approximations of the right CAUCHY–GREEN tensors are required.

$$\bar{\mathbf{S}}^{e} = \bar{\mathbf{S}}^{e}\left(\bar{\mathbf{C}}^{e,\mathrm{h}}, \mathbf{C}^{e,\mathrm{h}}_{\mathrm{i}}\right) \neq \mathbf{S}^{e}\left(\mathbf{C}^{e}\left(\mathbf{q}^{e,\mathrm{h}}\right), \mathbf{C}^{e,\mathrm{h}}_{\mathrm{i}}\right) \tag{57}$$

$$\frac{\partial \mathbf{C}^{e}\left(\mathbf{q}^{e,\mathrm{h}}\right)}{\partial \alpha} = \mathbf{C}^{e,\mathrm{hd}} = \left(\mathbf{F}\left(\mathbf{q}^{e,\mathrm{h}}\right)\right)^{\mathrm{T}} \cdot \mathbf{F}\left(\mathbf{q}^{e,\mathrm{d}}\right) + \left(\mathbf{F}\left(\mathbf{q}^{e,\mathrm{d}}\right)\right)^{\mathrm{T}} \cdot \mathbf{F}\left(\mathbf{q}^{e,\mathrm{h}}\right) \tag{58}$$

The second change in the time stepping scheme is the new stress definition, see [9]. In addition to the normal stress term an algorithmic correction is necessary, given by:

$$\bar{\mathbf{S}}^e + \frac{2\psi\left(\bar{\mathbf{C}}^e_{k+1}, \mathbf{C}_{\mathrm{i}k+1}\right) - 2\psi\left(\bar{\mathbf{C}}^e_k, \mathbf{C}_{\mathrm{i}k}\right) - \mathcal{G}^{\mathrm{e}} - 2\mathcal{G}^{\mathrm{e,vis}}}{\mathcal{N}^{\mathrm{e}}} \bar{\mathbf{C}}^{e,\mathrm{d}} = \bar{\mathbf{S}}^e + \mathcal{D}^{\mathrm{e}} \bar{\mathbf{C}}^{e,\mathrm{d}} \tag{59}$$

$\mathcal{N}^{\mathrm{e}}$, $\mathcal{G}^{\mathrm{e}}$ and $\mathcal{G}^{\mathrm{e,vis}}$ are defined by:

$$\mathcal{N}^{\mathrm{e}} = \sum_{g=1}^{n_{\mathrm{pg}}} h_n w_g \mathrm{tr}\left(\overset{\mathrm{AD}}{\mathcal{P}}_{\mathrm{M}}\left\{\bar{\mathbf{C}}^{e,\mathrm{d}}\ ,\ \mathbf{C}^{e,\mathrm{hd}}\right\}\right) \tag{60}$$

$$\mathcal{G}^{\mathrm{e}} = \sum_{g=1}^{n_{\mathrm{pg}}} h_n w_g \mathrm{tr}\left(\overset{\mathrm{AD}}{\mathcal{P}}_{\mathrm{M}}\left\{2\frac{\partial\psi(\bar{\mathbf{C}}^{e,\mathrm{h}}, \mathbf{C}^{e,\mathrm{h}}_{\mathrm{i}})}{\partial\bar{\mathbf{C}}^{e,\mathrm{h}}}\ ,\ \mathbf{C}^{e,\mathrm{hd}}\right\}\right) = \sum_{g=1}^{n_{\mathrm{pg}}} h_n w_g \mathrm{tr}\left(\overset{\mathrm{AD}}{\mathcal{P}}_{\mathrm{M}}\left\{\bar{\mathbf{S}}^e\ ,\ \mathbf{C}^{e,\mathrm{hd}}\right\}\right) \tag{61}$$

$$\mathcal{G}^{\mathrm{e,vis}} = \sum_{g=1}^{n_{\mathrm{pg}}} h_n w_g \mathrm{tr}\left(\overset{\mathrm{AD}}{\mathcal{P}}_{\mathrm{M}}\left\{\frac{\partial\psi(\mathbf{C}^{e,\mathrm{h}}_{\mathrm{i}}, \bar{\mathbf{C}}^{e,\mathrm{h}})}{\partial\mathbf{C}^{e,\mathrm{h}}_{\mathrm{i}}}\ ,\ \mathbf{C}^{e,\mathrm{d}}_{\mathrm{i}}\right\}\right) = \sum_{g=1}^{n_{\mathrm{pg}}} h_n w_g \mathrm{tr}\left(\overset{\mathrm{AD}}{\mathcal{P}}_{\mathrm{M}}\left\{\hat{\mathbf{S}}^e\ ,\ \mathbf{C}^{e,\mathrm{d}}_{\mathrm{i}}\right\}\right). \tag{62}$$

Eq. (62) follows from the definition of the internal dissipation. As a result, the balance of the total internal energy can be written as:

$$0 = \psi(1) - \psi(0) + D^{\mathrm{int}} = \psi\left(\bar{\mathbf{C}}^e_{k+1}, \mathbf{C}_{\mathrm{i}k+1}\right) - \psi\left(\bar{\mathbf{C}}^e_k, \mathbf{C}_{\mathrm{i}k}\right) - \frac{\partial\psi}{\partial\mathbf{C}_{\mathrm{i}}} \cdot \dot{\mathbf{C}}_{\mathrm{i}}. \tag{63}$$

The stress definitions are listed in Appendix C.2 and for the operator definitions see Appendix A, the construction of the operators goes back to the work in [2]. The introduction of the discrete gradient cannot be directly generated by a derivative of a free energy function, but potential forces can also be introduced by D'ALEMBERT's principle. D'ALEMBERT's principle is used for the discrete gradient with an assumed strain formulation to introduce this term in the variational principle. Therefore, we use the terms of virtual work at each micro time step.

$$\delta W^{\mathrm{int}}_{k+1}(\breve{\mathbf{q}}_{k+1}) = -\bigcup_{e=1}^{n_{\mathrm{el}}} \int_{\mathcal{B}^e_0} \sum_{g=1}^{n_{\mathrm{pg}}} w_g h_n \overset{\mathrm{BS}}{\mathcal{P}}_{\mathrm{M}}\left\{\mathbf{F}\left(\mathbf{q}^{e,\mathrm{h}}\right)\ ,\ \bar{\mathbf{S}}^e + \mathcal{D}^{\mathrm{e}}\bar{\mathbf{C}}^{e,\mathrm{d}}\right\} \sum_{i=0}^{n_{\mathrm{pg}}} M_{\frac{i}{n_{\mathrm{pg}}}} \delta\mathbf{q}^e_{k+\frac{i}{n_{\mathrm{pg}}}} \tag{64}$$

As a result, the new time stepping scheme with the Mass-matrix $\mathbf{M}_{\mathrm{M}}$ is achieved. This scheme is called VDG for a Variational integrator with Discrete Gradient in comparison to VI for Variational Integrator. The discretized variables have to be evaluated at every quadrature point α_g with its corresponding weights w_g, compare (11).

$$\mathbf{0} = \mathbf{R}^{\mathrm{loc}}(\mathbf{C}_{\mathrm{i}\iota}, \mathbf{q}_\iota) \tag{65}$$

$$-\mathbf{p}_k = \bigcup_{e=1}^{n_{\mathrm{el}}} \left[\sum_{g=1}^{n_{\mathrm{pg}}} w_g h_n \dot{M}_0 \mathbf{M}_{\mathrm{M}}{}^e \cdot \mathbf{q}^{e,\mathrm{d}}\right] - h_n \left(\frac{\partial\boldsymbol{\phi}(\mathbf{q}_k)}{\partial\mathbf{q}_k}\right)^{\mathrm{T}} \cdot \boldsymbol{\lambda}_k \tag{66}$$

$$- \bigcup_{e=1}^{n_{\mathrm{el}}} \int_{\mathcal{B}^e_0} \sum_{g=1}^{n_{\mathrm{pg}}} w_g h_n M_0 \left[\overset{\mathrm{BS}}{\mathcal{P}}_{\mathrm{M}}\left\{\mathbf{F}\left(\mathbf{q}^{e,\mathrm{h}}\right)\ ,\ \bar{\mathbf{S}}^e + \mathcal{D}^{\mathrm{e}}\bar{\mathbf{C}}^{e,\mathrm{d}}\right\} - \rho_0\mathbf{g}^e\right] \mathrm{d}V \tag{67}$$

$$\forall\ n_{\mathrm{pg}} > 1 \quad \left\{\begin{array}{l} \mathbf{0} = \bigcup_{e=1}^{n_{\mathrm{el}}} \left[\sum_{g=1}^{n_{\mathrm{pg}}} w_g h_n \dot{M}_{\frac{i}{n_{\mathrm{pg}}}} \mathbf{M}_{\mathrm{M}}{}^e \cdot \mathbf{q}^{e,\mathrm{d}}\right] - h_n \left(\frac{\partial\boldsymbol{\phi}\left(\mathbf{q}_{k+\frac{i}{n_{\mathrm{pg}}}}\right)}{\partial\mathbf{q}_{k+\frac{i}{n_{\mathrm{pg}}}}}\right)^{\mathrm{T}} \cdot \boldsymbol{\lambda}_{k+\frac{i}{n_{\mathrm{pg}}}} \\ \qquad - \bigcup_{e=1}^{n_{\mathrm{el}}} \int_{\mathcal{B}^e_0} \sum_{g=1}^{n_{\mathrm{pg}}} w_g h_n M_{\frac{i}{n_{\mathrm{pg}}}} \left[\overset{\mathrm{BS}}{\mathcal{P}}_{\mathrm{M}}\left\{\mathbf{F}\left(\mathbf{q}^{e,\mathrm{h}}\right)\ ,\ \bar{\mathbf{S}}^e + \mathcal{D}^{\mathrm{e}}\bar{\mathbf{C}}^{e,\mathrm{d}}\right\} - \rho_0\mathbf{g}^e\right] \mathrm{d}V \\ \mathbf{0} = \boldsymbol{\phi}\left(\mathbf{q}_{k+\frac{i}{n_{\mathrm{pg}}}}\right) \end{array}\right|_{i=1}^{n_{\mathrm{pg}}-1} \tag{68}$$

$$\mathbf{0} = \boldsymbol{\phi}(\mathbf{q}_{k+1}) \tag{69}$$

$$\mathbf{p}_{k+1} = \bigcup_{e=1}^{n_{\mathrm{el}}} \left[\sum_{g=1}^{n_{\mathrm{pg}}} w_g h_n \dot{M}_1 \mathbf{M}_{\mathrm{M}}{}^{e} \cdot \mathbf{q}^{e,\mathrm{d}} \right] - h_n \left(\frac{\partial \boldsymbol{\phi}(\mathbf{q}_{k+1})}{\partial \mathbf{q}_{k+1}} \right)^{\mathrm{T}} \cdot \boldsymbol{\lambda}_{k+1} \tag{70}$$

$$- \bigcup_{e=1}^{n_{\mathrm{el}}} \int_{\mathcal{B}_0^e} \sum_{g=1}^{n_{\mathrm{pg}}} w_g h_n M_1 \left[\overset{\mathrm{BS}}{\mathcal{P}}_{\mathrm{M}} \left\{ \mathbf{F}\left(\mathbf{q}^{e,\mathrm{h}}\right),\ \bar{\mathbf{S}}^e + \mathcal{D}^{\mathrm{e}} \bar{\mathbf{C}}^{e,\mathrm{d}} \right\} - \rho_0 \mathbf{g}^e \right] \mathrm{d}V \tag{71}$$

$$\mathbf{0} = \boldsymbol{\chi}\left(\mathbf{q}_{k+1}, \mathbf{p}_{k+1}\right) \tag{72}$$

This very general form of the time stepping scheme can be used for a large class of material formulations. Notes on implementation can be found in [2] for a hyperelastic material. A detailed implementation for several material formulations is listed in Appendix C.1. The difference to an implementation with a hyperelastic material formulation consists of the internal variable and the associated calculation of the local residual. This point is very important for the consistent linearization. A short algorithm for the consistent linearization is shown in Algorithm 1.

2.1. External forces

For the calculation of external forces and the calculation of conservation properties, the position vector $\mathbf{q}$, the vector of linear momentum $\mathbf{p}$ and the vector of the LAGRANGE-multiplier are also needed as node values. In this case, the index N describes the number of the respective node. The each node value contains the spatial directions in the x, y, z-direction. Every finite element contains a list of nodes depending on the approximation order in space. More details about the shape functions in space can be found in [2,21]. The number of nodes in each element is described by n_{no} as opposed to the total number of nodes n_{ano}.

$$\mathbf{q}^e = \left[\mathbf{q}^{e,1}, \mathbf{q}^{e,2} \ldots, \mathbf{q}^{e,n_{\mathrm{no}}-1}, \mathbf{q}^{e,n_{\mathrm{no}}}\right] \tag{73}$$

$$\mathbf{p}^e = \left[\mathbf{p}^{e,1}, \mathbf{p}^{e,2} \ldots, \mathbf{p}^{e,n_{\mathrm{no}}-1}, \mathbf{p}^{e,n_{\mathrm{no}}}\right] \tag{74}$$

$$\boldsymbol{\lambda}^e = \left[\boldsymbol{\lambda}^{e,1}, \boldsymbol{\lambda}^{e,2} \ldots, \boldsymbol{\lambda}^{e,n_{\mathrm{no}}-1}, \boldsymbol{\lambda}^{e,n_{\mathrm{no}}}\right] \tag{75}$$

In the same way, the variables can be defined for all nodes.

$$\mathbf{q}^N = \left[\mathbf{q}_x^N, \mathbf{q}_y^N, \mathbf{q}_z^N\right] \tag{76}$$

$$\mathbf{p}^N = \left[\mathbf{p}_x^N, \mathbf{p}_y^N, \mathbf{p}_z^N\right] \tag{77}$$

$$\boldsymbol{\lambda}^N = \left[\boldsymbol{\lambda}_x^N, \boldsymbol{\lambda}_y^N, \boldsymbol{\lambda}_z^N\right] \tag{78}$$

Fig. 2 shows a space–time mesh for all nodes. Boundary conditions in the form of NEUMANN boundaries can easily be implemented in the variational framework by using D'ALEMBERT's principle. Forces can be applied at each node in a specific direction or to the boundaries for every element. Additionally, time dependent forces are applied to the system. The goal is to define a force in the following way.

$$\delta W_{\mathrm{nc}}^{\mathrm{Fext}} = \int_{t_0}^{t_1} \mathcal{F}^{\mathrm{ext}}(t)\, \mathrm{d}t = \int_{t_0}^{t_1} \hat{\mathbf{F}}^e f(t)\, \mathrm{d}t \tag{79}$$

$$\delta W_{\mathrm{nc},k+1}^{eA} = \hat{\mathbf{F}}^{e,A} \sum_{g=1}^{n_{\mathrm{pg}}} w_g \sum_{i=0}^{n_{\mathrm{pg}}} M_{\frac{i}{n_{\mathrm{pg}}}}(\alpha_g) f_{\frac{i}{n_{\mathrm{pg}}}} \sum_{i=0}^{n_{\mathrm{pg}}} M_{\frac{i}{n_{\mathrm{pg}}}}(\alpha_g) \delta \mathbf{q}^e_{k+\frac{i}{n_{\mathrm{pg}}}} \tag{80}$$

$$\delta W_{\mathrm{nc},k+1}^{eA} = \hat{\mathbf{F}}^{e,A} \sum_{g=1}^{n_{\mathrm{pg}}} w_g f^{\mathrm{h}}(\alpha_g) \sum_{i=0}^{n_{\mathrm{pg}}} M_{\frac{i}{n_{\mathrm{pg}}}}(\alpha_g) \delta \mathbf{q}^{eA}_{k+\frac{i}{n_{\mathrm{pg}}}} \tag{81}$$

For the numerical examples we apply a time dependent pressure on various surfaces on the blades of the rotor. The pressure p_0 is applied on the boundary area Ω_0^e. This nodal force is calculated from the pressure by using only the reference configuration and will not be updated by the actual configuration, compare [21]. Therefore, 2-D shape functions $\mathrm{N}_{2\mathrm{D}}$ are required in space with the number of nodes $n_{\mathrm{no}\Omega}$ and the number of quadrature points $n_{\mathrm{gph}\Omega}$ for the 2-D boundary.

$$\hat{\mathbf{F}}^e = \int_{\Omega_0^e} p_0 \mathrm{N}_{2\mathrm{D}}\left(\mathbf{X}_{\mathrm{ref}}\right) \mathbf{n}_0\, \mathrm{d}\xi\, \mathrm{d}\eta \tag{82}$$

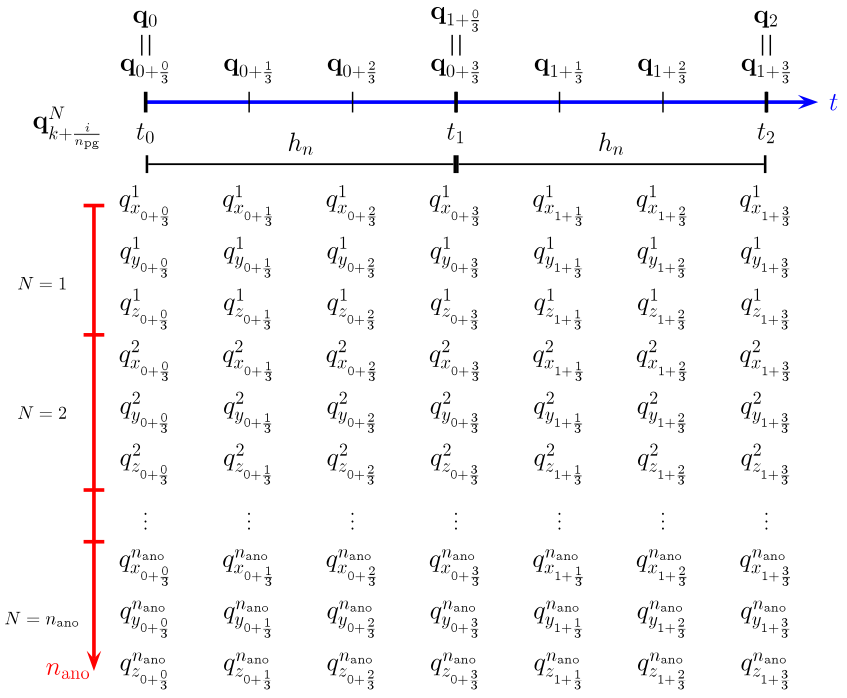

Fig. 2. Time and space lines for the position vector with an approximation in time of order $n_{\mathrm{pg}} = 3$.

$$\hat{\mathbf{F}}^{eA} = p_0 \sum_{k=1}^{n_{\mathrm{gph}\Omega}} w_k^{\Omega} \mathrm{N}^A_{\mathrm{2D}}\left(\mathbf{X}_{\mathrm{ref}}\right) \sum_{B=1}^{n_{\mathrm{no}\Omega}} \sum_{C=1}^{n_{\mathrm{no}\Omega}} \frac{\partial \mathrm{N}^B_{\mathrm{2D}}}{\partial \xi} \frac{\partial \mathrm{N}^C_{\mathrm{2D}}}{\partial \eta} \left(\mathbf{X}^B_{\mathrm{ref}} \times \mathbf{X}^C_{\mathrm{ref}}\right) \tag{83}$$

2.2. Conservation properties

Balance of total linear momentum. As shown in [2], the variational integrator with discrete gradient (VDG) conserves both the balance of total linear momentum and the balance of total angular momentum. The balance of total angular momentum without any DIRICHLET boundary conditions and the conservation of the balance of total linear momentum in the x, y, z-direction for one time step can be calculated by the sum over all nodes n_{ano}:

$$\underbrace{\sum_{N=1}^{n_{\mathrm{ano}}} \left[\mathbf{p}^N_{k+1} - \mathbf{p}^N_k\right]}_{\mathbf{P}_{k+1}-\mathbf{P}_k} = \mathbf{0} \tag{84}$$

Under influence of external forces and DIRICHLET boundary conditions the balance of total linear momentum is expanded to the equation:

$$\underbrace{\sum_{N=1}^{n_{\mathrm{ano}}} \left[\mathbf{p}^N_{k+1} - \mathbf{p}^N_k\right]}_{\mathbf{P}_{k+1}-\mathbf{P}_k} + \underbrace{h_n \sum_{N=1}^{n_{\mathrm{ano}}} \sum_{i=0}^{n_{\mathrm{pc}}} \boldsymbol{\lambda}^N_{k+\frac{i}{n_{\mathrm{pc}}}} - h_n \sum_{N=1}^{n_{\mathrm{ano}}} \sum_{i=0}^{n_{\mathrm{pg}}} w M_{\frac{i}{n_{\mathrm{pg}}}}(\alpha) \mathbf{F}^{\mathrm{ext},N}_{k+\frac{i}{n_{\mathrm{pg}}}}}_{\mathbf{P}^{\mathrm{bc}}_k} = \mathbf{0} \tag{85}$$

Balance of total angular momentum. In the same way follows the balance of total angular momentum without a DIRICHLET or a NEUMANN boundary.

$$\underbrace{\sum_{N=1}^{n_{\mathrm{ano}}} \left[\mathbf{q}^N_{k+1} \times \mathbf{p}^N_{k+1} - \mathbf{q}^N_k \times \mathbf{p}^N_k\right]}_{\mathbf{L}_{k+1}-\mathbf{L}_k} = \mathbf{0} \tag{86}$$

The balance of total angular momentum can be formulated with DIRICHLET boundary conditions and external forces at the boundary nodes by:

$$\underbrace{\sum_{N=1}^{n_{\text{ano}}}\left[\mathbf{q}_{k+1}^{N}\times\mathbf{p}_{k+1}^{N}-\mathbf{q}_{k}^{N}\times\mathbf{p}_{k}^{N}\right]}_{\mathbf{L}_{k+1}-\mathbf{L}_{k}}+h_n\sum_{N=1}^{n_{\text{ano}}}\sum_{i=0}^{n_{\text{pc}}}w^{\text{c}}\left[M^{\text{c}}{}_{\frac{i}{n_{\text{pc}}}}\,\mathbf{q}_{k+\frac{i}{n_{\text{pc}}}}^{N}\times M^{\text{c}}{}_{\frac{i}{n_{\text{pc}}}}(\alpha^{\text{c}})\tilde{\boldsymbol{\lambda}}_{k+\frac{i}{n_{\text{pc}}}}^{N}\right]$$
$$\underbrace{-h_n\sum_{N=1}^{n_{\text{ano}}}\sum_{i=0}^{n_{\text{pg}}}w\left[M_{\frac{i}{n_{\text{pg}}}}\,\mathbf{q}_{k+\frac{i}{n_{\text{pg}}}}^{N}\times M_{\frac{i}{n_{\text{pg}}}}\mathbf{F}_{k+\frac{i}{n_{\text{pg}}}}^{\text{ext},N}\right]}_{\mathbf{M}^{\text{ext}}}=\mathbf{0} \tag{87}$$

The external forces $\mathbf{F}^{\text{ext}}$ can only depend on the gravitational force and/or time depended forces like the external pressure from the numerical example. More details can be found in [2].

Balance of total energy. As it was already mentioned the VDG fulfills the balance of total energy. The total balance of energy can be written as the difference of the total energies for every time step. For this purpose all involved energy parts taken the form

$$\begin{aligned}E_{k+1}-E_k=&\frac{1}{2}\mathbf{p}_{k+1}\cdot{\mathbf{M}_{\text{M}}}^{-1}\cdot\mathbf{p}_{k+1}-\frac{1}{2}\mathbf{p}_{k}\cdot{\mathbf{M}_{\text{M}}}^{-1}\cdot\mathbf{p}_{k}\\&+\sum_{e=1}^{n_{\text{el}}}\int_{\mathcal{B}_0^e}\rho_0\mathbf{g}\cdot\mathbf{q}_{k+1}\,\mathrm{d}V-\sum_{e=1}^{n_{\text{el}}}\int_{\mathcal{B}_0^e}\rho_0\mathbf{g}\cdot\mathbf{q}_{k}\,\mathrm{d}V+\sum_{e=1}^{n_{\text{el}}}\int_{\mathcal{B}_0^e}\int_0^1\mathbf{F}^{\text{ext,h}}\cdot\mathbf{q}^{e,\text{d}}\,\mathrm{d}\alpha\,\mathrm{d}V\\&+\sum_{e=1}^{n_{\text{el}}}\int_{\mathcal{B}_0^e}\int_0^1\mathbf{D}\psi\,\mathrm{d}\alpha\,\mathrm{d}V\end{aligned} \tag{88}$$

The introduced $\mathbf{M}_{\text{M}}$ represents the mass matrix of the finite element system. We give the proof of the energy consistence only for the material function ψ. By the definition of:

$$\mathcal{N}^{\text{e}}=\sum_{\bar{g}=1}^{n_{\text{pg}}}h_n w_{\bar{g}}\text{tr}\left(\overset{\text{AD}}{\mathcal{P}}_{\text{M}}\left\{\bar{\mathbf{C}}^{e,\text{d}}\,,\;\mathbf{C}^{e,\text{hd}}\right\}\right)\qquad\mathcal{G}^{\text{e}}=\sum_{\bar{g}=1}^{n_{\text{pg}}}h_n w_{\bar{g}}\text{tr}\left(\overset{\text{AD}}{\mathcal{P}}_{\text{M}}\left\{\bar{\mathbf{S}}^{e}\,,\;\mathbf{C}^{e,\text{hd}}\right\}\right) \tag{89}$$

$$\mathcal{G}^{\text{e,vis}}=\sum_{\bar{g}=1}^{n_{\text{pg}}}h_n w_{\bar{g}}\text{tr}\left(\overset{\text{AD}}{\mathcal{P}}_{\text{M}}\left\{\hat{\mathbf{S}}^{e}\,,\;\mathbf{C}_{\text{i}}^{e,\text{d}}\right\}\right)\qquad\mathcal{D}^{\text{e}}=\frac{2\psi\left(\bar{\mathbf{C}}_{k+1}^{e},\mathbf{C}_{\text{i}k+1}\right)-2\psi\left(\bar{\mathbf{C}}_{k}^{e},\mathbf{C}_{\text{i}k}\right)-\mathcal{G}^{\text{e}}-2\mathcal{G}^{\text{e,vis}}}{\mathcal{N}^{\text{e}}} \tag{90}$$

it follows that:

$$\int_0^1\mathbf{D}\psi\,\mathrm{d}\alpha=\sum_{g=0}^{n_{\text{pg}}}h_n w_g\text{tr}\left(\overset{\text{AD}}{\mathcal{P}}_{\text{M}}\left\{2\frac{\partial\psi}{\partial\bar{\mathbf{C}}^{e,\text{h}}}+\mathcal{D}^{\text{e}}\bar{\mathbf{C}}^{e,\text{d}}\,,\;\frac{1}{2}\mathbf{C}^{e,\text{hd}}\right\}\right) \tag{91}$$

$$=\underbrace{\sum_{g=1}^{n_{\text{pg}}}h_n w_g\text{tr}\left(\overset{\text{AD}}{\mathcal{P}}_{\text{M}}\left\{\frac{\partial\psi}{\partial\bar{\mathbf{C}}^{e,\text{h}}}\,,\;\mathbf{C}^{e,\text{hd}}\right\}\right)}_{\frac{1}{2}\mathcal{G}^{\text{e}}}+\frac{1}{2}\mathcal{D}^{\text{e}}\underbrace{\sum_{g=1}^{n_{\text{pg}}}h_n w_g\text{tr}\left(\overset{\text{AD}}{\mathcal{P}}_{\text{M}}\left\{\bar{\mathbf{C}}^{e,\text{d}}\,,\;\mathbf{C}^{e,\text{hd}}\right\}\right)}_{\mathcal{N}^{\text{e}}} \tag{92}$$

$$=\frac{1}{2}\mathcal{G}^{\text{e}}+\frac{1}{2}\mathcal{D}^{\text{e}}\mathcal{N}^{\text{e}}=\frac{1}{2}\mathcal{G}^{\text{e}}+\left(\psi\left(\bar{\mathbf{C}}_{k+1}^{e},\mathbf{C}_{\text{i}k+1}\right)-\psi\left(\bar{\mathbf{C}}_{k}^{e},\mathbf{C}_{\text{i}k}\right)-\frac{1}{2}\mathcal{G}^{\text{e}}-\mathcal{G}^{\text{e,vis}}\right)\frac{\mathcal{N}^{\text{e}}}{\mathcal{N}^{\text{e}}} \tag{93}$$

$$=\psi\left(\bar{\mathbf{C}}_{k+1}^{e},\mathbf{C}_{\text{i}k+1}\right)-\psi\left(\bar{\mathbf{C}}_{k}^{e},\mathbf{C}_{\text{i}k}\right)-\mathcal{G}^{\text{e,vis}} \tag{94}$$

$$=\psi\left(\bar{\mathbf{C}}_{k+1}^{e},\mathbf{C}_{\text{i}k+1}\right)-\psi\left(\bar{\mathbf{C}}_{k}^{e},\mathbf{C}_{\text{i}k}\right)+D^{\text{int}}\quad\square \tag{95}$$

3. Numerical examples

The presented numerical example is a rotor that is fixed at the shaft, see Fig. 3. Different material domains are used for the rotor. Each material domain has its own fiber direction and material parameters, see Fig. 4, and Tables 1 and 2.

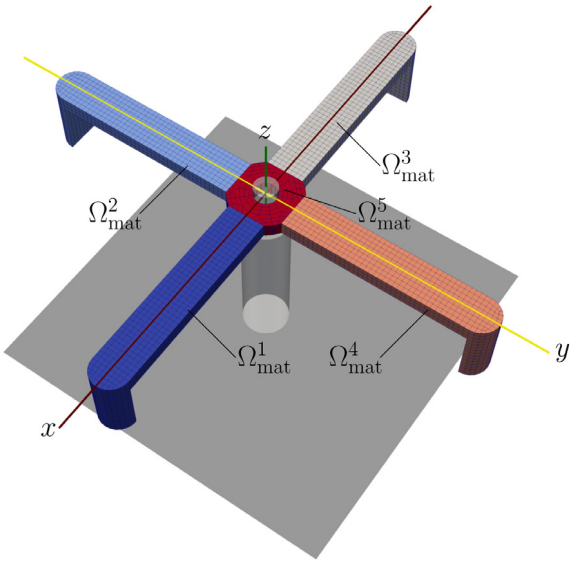

Fig. 3. Material domains on the parted rotor.

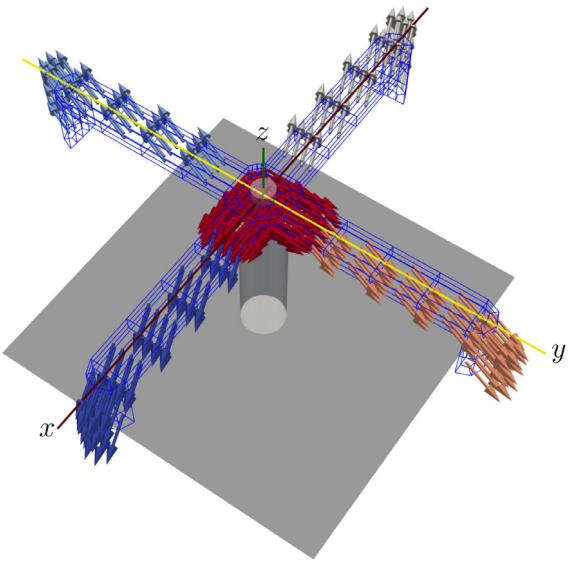

Fig. 4. Fiber directions on the parted rotor.

Table 1
Material functions for the material domains and unnormalized fiber directions.

	Ω^1_{mat}	Ω^2_{mat}	Ω^3_{mat}	Ω^4_{mat}	Ω^5_{mat}
ψ^{ela}	ψ_1^{ela}	ψ_1^{ela}	ψ_2^{ela}	ψ_2^{ela}	ψ_1^{ela}
ψ^{vol}	ψ_1^{vol}	ψ_2^{vol}	ψ_1^{vol}	ψ_2^{vol}	ψ_1^{vol}
ψ^{vis}	ψ_1^{vis}	ψ_1^{vis}	ψ_2^{vis}	ψ_2^{vis}	ψ_1^{vis}
$\psi^{\text{vis,vol}}$	$\psi_1^{\text{vis,vol}}$	$\psi_2^{\text{vis,vol}}$	$\psi_1^{\text{vis,vol}}$	$\psi_2^{\text{vis,vol}}$	$\psi_1^{\text{vis.vol}}$
ψ^{aniso}	$\mathbf{a}_1 = \begin{bmatrix}2.5\\1.0\\0.0\end{bmatrix}$	$\mathbf{a}_1 = \begin{bmatrix}-1.0\\-2.5\\0.0\end{bmatrix}$	$\mathbf{a}_1 = \begin{bmatrix}-2.5\\-1.0\\0.0\end{bmatrix}$	$\mathbf{a}_1 = \begin{bmatrix}2.5\\1.0\\0.0\end{bmatrix}$	$\mathbf{a}_1 = \begin{bmatrix}2.5\\1.0\\0.0\end{bmatrix}$ $\mathbf{a}_2 = \begin{bmatrix}2.5\\1.0\\0.0\end{bmatrix}$

Only for material domain number five two fiber directions are used. The material functions for the material domains can be found in Appendix C.1. The fixed rotor is loaded by a pressure on each blade. The end of the

Table 2
Material parameters for the material domains.

Domains	μ	$\bar{\mu}$	λ	n	μ^{vis}	$\bar{\mu}^{\text{vis}}$	λ^{vis}	n	V^{dev}	V^{vol}	ϵ_1	ϵ_2	ρ
Ω^1_{mat}	3e8	–	5e9	–	5e8	–	1e8	–	1e8	3e9	8e10	2.0	940
Ω^2_{mat}	5e8	–	8e9	4	4.5e9	–	1e8	4	3e8	3e9	8e10	4.0	940
Ω^3_{mat}	2e8	1e8	2e9	–	1.5e9	0.5e9	2e8	–	1e8	5e9	8e10	2.0	940
Ω^4_{mat}	7e8	5e8	7e9	6	4.5e9	3e9	3e8	6	5e8	7e9	8e10	8.0	940
Ω^5_{mat}	15e8	–	15e9	–	12.5e9	–	11e8	–	13e8	13e9	2e11	4.0	940

blade is led in the z-direction. The pressure starts at a value of zero and is increased over time to a constant value in the z-direction, compare Fig. 5. The colored planes at the end of the blades describe the vertical loading through the pressure $p_0 = 8e8$. During the first second, the pressure is increased linearly. After this time, the pressure is hold constant. The simulation parameters are set to:

$$h_n = 0.01\ \text{s} \qquad T_{\text{end}} = 2\ \text{s} \qquad n_{\text{pg}} = 2$$
$$TOL_{\text{ener}} = 1e-7 \qquad TOL_{\text{Ci}} = 1e-5 \qquad g = 9.81$$
$$n_{\text{el}} = 5952 \qquad \text{element type} = \text{H27} \qquad n_{\text{dof}} = 370432$$

$$p(t) = \begin{cases} \dfrac{p_0}{1.0\ \mathrm{s}} t & \text{for } 0\ \mathrm{s} < t <= 1.0\ \mathrm{s} \\ p_0 & \text{for } t > 1.0\ \mathrm{s} \end{cases} \qquad p_0 = 1.2e8$$

The simulation starts in the undeformed configuration with $t = 0$ s with 5952 fully quadratic hexahedral elements. The quadratic approximation in time with the number of elements and the fixed boundary nodes leads to a total number of degrees of freedom $n_{\mathrm{dof}} = 370432$.

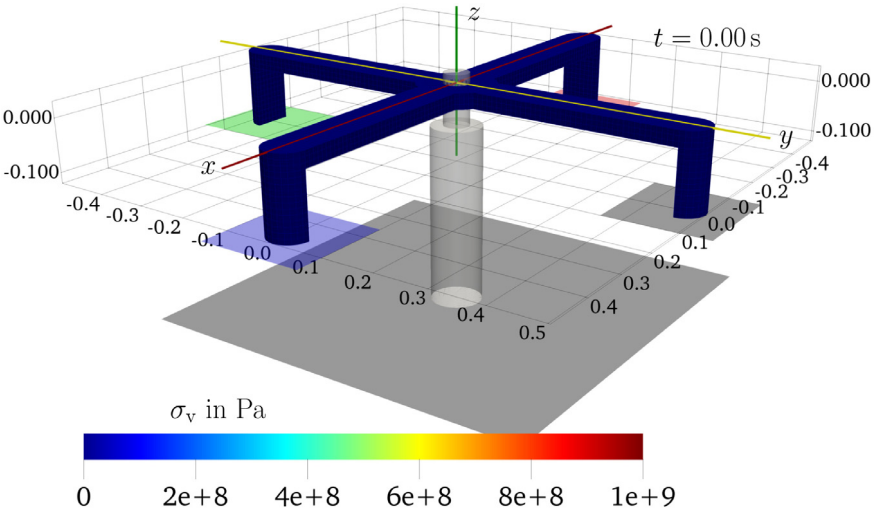

Fig. 5. Reference configuration of the parted rotor for 5952 H27 elements and $n_{\mathrm{pg}} = 2$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The deformed rotor after $t = 0.5$ s is shown in Fig. 6 with the VON MISES stress σ_{v}. Due to the fibers, the blades are not only deformed in z-direction but twist along the orientation. This behavior can also be verified in Figs. 7 and 8. The forces created by the time depended pressure can be seen as yellow arrows under the ends of the blades in the figures. At the time $t = 1.0$ s they reach their maximum value for the given p_0. The maximum

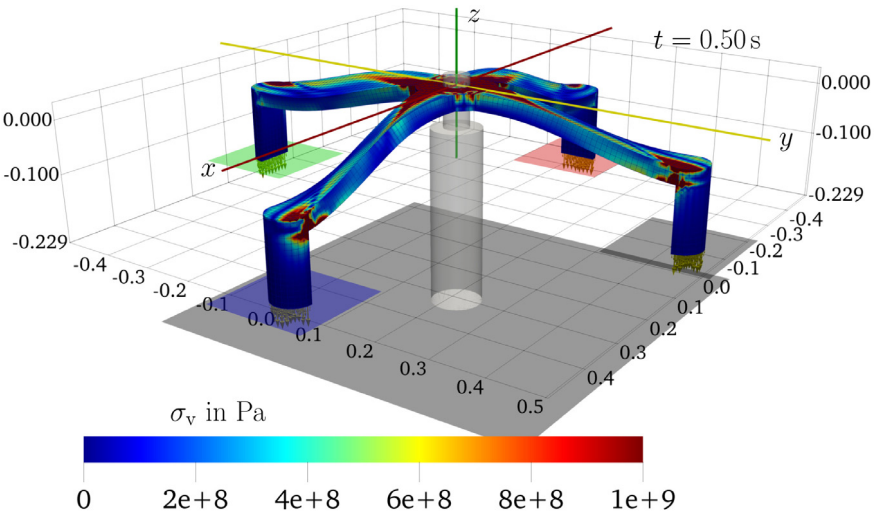

Fig. 6. Current configuration of the parted rotor for 5952 H27 elements and $n_{\mathrm{pg}} = 2$ at $t = 0.5$ s. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

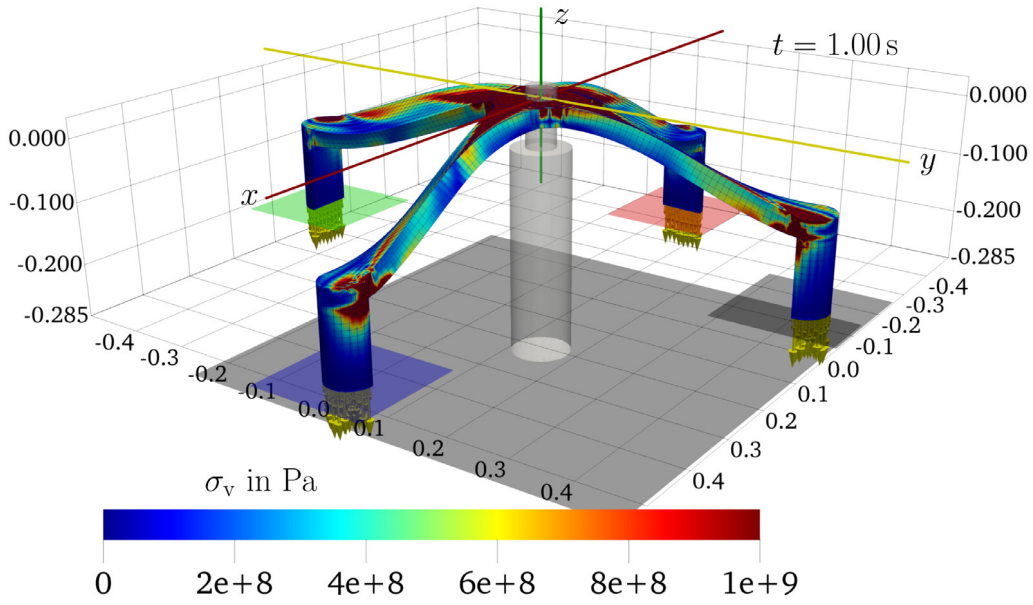

Fig. 7. Current configuration of the parted rotor for 5952 H27 elements and $n_{pg} = 2$ at $t = 1.0$ s. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

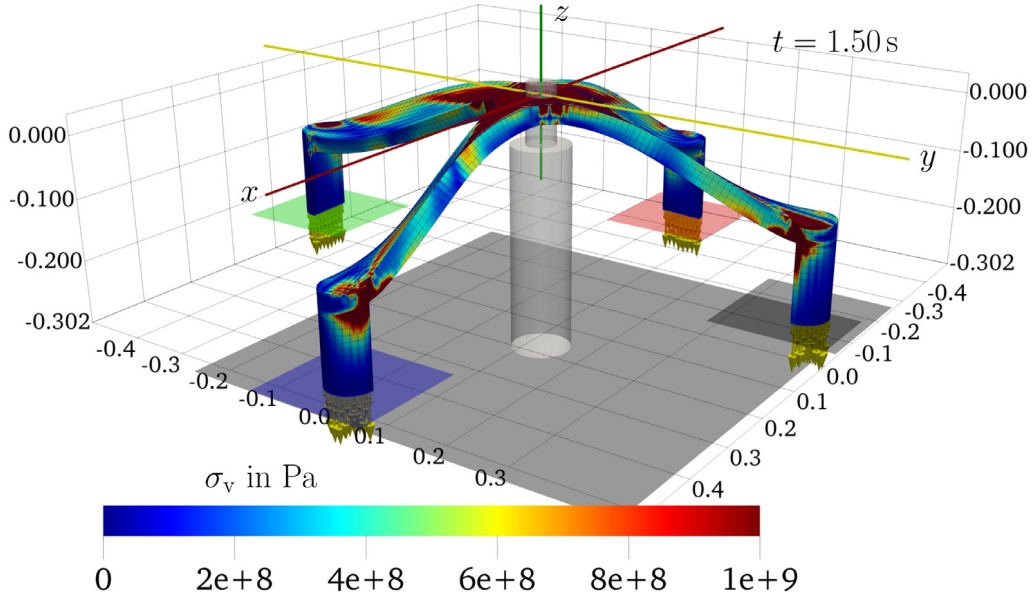

Fig. 8. Current configuration of the parted rotor for 5952 H27 elements and $n_{pg} = 2$ at $t = 1.5$ s. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

values for the VON MISES stresses σ_v will be reached in the material domain Ω^5_{mat} and in the corners of the blades.

After the time $t = 1.0$ s, the pressure is constant but the rotor deformation keeps growing in the z-direction. This results from the viscoelastic material formulation and is known as relaxation of a material under a constant load, compare Figs. 8 and 9. The maximum values in the z-direction for the blades can also be found in Fig. 17. In the next subsection, the numerical conservation properties are shown for this simulation.

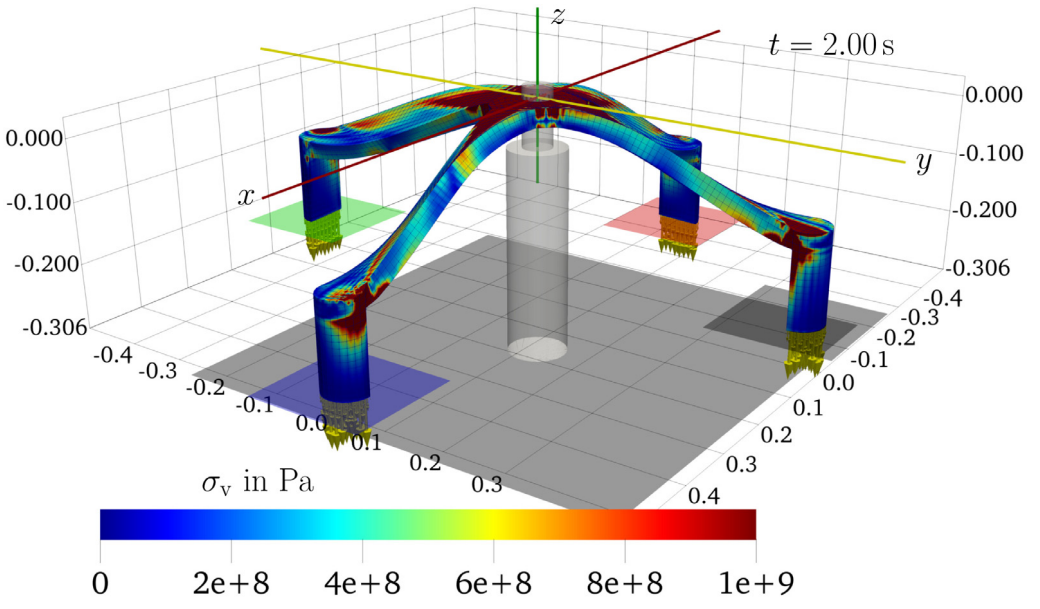

Fig. 9. Current configuration of the parted rotor for 5952 H27 elements and $n_{\mathrm{pg}} = 2$ at $t = 2.0$ s.

3.1. Conservation properties

By definition, every value of the total balance of angular momentum should be in the order of the tolerance TOL_{q}.

$$TOL_{\mathrm{q}} = \max\left(\min\left(\|\mathbf{R}_k^m\|\Big|_{m=0}^{n_{\mathrm{new}}}\right)\Big|_{k=0}^{n_{\mathrm{ti}}}\right) \tag{96}$$

This is fulfilled for linear approximation in time with $n_{\mathrm{pg}} = 1$, see Fig. 10. For the quadratic approximation in time with $n_{\mathrm{pg}} = 2$, Fig. 11, this can be regarded as fulfilled likewise. The small peaks over the value 1 in Fig. 11 depend

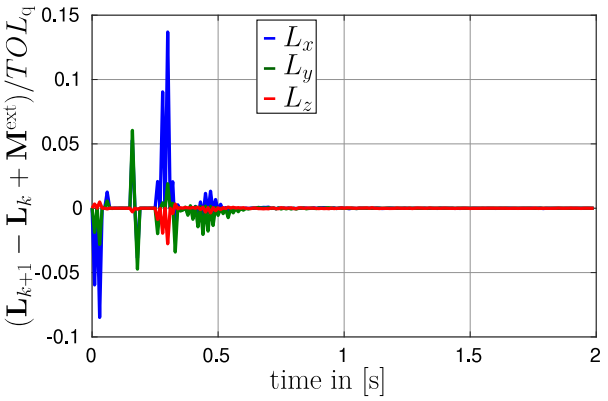

Fig. 10. Balance of total angular momentum for the rotor, H27 with $n_{\mathrm{el}} = 5952$, $n_{\mathrm{dof}} = 185216$ and $n_{\mathrm{pg}} = 1$.

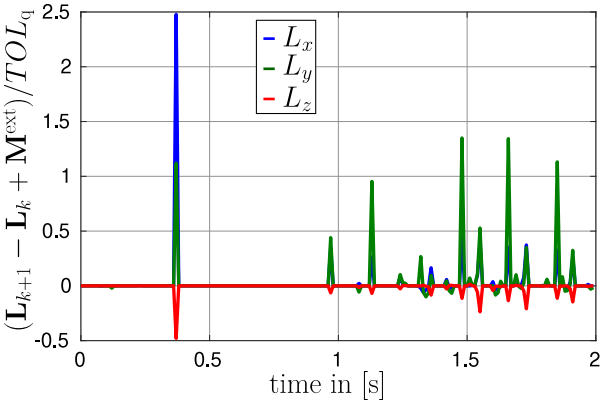

Fig. 11. Balance of total angular momentum for the rotor, H27 with $n_{\mathrm{el}} = 5952$, $n_{\mathrm{dof}} = 370432$ and $n_{\mathrm{pg}} = 2$.

on NEWTON criterion in the time stepping scheme. In order to guarantee the fulfillment of the total balance of energy, a criterion based on the energy is used for the NEWTON iterations. This criterion does not check the error of the global residual, but it guarantees the fulfillment of the balance of the total energy. Therefore, it may happen that the numerical effects through addition and subtractions disturb the balances. This only occurs for the balance

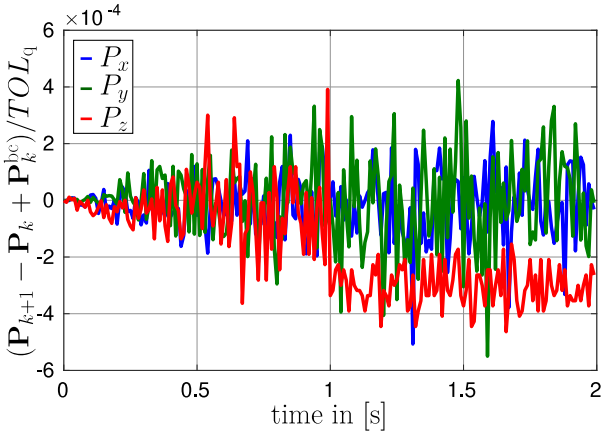

Fig. 12. Balance of total linear momentum for the rotor, H27 with $n_{\mathrm{el}} = 5952$, $n_{\mathrm{dof}} = 185216$ and $n_{\mathrm{pg}} = 1$.

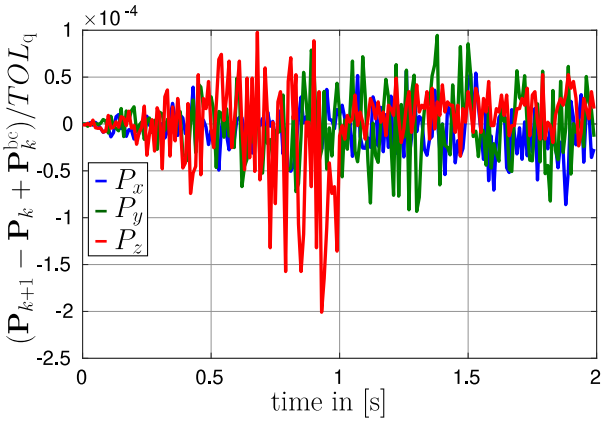

Fig. 13. Balance of total linear momentum for the rotor, H27 with $n_{\mathrm{el}} = 5952$, $n_{\mathrm{dof}} = 370432$ and $n_{\mathrm{pg}} = 2$.

of total angular momentum but not for the balance of total linear momentum. In Fig. 12, the balance of total linear momentum for linear approximation in time is shown. The quadratic approximation in time for the balance of total linear momentum is plotted in Fig. 13. Both balances are fulfilled and the error in the balances is significantly smaller than the error in the balance of total angular momentum. The balance of total energy should be fulfilled by the construction of the time stepping scheme. For a NEWTON criterion that only checks the global residual in the time stepping scheme it may happen that there also arise some peaks. An energy consistent NEWTON criterion can be formulated by calculating the total balance of energy in every NEWTON step. This includes the update of the linear momentum for the kinetic energy, the calculation of the internal energy, the calculation of the dissipation for a viscoelastic material, and the calculation of the non-conservative work. The conservations of the balances of the total energies are listed in Figs. 14 and 15. Every single energy term is shown in Fig. 16. The addition of every single energy term leads to the total energy. One considers the position in z-direction for the end of the blades in Fig. 17. It can be verified that after the time point 1 s the slope wears off. The displacement in the z-direction only follows by the relaxation of the material under a constant force. This behavior is also shown in Fig. 16 for the internal dissipation which reached a nearly constant value at $t = 2$ s.

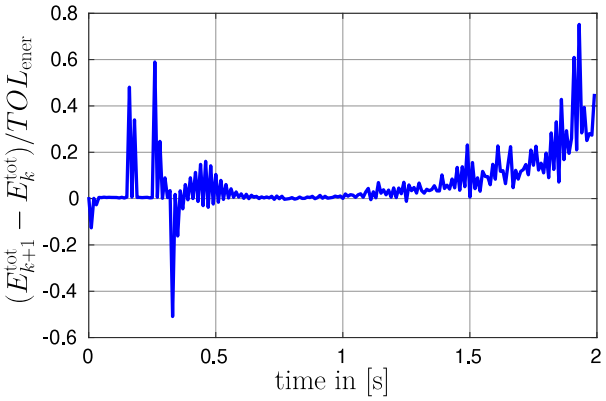

Fig. 14. Balance of total energy for the rotor, H27 with $n_{\mathrm{el}} = 5952$, $n_{\mathrm{dof}} = 185216$ and $n_{\mathrm{pg}} = 1$.

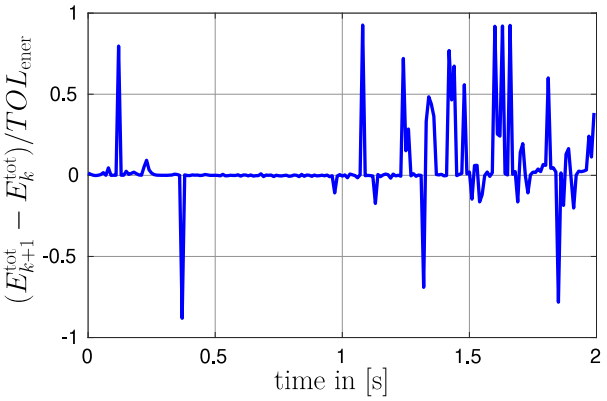

Fig. 15. Balance of total energy for the rotor, H27 with $n_{\mathrm{el}} = 5952$, $n_{\mathrm{dof}} = 370432$ and $n_{\mathrm{pg}} = 2$.

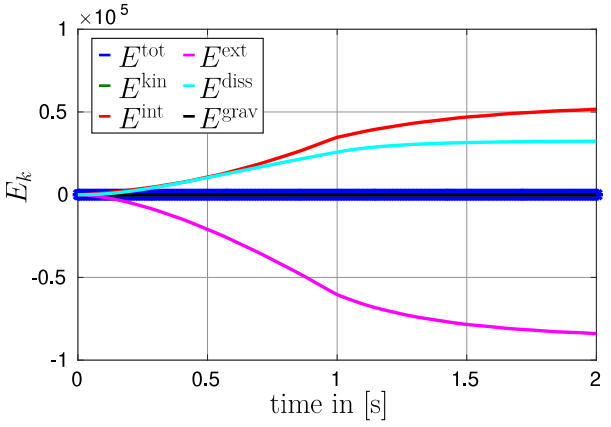

Fig. 16. Energies for the rotor, H27 with $n_{\text{el}} = 5952$ and $n_{\text{dof}} = 370432$ and $n_{\text{pg}} = 2$.

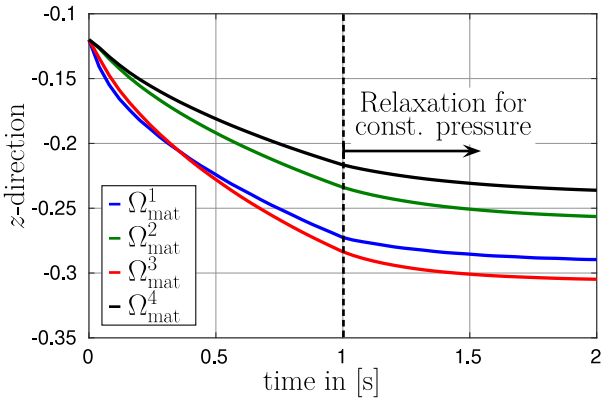

Fig. 17. Maximum shift in z-direction for the rotor, H27 with $n_{\text{el}} = 5952$ and $n_{\text{dof}} = 370432$ and $n_{\text{pg}} = 2$.

3.2. Convergence order of the time stepping scheme

In this section the convergence order in time is presented. The rotor is used again as a model problem, but only one material configuration is used for the whole rotor this time. In this example no DIRICHLET or NEUMANN boundary conditions are applied. The free flying rotor is simulated with a translational and angular initial velocity.

$$\mathbf{v}_0 = \begin{bmatrix} 0 & 0 & 10 \end{bmatrix}^{\mathrm{T}} \qquad \mathbf{X}_\omega = \begin{bmatrix} 5 & 0 & 0 \end{bmatrix}^{\mathrm{T}} \qquad \boldsymbol{\omega} = \begin{bmatrix} -2 & 2 & 3 \end{bmatrix}^{\mathrm{T}} \tag{97}$$

The simulation was performed for $T_{\text{end}} = 0.5$ s with the material from Table 3. The material function $\psi = \psi_1^{\text{ela}} + \psi_1^{\text{ela,vol}} + \psi_1^{\text{vis}} + \psi_1^{\text{vis,vol}} + \psi_1^{\text{aniso}}$ with the unnormalized direction $\mathbf{a}_1 = [2.5 \;\; 1.0 \;\; 0]^{\mathrm{T}}$ was used. In Fig. 18 the convergence order in time for the position vector $\mathbf{q}$ is shown. The convergence order for the linear momentum, the VON MISES stress, and the internal variable are shown in Figs. 19–21. On the abscissa the time step size from $h_n = 2^{-n}$ for $n = 7, \ldots, 18$ is presented. On the ordinate the relative error ϵ_z is plotted. The error was calculated by:

$$\epsilon_z = \frac{\left\| \mathbf{z}_{n_{\text{ti}}} - \mathbf{z}_{\text{ref}} \right\|}{\left\| \mathbf{z}_{\text{ref}} \right\|} \qquad z \in \{q, p, C_{\text{i}}, S\}. \tag{98}$$

The reference solution was chosen individually for every approximation order.

$$\begin{aligned}
n_{\text{pg}} = 1: &\quad \mathbf{q}_{\text{ref}} = \mathbf{q}(h_{18}) &\quad \mathbf{p}_{\text{ref}} = \mathbf{p}(h_{18}) &\quad \mathbf{C}_{\text{iref}} = \mathbf{C}_{\text{i}}(h_{18}) &\quad \mathbf{S}_{\text{ref}} = \mathbf{S}(h_{18}) \\
n_{\text{pg}} = 2: &\quad \mathbf{q}_{\text{ref}} = \mathbf{q}(h_{15}) &\quad \mathbf{p}_{\text{ref}} = \mathbf{p}(h_{16}) &\quad \mathbf{C}_{\text{iref}} = \mathbf{C}_{\text{i}}(h_{15}) &\quad \mathbf{S}_{\text{ref}} = \mathbf{S}(h_{17}) \\
n_{\text{pg}} = 3: &\quad \mathbf{q}_{\text{ref}} = \mathbf{q}(h_{14}) &\quad \mathbf{p}_{\text{ref}} = \mathbf{p}(h_{14}) &\quad \mathbf{C}_{\text{iref}} = \mathbf{C}_{\text{i}}(h_{13}) &\quad \mathbf{S}_{\text{ref}} = \mathbf{S}(h_{15})
\end{aligned}$$

It can be verified that all quantities reach their theoretical order of convergence. One characteristic is that the convergence for the linear approximation starts much later than for the higher order approximation. In fact, much more time steps are needed to reach this area. An additional interesting fact is the offset of the error. Every time, the higher order approximation starts with a smaller error than the linear approximation.

Table 3
Material parameters for the material domains.

Material parameters	μ	$\bar{\mu}$	λ	n	μ^{vis}	$\bar{\mu}^{\text{vis}}$	λ^{vis}	n	V^{dev}	V^{vol}	ϵ_1	ϵ_2	ρ
$\Omega^{1-5}_{\text{mat}}$	3e6	–	5e7	–	5e6	–	1e6	–	1e6	3e7	4e7	2.0	940

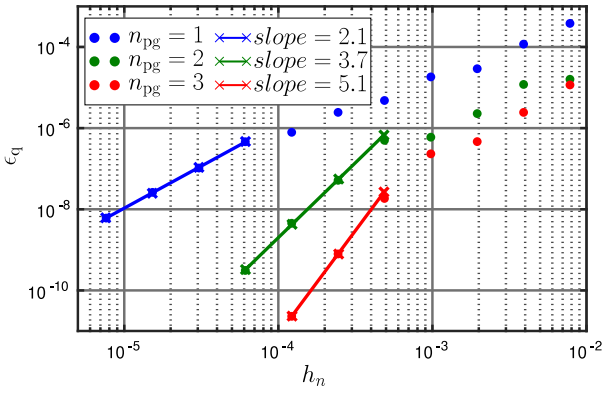

Fig. 18. Convergence order for the position vector.

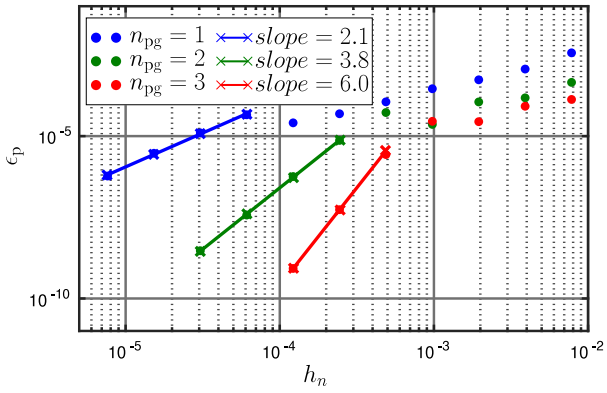

Fig. 19. Convergence order for the linear momentum.

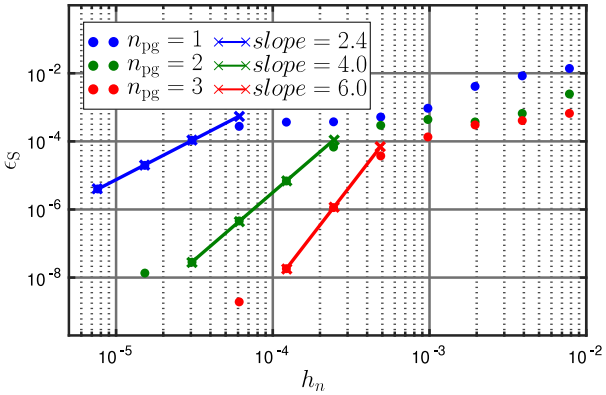

Fig. 20. Convergence order for the von MISES stress.

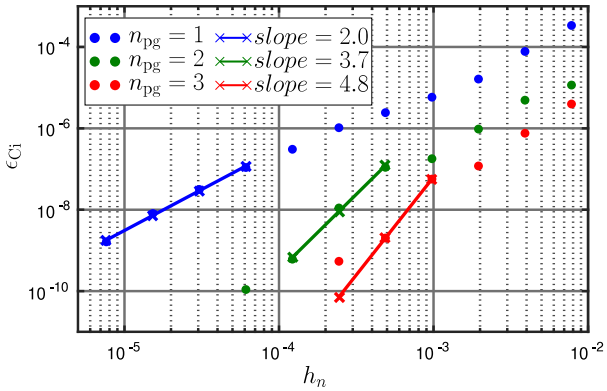

Fig. 21. Convergence order for the internal variable.

3.3. Conclusion to the time stepping scheme

The aim of this paper was to develop a time integration scheme based on a variational integrator that combines the external forces and boundary conditions for viscoelastic material formulations with fibers on different material domains. The application of a variational integrator was motivated by the physical structure of a finite element simulation and their robustness for stiff problems. These properties were extended to an integrator that does not only preserve the total balance of angular momentum and the total balance of linear momentum but also the balance of total energy. In Section 1.1, the general time approximation was introduced for a system with boundary conditions and a viscoelastic material formulation that depends on internal variables. In the next section, the idea of the energy consistent time stepping scheme was proposed by using discrete gradients. The main point is to formulate a condition for the energy that depends on the internal energy and the dissipative part that a discrete gradient can fulfill. D'ALEMBERTS principal can be used to formulate a consistent time stepping scheme in a variational frame work. The time stepping scheme can easily be extended to an arbitrary approximation order in time. This can be used to create an optimal balance between both the accuracy and the simulation time depended on the system size.

The theoretical properties for the balance of total linear momentum, the balance of total angular momentum and the balance of total energy has been demonstrated by the numerical examples. The application of the viscoelastic material formulation with fiber directions on different material domains can cover a large class of problems.

In addition to the quality of the solution, the duration of the calculation plays a decisive role today. In many cases solutions with maximum accuracy are not required but results calculated with maximum required accuracy.

Hardware development always brings new possibilities for the implementation of new algorithms. Based on the previous presented time stepping scheme a possibility for an implementation on a GPU will be presented in the next section.

4. GPU implementation

Compared to a CPU, a GPU is specialized in performing the same instruction on multiple data sets in parallel, which leads to a high level of computational parallelism. In addition, a GPGPU allows the execution of self-written code, e.g. written in C/C++, on the GPU using a programming model which is specialized on the GPU, so programming code can be easily ported on the GPU. However, GPUs suffer from their limited amount of usable memory space, e.g. 5GB on the NVIDIA Tesla K20c, compared to the available RAM on a computer system, e.g. 64GB in our testing scenarios. Nevertheless, both hardware resources can be combined for achieving maximum speedup, as shown in Fig. 22. In this section we present a GPU implementation based on a pipeline design for the calculation of the global tangent $\mathbf{T}_{\mathrm{q}}^{\mathrm{glo},e}$. Furthermore, we optimize and extend the GPU implementation so the CPU is used as well for the calculation and we overcome the memory bound with further approaches. First, an overview about the hardware architecture is provided. Then, the pipeline design is discussed in the GPU implementation. Next, optimizations are explained in detail. Finally, the GPU implementation is compared to the CPU-only implementation and the findings are discussed.

4.1. GPU architecture

For this implementation a single GPU accelerator NVIDIA TESLA K20C, or K20c, is used, see Table 4. To be able to run C code on the GPU, the Cuda framework is used. The Compute Unified Device Architecture, or so-called Cuda, is an architecture for parallel calculations developed by NVIDIA [6]. For understanding the challenges and possibilities of Cuda, a short view on the GPU architecture is provided in this section. First, the concept of threads is introduced and then an overview on the memory architecture is provided.

4.1.1. Cuda kernel and threads

In terms of processors, a sequence of computational tasks and operations can be described as a thread. In contrast to the CPU, a GPU is capable of handling a high amount of threads, e.g. up to 2048 threads per multiprocessor on the K20c. Anyhow, a GPU thread is not as powerful as a CPU thread. Some examples for this issue will be provided in subsequent sections.

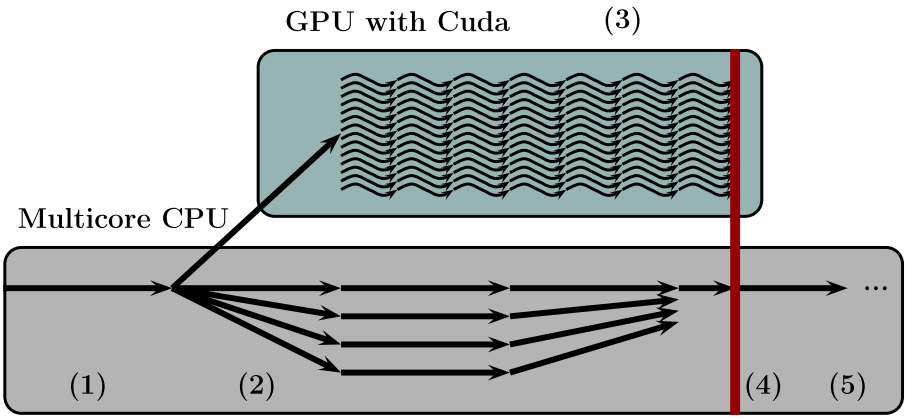

Fig. 22. Parallel processing of CPU and GPU: (1) a single threaded CPU starts the GPU threads; (2) the CPU moves on with its own tasks; (3) the GPU operates in parallel to the CPU; (4) the CPU waits until the GPU has finished; (5) the CPU gathers the results and continues work.

Table 4
Datasheet for NVIDIA TESLA K20C.

GPU accelerator	
Model	NVIDIA TESLA K20C
Chip	KEPLER GK110
Performance	
Peak double-precision floating point performance	1.17 Tflops
Peak single-precision floating point performance	3.52 Tflops
Number of Cuda cores	2496
Memory	
Memory size (VRAM)	5 GB
Memory bandwidth	208 Gbytes/s
Multiprocessors and threads	
Maximum number of multiprocessors	15
Threads per warp	32
Maximum threads per Block	1024
Maximum warps per multiprocessor	64
Maximum threads per multiprocessor	2048
Maximum blocks per multiprocessor	16
On-chip memory	
Maximum 32-bit registers per multiprocessor	65536
Maximum registers per thread	255
Level 1 cache	16 KB
Maximum shared memory per multiprocessor	48 KB
Read-only cache	48 KB

In Cuda, a Cuda kernel is a self-written GPU function that might represent a single task, an algorithm or a smaller part of it, e.g. a single vector–vector operation or a matrix–matrix operation. After the start, the kernel function is executed multiple times on the GPU where each copy is handled by a thread. As shown in Fig. 23, each thread is indexed by a unique ID. These threads are arranged in blocks of up to 1024 threads, where each block is defined by a block ID. Threads of a single block are performed in groups of up to 32 threads at once on K20c, which is called a warp. However, all threads of a single warp have to perform the same instruction per clock cycle. In if-else constructions, for example, it is possible that not all threads of a warp execute the same branch. In this case all the threads of a warp which fulfill the if-clause execute the if-branch at first while the others are stalled. Only afterwards the else-branch is executed in a second cycle, reducing the overall performance, see [7]. The concept of threads and warps in Cuda follows the Single Instruction, Multiple Threads (SIMT) principle, which is based on the SIMD principle.

4.1.2. Memory architecture

In GPUs memory is ordered in a hierarchy, as shown in Table 5, where each type varies in its location on the GPU, the size and access times. Each thread has an exclusive set of registers for variables. If a higher amount of registers is required than available, e.g. for large arrays and structs, then the registers are spilled into the local memory. This kind of memory is also thread exclusive and resides on the VRAM. However, in comparison to registers, its access time is considerably higher. With the shared memory each thread of a single block is able to exchange data with each other. Compared to the VRAM its access time is almost as low as the access time for registers. Anyhow, only the Global memory "VRAM" allows to exchange data between all threads of a kernel. In addition, only the VRAM is persistent, i.e. its data can be read and written across multiple kernels while the other memory types only exist for the duration of a single kernel. Therefore, VRAM is also the slowest memory type in terms of accessing. This drawback, however, can be reduced by the read-only memory, which caches read operations, e.g. reading a large array, and does not allow any modifications of the data. The actual reached access time of a memory type, however, may vary depending on caching effects, access size and instruction order as explained in [14].

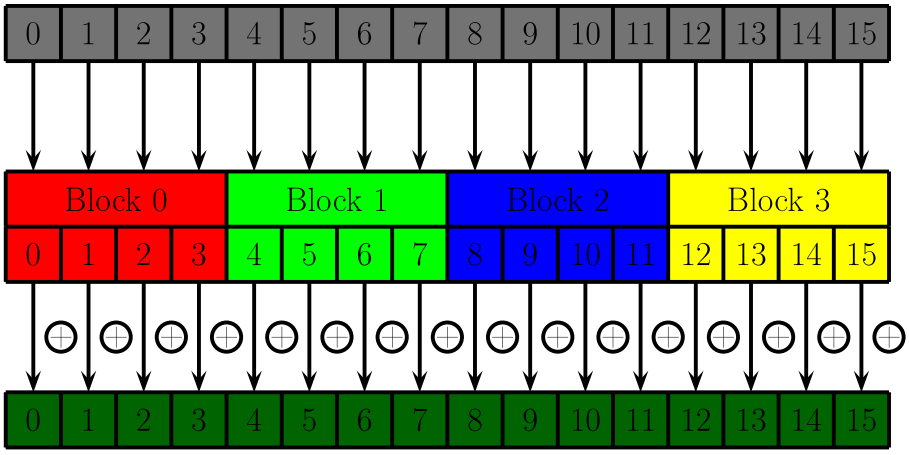

Fig. 23. Cuda thread model with threads and blocks of threads.

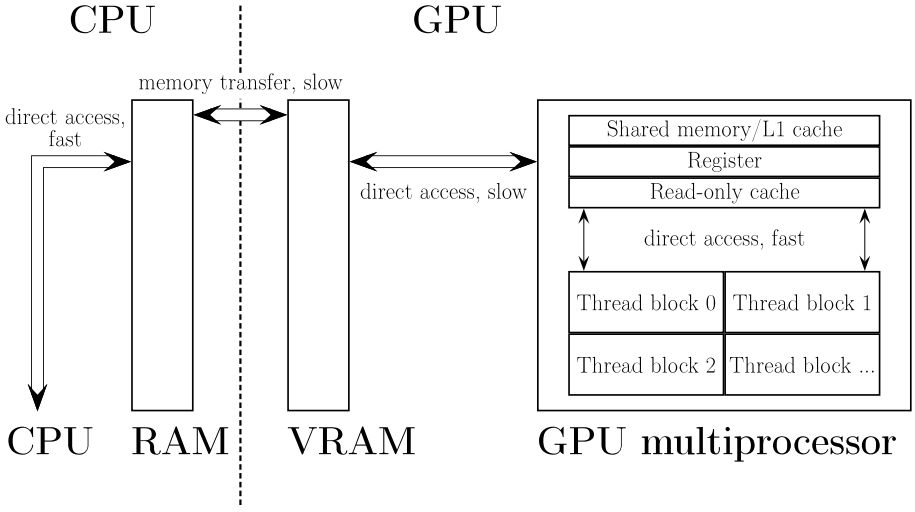

Fig. 24. Cuda memory concept.

Table 5
Different types of memory according to [6].

Memory type	Access Type	Visibility	Access speed
Registers	read and write	Thread only	fastest
Local memory	read and write	Thread only	on the VRAM, slightly faster than VRAM due to optimizations
Shared memory	read and write	Block-wide	almost as fast as registers
Read-only/texture memory	read-only	global	faster than VRAM
Global memory / VRAM	read and write	global	slowest

Data required from the RAM, i.e. data which was calculated by the CPU, cannot be accessed directly from the GPU, and vice versa, as shown in Fig. 24. Therefore, data has to be exchanged between the VRAM and the RAM. This is done by memory transfer operations which are provided by Cuda itself. They allow RAM to VRAM, VRAM to RAM and VRAM to VRAM transfers with a maximum bandwidth of 6 GB/s in our examples.

4.2. Implementation

The main goal of the Cuda implementation is gaining a speedup in comparison to the CPU-only implementation. To achieve this, a pipeline design is introduced which consists of multiple stages. Those stages are designed to exploit the SIMT architecture to gain high efficiency. In addition, the GPU pipeline does not interfere the CPU after its start so the calculations can be distributed between the CPU and GPU. Both the pipeline designs and stage designs, will be discussed in the following section.

4.2.1. Parallel loop

The calculation of $\mathbf{T}_{\mathrm{q}}^{\mathrm{glo},e}$ is done element-wise in the C implementation, while a for-loop iterates over all elements. Those elements can be seen independently due to the fact that single elements do not depend on each other. With the help of libraries which are specialized in parallelizing, e.g. OpenMP, this loop can be transformed into a parallel-for-loop where single iterations are performed in parallel on the CPU with multithreading.

In the view of Cuda, this approach can be transferred to the GPU so iterations of the element loop are performed in parallel on the GPU. For doing this, a single iteration is described inside a Cuda kernel following the CPU implementation. And after its start, the kernel starts with a sufficient number of threads to cover all elements.

The advantage of this approach is the easy porting of the C implementation to Cuda. Since the C code can be copied from the C implementation with minor adjustments for the GPU, e.g. memory transfers from and to the GPU for the residual and tangent.

However, this concept leads to huge and complex kernels which suffer negative impacts on the performance due to the architecture of the GPU. Therefore, this implementation might be slower than a parallel CPU approach. The main reason for this behavior is the memory architecture of the GPU. Every element needs a specific amount of memory for data, e.g. for scalar values and matrices. In fact, each GPU only provides a limited amount of very fast on-chip registers while larger amounts of data, which exceed the available amount of registers, are spilled into the local memory. Thus, those memory accesses are the bottleneck of graphic cards and lead into a serious loss of performance as it was observed in [17]. Therefore, another approach is required.

4.2.2. Pipeline design

In contrast to an implementation of a complex single kernel as explained previously, a pipeline design is introduced for overcoming the performance challenges. The C implementation is therefore divided into smaller parts and are adjusted to avoid the memory bottleneck and to exploit further possibilities of improvements. These small parts implemented on the GPU are called stages and are handled by a pipeline, which ensures the correct order of kernels and transition of data from one stage to the next one. This idea of a GPU pipeline instead of a parallel-loop implementation is discussed in the following subsection.

4.2.3. Pipeline stage

Each stage consists of either a small Cuda kernel (kernel stage) or a memory transfer (memory stage) between the RAM and VRAM and describes a small part of the C implementation. A single kernel stage is divided into five phases. A memory stage, however, can be reduced to three phases due to its lack of calculations:

1. Starting phase (kernel and memory);
   start of the kernel or memory transfer after waiting for the finish signal of previous stages,
2. Reading phase (kernel only);
   loading the input values from the main memory into the kernel,
3. Calculation phase (kernel only);
   performing calculations depending on the input values,
4. Writing phase/Transfer phase (kernel and memory);
   transferring results from the VRAM to the RAM (kernel and memory) or vice versa (memory only),
5. Signal phase (kernel and memory);
   indicating the completion by sending a signal.

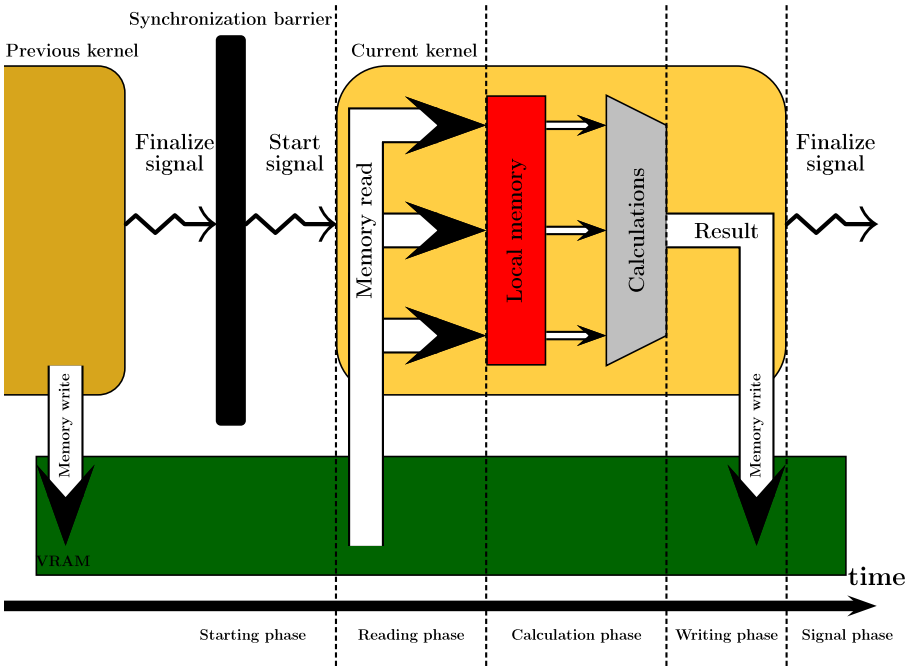

Fig. 25. Scheme of a kernel pipeline stage.

Fig. 25 provides a detailed scheme on a kernel pipeline stage. In the starting phase of a stage its stage function is called. In a kernel stage this is done by calling the Cuda kernel function from the CPU. The input and result values, i.e. array pointers, are passed as function parameters to the Cuda function. Those arrays should have a sufficient number of entries for each thread. With additional optimizations, the amount of required memory can even exceed the available memory space of the GPU, see Section 4.3.3.

During a memory stage, the Cuda built-in function "CudaMemcpy[transfer direction]" is called. The function arguments are a pointer to the target array, to the source array, the size of the transferred bytes, and the direction as arguments, i.e. from the RAM to the VRAM or vice versa, see [6].

The kernel function call can be attached to a Cuda stream as well as to the transfer function. A stream is also attached to a Cuda event. If the current kernel depends on previous stages, i.e. its input values are the results of other ones, then the start is stalled by the stream until the other kernels are finished and have sent their event signal to the stream. An event signal is only sent when the associated stream finished its function.

Next, during a kernel stage, the kernel loads the necessary input values into the kernel from the VRAM. Since an array is passed, which contains the values for each thread, the right entries for the current thread have to be extracted. This is done by calculating the memory indices from the thread ID. With the help of those indices, the required input values are read from the VRAM and are saved to local or shared memory.

Once the memory is set up for each thread inside a kernel stage, its calculations are performed during the calculation phase. Intermediate results are saved back to local or shared memory while final results can either be saved locally or directly to the VRAM, depending on their further usage.

The calculation phase is followed by the writing phase. During this phase the results, which were stored locally, are written back to the VRAM. However, this phase can be skipped if no results have been saved inside the kernel.

Nevertheless, inside a memory stage the transfer phase follows right after the starting phase. In this phase the memory is transferred either from the RAM to the VRAM or vice versa.

The final phase of both kernel stage and memory stage is the signal phase. Once the kernel or the transfer finishes, the kernel sends an event signal. This allows further stages to start, if they depend on the current one.

4.3. Optimizations techniques

The pipeline stages follow some rules for avoiding lacks in performance, gathered from:

- A stage has to be as simple as possible and should only perform a few calculations.
- Memory operations on the VRAM should be as minimal as possible.
- Each stage allocates sufficient exclusive VRAM space for its result values.
- Input values should be stored in the kernel in registers.
- Stages should be independent from each other, so they can run in parallel or overlap.
- Depending stages have to start in the right order to avoid inconsistencies.

These optimizations have major advantages as well as minor disadvantages. However, the most crucial improvements are gained by small and less complex kernels. On one hand, the register usage inside such kernels is reduced drastically since they perform only a few calculations so that less memory is required and registers are truly used since register spilling is avoided. On the other hand, a huge amount of VRAM space is required for saving the results of each stage so they can be passed to another one independently. In the same manner, additional memory operations are required. However, these disadvantages are overcome by the exploitation of various memory features of Cuda and further implementation improvements, which are discussed in the following section.

4.3.1. Memory optimizations

The most significant optimization is the usage of caching, see [19]. If a kernel does not modify its input values, i.e. those values are read-only, Cuda caches them so the reading operations for these values do not suffer high memory access latencies during the reading phase. Furthermore, the overall number of memory accesses on the hardware can be reduced due to a proper memory alignment. In Cuda memory transfers are performed half-warp-wise and coalesced due to the SIMT concept. This means that when the threads of a half-warp read from an array in global memory, a word of up to 128-byte is read in one transaction and shared between these threads. However, if not all required entries for every thread in the half-warp were loaded, e.g. due to non-linear memory alignment, further words are being loaded until all necessary entries are provided, which causes a negative impact on memory access time. In other words, a linear memory alignment where elements are ordered thread-wise, i.e. thread 0 accesses element 0, thread 1 accesses element 1 etc., decreases the memory latency dramatically whereas a stride based alignment or a random permutation increase the latency drastically. Nevertheless, a proper memory alignment reduces the number of memory transactions in the reading and writing phase to a minimum. For further information on this topic, see [6]. Keeping this in mind, iterations inside a kernel have to be seen warp-wise and not single thread-wise. In the view of a single thread, the iteration over the columns of a matrix using the column-major format should be faster than the iteration over its rows due to caching, where each column lies linearly in memory and is loaded fully or partly into the cache as a cache line. In Cuda, however, the same iteration implemented in a kernel, where the thread-ID represents the current column, leads to high memory latencies. Since memory transfers are done warp-wise and a warp is acting as a row instead of a column, the required values during an iteration step do not lie linearly in memory in the view of a warp. Thus, swapping the iteration direction inside the kernel exploits the memory architecture and decreases the memory latency dramatically, where a kernel iterates over the rows of a matrix using the column-major format and the thread-ID represents the current row so that a warp lies inside a column.

Another important point is that memory transactions can also be performed during the calculation phase in various stages to avoid overhead. Especially when input values are only used once inside a kernel, storing them separately before their usage might create avoidable overhead.

4.3.2. Kernel merging

As mentioned in Section 4.3, a kernel should be as simple as possible and perform only basic tasks to exploit where the benefits of the SIMT concept. However, strictly following this rule might create various small kernels some of which may overlap in their functionality. Therefore, some kernels are merged. This does not only increase their complexity and size but also joins equal functionalities and reduces notably overhead. On one hand, stages fused together reduce overhead for starting, finishing and synchronizing. On the other hand, if kernels e.g. have equal memory operations in common, fusing them also reduces the number of overall memory operations.

4.3.3. Pipeline split

Regarding Section 4.2.3, one major disadvantage of the pipeline design is the excessive usage of VRAM. Since kernels might require a huge amount of memory for their result, the amount of required VRAM might exceed the available capacity. In our examples, the kernel for the stress-stiffness-matrix from Eq. (A.10) already requires about 3.441 GB for linear hexahedral elements with $n_{\mathrm{el}} = 46668$ and $n_{\mathrm{pg}} = 2$, not considering the other kernels as well. For overcoming this issue, the pipeline design itself provides a proper solution. Instead of calculating $\mathbf{T}_{\mathrm{q}}^{\mathrm{glo},e}$ for each element in one pipeline run, the pipeline is divided into multiple iterations. Each iteration covers the calculation of a specific amount of elements, i.e. the amount of allocated memory has to be sufficient for a whole iteration. In general, first, the amount of overall required memory is calculated and then, depending on this, the number of iterations determined.

As a great advantage, this feature allows to use any number of elements. However, this also requires additional coordination of memory and stages, which increases the overhead:

- For predicting the number of iterations, the amount of required memory has to be calculated at the beginning when waiting for the completion signal of previous stages.
- Every kernel is started multiple times whereby each start and synchronization adds overhead.
- Some kernels have to calculate their indices during the reading phase depending on the element they are currently working on. Therefore, the current iteration number has to be passed to the kernel.

Even if those disadvantages increase the overhead, this feature is inevitable for the pipeline and the loss of performance can be covered by further improvements. As another benefit, this feature also allows to easily distribute pipeline iterations over multiple GPUs, which might be an improvement for the future.

4.3.4. Memory sharing

As mentioned in the previous section, one crucial bottleneck of the pipeline implementation is the limited VRAM size. One inevitable solution for this problem is the division of the pipeline into multiple iterations, which, however, causes overhead. The best case would be one single iteration where no further overhead is generated. Anyhow, since the number of iterations depends on the amount of required memory space, a reduction of required memory reduces the number of iterations and therefore the overhead. For achieving this, stages share their exclusive allocated memory space in order to reduce the overall memory consumption. Two stages sharing their memory, however, means that both become dependent on each other. Hence only one of them is allowed to use this memory at once so both have to be synchronized, especially across subsequent pipeline iterations. This is done by efficient synchronization tools from Cuda.

4.3.5. Multidimensional threads

Another important advantage is the possibility of sorting threads of a kernel in up to three dimensions $dim \in \{x, y, z\}$. Here, $n_{\mathrm{T}}(dim)$ denotes the number of threads per block for a specific dimension and $n_{\mathrm{B}}(dim)$ the number of blocks, and combined the overall number of threads in a dimension is calculated: $n(dim) = n_{\mathrm{T}}(dim) \cdot n_{\mathrm{B}}(dim)$. For example, in $n(x)$ single vector entries are described, in $n(y)$ the quadrature points in space and time are displayed, while $n(z)$ indicates the current element. Also the following configuration is used very often in the implementation: $n(x)$ for quadrature points in time, $n(y)$ for quadrature points in space, $n(z)$ for elements. Fig. 26 shows such a corresponding thread grid. This feature not only reduces the overhead created by the index calculations, but also simplifies the porting of the C implementation, since every dimension can be seen as an independent for-each loop. The number of threads per block, however, has a significant influence on the performance, see [7]. Therefore, the threads are distributed among the dimensions under the following constrains:

- $n_{\mathrm{T}}(x) \geq n_{\mathrm{T}}(y) \geq n_{\mathrm{T}}(z)$
- The order of dimensions should follow the memory design.
- The maximum number of threads is limited: $n_{\mathrm{T}}(x) \cdot n_{\mathrm{T}}(y) \cdot n_{\mathrm{T}}(z) \leq \max(n_{\mathrm{T}})$, where $\max(n_{\mathrm{T}})$ can be modified.
- According to [17], the number of threads does not necessarily reach the maximum to achieve the maximum occupancy.

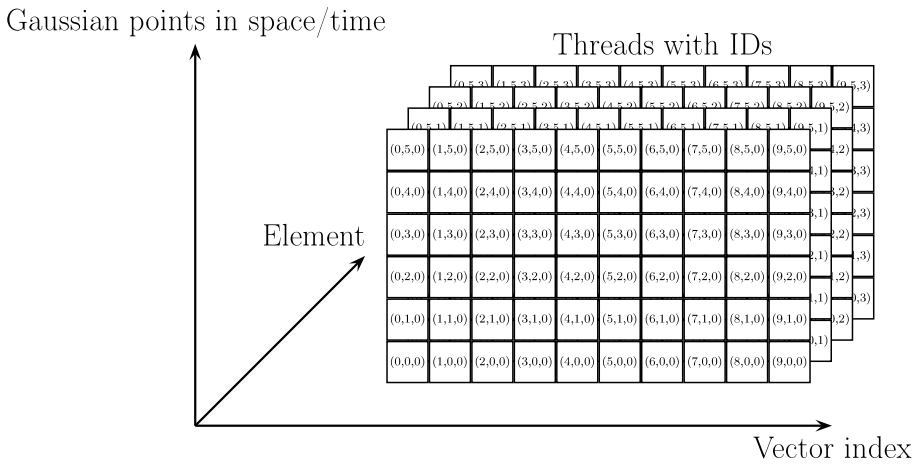

Fig. 26. Multidimensional thread grid.

4.3.6. Overlapping kernels

Besides this improvement, the concept of Cuda also allows two different kernels to run concurrently as shown in Fig. 27. This can be done when the kernels do not depend on each other. However, since the GPU is limited in its capabilities, this feature can only be used when two small, i.e. less complex, kernels are performing or when both of them cannot utilize the graphic card sufficiently enough.

Improvements are possible when memory transfers are done between the graphic card and the main memory, since the GPU can perform them in parallel on kernels. In other words, while a kernel performs operations on an array of data, another finished part of the data array can be transferred without any loss of time.

As an example, Fig. 27 provides an extract of a schematic time line where various kernels and memory transfer operations are displayed. A red rectangle here stands for a memory transfer where "DTH" indicates a transfer from the GPU to the main memory and "HTD" from the main memory to the GPU. The other rectangles symbolize single Cuda kernels which are used in the implementation:

- **q** selects the position.
- **F** calculates the deformation gradient.
- **C** calculates the right Cauchy–Green.
- **S** and **CC** apply the stress, see Appendix C.2.
- **B** calculates the "B-matrix" from Eq. (A.8), "SST" the stress-stiffness-matrix from Eq. (A.10).
- **Tang** calculates $\mathbf{T}_{\mathrm{q}}^{\mathrm{glo},e}$, so afterwards the tangent can be transferred to the main memory.

In fact, the actual implementation includes additional kernels which are not displayed here since they are only used for optimizations. The position of the kernels, represented as rectangles in the time line, indicates the start and end time of the kernel or the transfer so the width displays the duration. With this in mind, kernel overlapping occurs when at least two kernels or transfer operations are below each other, e.g. "HTD", "C" and "B" as shown in the time line. This means, the kernels "C" and "B" run in parallel while a memory transfer from the main memory to the GPU is performed.

Regarding the pipeline split from Section 4.3.3, the overlapping also allows to start and run kernels and transfers from the following pipeline iteration in the current one. In the time line, blue and yellow colored rectangles represent kernels from the current iteration while green colored ones are kernels from the following iteration.

In general, if implemented correctly regarding the dependencies, kernel overlapping may reduce the run time significantly since the GPU can be utilized better and memory transfers can be done in parallel to calculations.

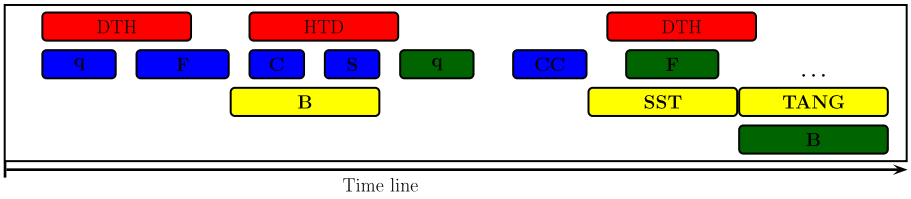

Fig. 27. Simplified time line for the Cuda streams. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4.4. Comparison of CPU and GPU implementation

This section focuses on the comparison of the performance of the GPU implementation and the CPU implementation. At the beginning, the test systems and the test cases are introduced. Next, the results of the test cases are shown and finally their outcomes and their consequences are discussed in the final subsection.

4.4.1. Testing system and test cases

For the comparison of the CPU and the GPU implementation, the same testing scenario and the same parameters were used as in Section 3. However, the tests were limited to only 20 time steps, since this already provides sufficient timing data and further time steps would only scale the results. For the comparison, we used workstations with an Intel XEON E5-1620 CPU with four physical cores at 3.6 GHz and 64 GB RAM. The C implementation was compiled by the GCC compiler with **-Ofast** as optimization option and OpenMP for parallelizing so the calculation of $\mathbf{T}_{\mathfrak{q}}^{\mathrm{glo},e}$ was distributed over all four physical cores. In addition, the Mkl BLAS library was used for the linear algebra operations. For the GPU implementation, the NVCC compiler from NVIDIA was used with **-O3 –restrict -maxrregcount 64** as optimization options. Further, the compiler option **-fmad=false** was used. Note that the Fused Multiply–Add feature of the GPU was turned off in order to gain almost identical result compared to the CPU.

For the comparison, a scaling factor called "Cuda factor" c_{F} is introduced. This factor describes the distribution of elements n_{el} for calculating $\mathbf{T}_{\mathfrak{q}}^{\mathrm{glo},e}$ between the CPU and GPU. As shown in Fig. 22, the GPU accelerator can be combined with the CPU to gain the maximum speedup so both hardware resources are fully utilized. Hence, the number of elements n_{el} being calculated on the GPU can be set by the Cuda factor $c_{\mathrm{F}} \in [0, 1]$, where $c_{\mathrm{F}} = 0$ means CPU only, $c_{\mathrm{F}} = 0.5$ first half on the GPU and the second one on the CPU and $c_{\mathrm{F}} = 1$ GPU only.

Depending on the factor c_{F}, the speedup S_{c} is calculated by the division of the CPU only time $t_{\mathrm{c}=0}$, i.e. the time the CPU requires for its portion of calculations, by the GPU time t_{c}: $S_{\mathrm{c}} = t_0/t_{\mathrm{c}}$. Here, the speedup S_{c} describes how many times the program is faster for $c_{\mathrm{F}} > 0$, compared to if the CPU only runs for $c_{\mathrm{F}} = 0$.

In addition, an optimal Cuda factor c_{op} can be calculated with the help of S_0 and S_1 regardless of any overhead for the coordination of the CPU and GPU. If, for example, $S_1 = 3$, then the GPU is 3 times faster than the CPU implementation. This means, calculating 75% of the elements on the GPU is as fast as calculating only 25% with the CPU, so $c_{\mathrm{F}} = 0.75$. In general, c_{op} is calculated as following:

$$c_{\mathrm{op}} = 1 - \frac{1}{1 + S_1}, 0 < c_{\mathrm{op}} < 1 \tag{99}$$

Including the effects of the overhead however, the actually reached optimum c'_{op} has to be higher than the theoretical one $c'_{\mathrm{op}} > c_{\mathrm{op}}$. The main reason for this is that both the CPU and GPU cannot start and finish at the same time since the CPU first has to start each kernel inside the pipeline and only then is able to calculate its portion of elements n_{el}. In the meantime, after the start of the first Cuda kernel, the GPU already gains an advantage over the CPU. Therefore, the number of elements n_{el} calculated on the GPU should be higher in comparison to the theoretical optimum which leads to a higher Cuda factor $c'_{\mathrm{op}} > c_{\mathrm{op}}$. Likewise, the number of elements for the CPU has to be lower since the CPU has to synchronize with the GPU at the end of its calculations. The synchronization, however, again induces overhead on the CPU. If both the CPU and the GPU finish at the same time, the highest utilization of both hardware resources and therefore the lowest run time is achieved since none of them is stalled. This, in fact, again requires a higher Cuda factor $c'_{\mathrm{op}} > c_{\mathrm{op}}$. In summary, the testing results should display that $c'_{\mathrm{op}} > c_{\mathrm{op}}$. If,

however, the pure calculation time is much higher than the overhead on both hardware resources, then the overhead is almost covered and the reached optimum has to return to the theoretical one so $c'_{\mathrm{op}} \approx c_{\mathrm{op}}$.

For testing, different values for c_{F} where used: $c_{\mathrm{F}} = \{0, 0.4, 0.5, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 9, 0.95, 1.0\}$.

4.4.2. Results

The testing results are shown in Figs. 28–37. They provide diagrams where the abscissa indicates the Cuda factor $c_{\mathrm{F}} \in [0, 1]$ while the ordinate displays the reached Speedup S_{c}. In addition, the speedup at the optimal Cuda factor S_{op} is also marked. For each n_{dof} in a testing scenario, the program is tested with various values for c_{F} and displayed in separate graphs.

The first testing scenario which is discussed is the quadratic hexahedral. For $n_{\mathrm{pg}} = 1$ in Fig. 30, the achieved speedup for Cuda is $S_1 \approx 2.70$, and the optimum is $c_{\mathrm{op}} \approx 0.73$. The achieved optimum c'_{op} in fact is slightly greater than c_{op}. For a constant Cuda factor c_{F} the reached speedup S_{c} remains stable for each n_{dof}. Thus, for an increasing n_{dof} the GPU implementation may provide a constant speedup and losses due to overhead, e.g. increased amount of required memory, pipeline splits, etc., might be neglected. From $c_{\mathrm{F}} = 0$ to $c_{\mathrm{F}} = c'_{\mathrm{op}}$, the values for S_{c} rise considerably from $S_{\mathrm{c}} = 1$ to the peak of $S_{\mathrm{op}} \approx 3.6$. Then, S_{c} moderately declines to $S_1 \approx 2.70$. In Section 4.4.1, this development was expected and proves the theoretical predictions.

Next, the quadratic hexahedral with $n_{\mathrm{pg}} = 2$ is analyzed, as shown in Fig. 31. In comparison to $n_{\mathrm{pg}} = 1$, the maximum speedup for Cuda is only $S_1 \approx 2.20$ and therefore $c_{\mathrm{op}} = 0.68$, while the reached optimum is again

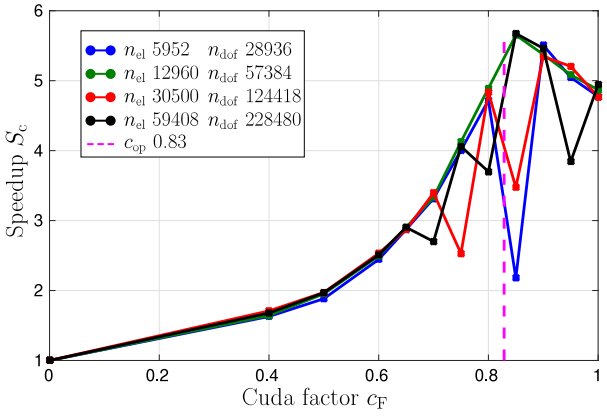

Fig. 28. Speedup for the linear hexahedral elements with $n_{\mathrm{pg}} = 1$.

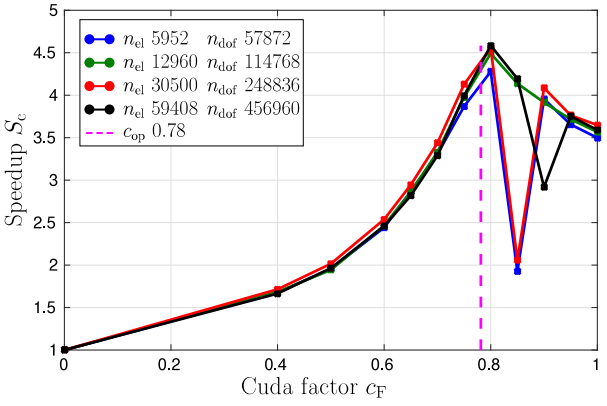

Fig. 29. Speedup for the linear hexahedral elements with $n_{\mathrm{pg}} = 2$.

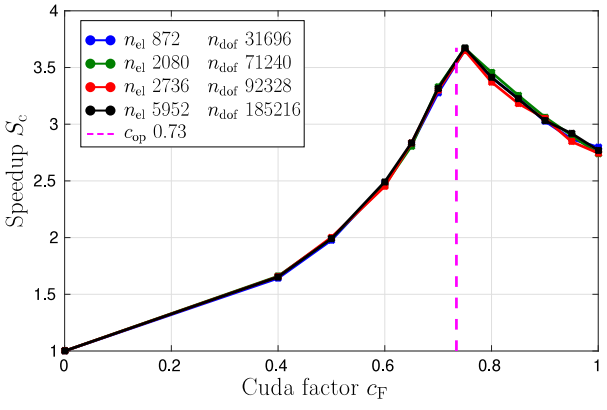

Fig. 30. Speedup for the quadratic hexahedral elements with $n_{\mathrm{pg}} = 1$.

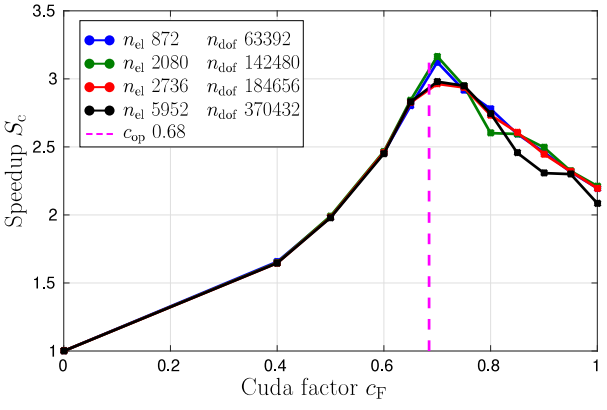

Fig. 31. Speedup for the quadratic hexahedral elements with $n_{\mathrm{pg}} = 2$.

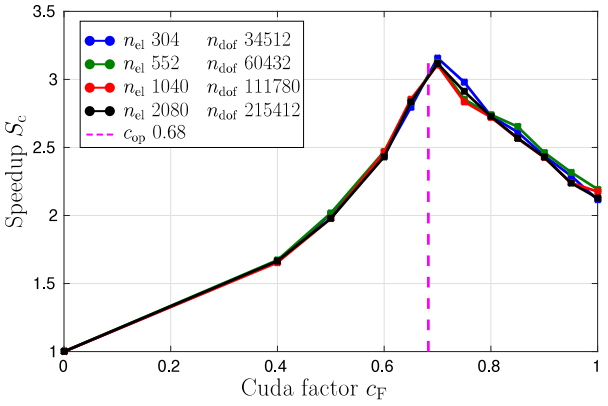

Fig. 32. Speedup for the cubic hexahedral elements with $n_{pg} = 1$.

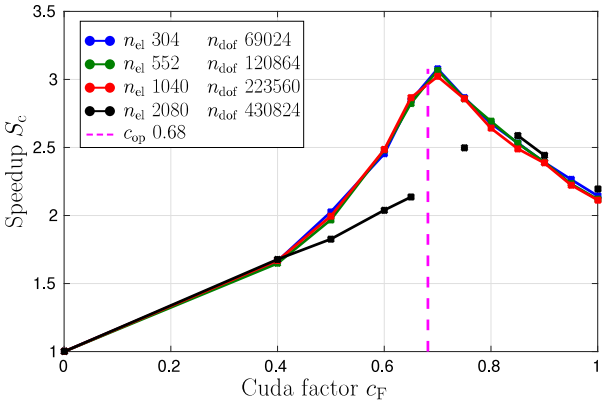

Fig. 33. Speedup for the cubic hexahedral elements with $n_{pg} = 2$.

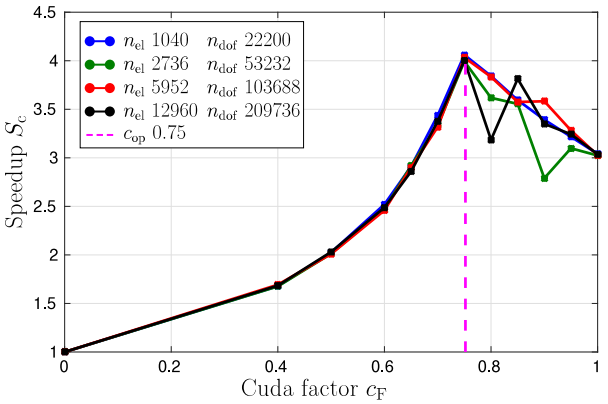

Fig. 34. Speedup for the quadratic serendipity hexahedral elements with $n_{pg} = 1$.

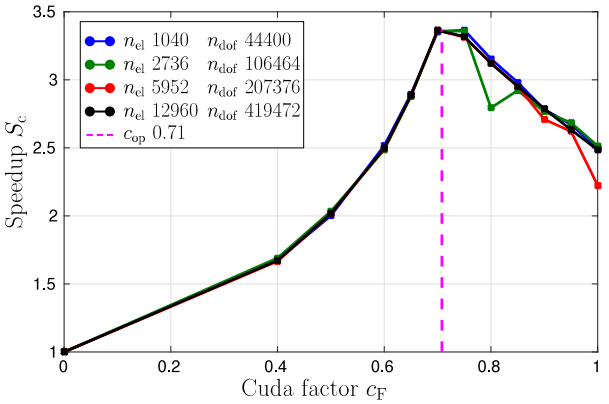

Fig. 35. Speedup for the quadratic serendipity hexahedral elements with $n_{pg} = 2$.

slightly higher: $c'_{op} > c_{op}$. The developments of the graphs are quite similar to $n_{pg} = 1$, except that the n_{dof} are doubled. For $n_{dof} = 184656$ and $n_{dof} = 370432$, however, the peak for $S_{op} \approx 3.0$ is lower than for the other two ones. Also, their plots fluctuate slightly in comparison to $n_{pg} = 1$. Reasons for this behavior are unfavorable thread distributions for Cuda kernels that may occur as well as negative effects due to the pipeline design. The thread configuration of the pipeline kernels mostly depends on the number of quadrature points in space and time, as mentioned in Section 4.3.5. In order to avoid unnecessary kernel starts and induced overhead, the configuration can be chosen in a way that insufficient threads are started for the maximum utilization of the GPU, see Section 4.3.5. Additionally, as a result of the pipeline split the last pipeline iteration may only cover a few elements, e.g. during the first iteration over 1000 elements are calculated while the second one only covers up to 10. The overhead for calculating these few elements, however, is greater compared to a higher amount of elements where the calculation time is high enough to cover the overhead. These fluctuations also can be clearly seen in the other testing scenarios. In general, all testing scenarios follow the same development as the quadratic hexahedral elements. Table 6 provides an overview for the maximum speedup of each element type.

Some scenarios, however, vary in their development, e.g. in Figs. 28 and 29. For the linear hexahedral elements with $n_{pg} = 1$ the speedup at $c_F = 0.85$ varies from approximately 2.1 for $n_{dof} = 28936$ up to 5.67 for $n_{dof} = 228480$. Even between two consecutive Cuda factors there is a rise in the speedup. For example, for $n_{dof} = 28936$ the speedup between $c_F = 0.85$ and $c_F = 0.9$ differs about a value of 3.4. Here, no pipeline split is used since the number of elements is small enough to be handled in a single pipeline iteration and therefore this is not the cause of this

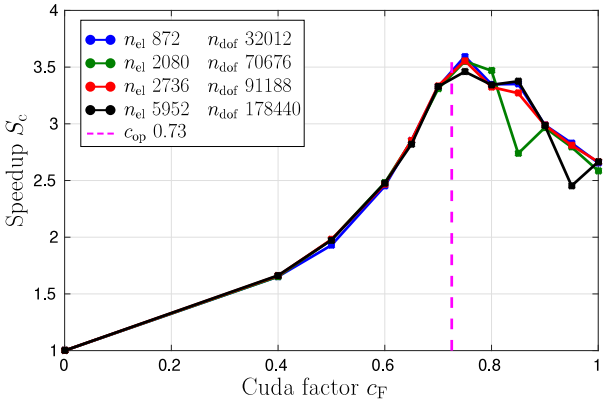

Fig. 36. Speedup for the cubic serendipity hexahedral elements with $n_{pg} = 1$.

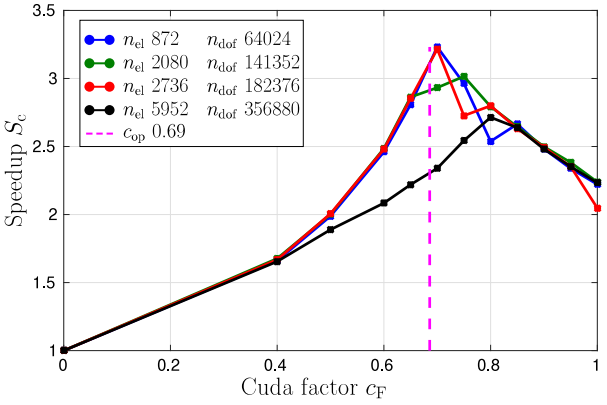

Fig. 37. Speedup for the cubic serendipity hexahedral elements with $n_{pg} = 2$.

behavior. Inside each pipeline iteration however, even some single stages might be split into multiple iterations to overlap memory transfer operations as mentioned in Section 4.3.6. For example, if results from a stage are required in the RAM for the CPU, the GPU first has to finish the stage and only then can start the memory transfer from the VRAM to the RAM. This can be accelerated by dividing the stage e.g. into two parts and after the first one finishes, the memory transfer for the first half is started while the second half is being calculated in the meantime. Dividing a stage into three, four, etc. parts is possible as well and is done by the pipeline in the program. However, similar to the thread configuration, an unfavorable choice for this division can be made which might decrease the achieved speedup. Nevertheless, despite the significant differences that may occur, the impact on the overall program time is neglectable. In fact, the whole scenario for $n_{dof} = 28936$ with $c_F = 0.9$ was only about 4 s slower compared to $c_F = 0.85$ and with higher GPU utilization due to a higher number of elements and quadratic points in space and time the thread configuration and the pipeline split might have an impact on the achieved speedup.

Other significant deviations occur for the cubic hexahedral and cubic serendipity hexahedral elements with $n_{pg} = 2$ in Figs. 31 and 35. While the plot for the cubic hexahedral elements with $n_{dof} = 430824$ is uncompleted for some values of c_F, the speedup for the cubic serendipity hexahedral elements is notably lower in case of $n_{dof} = 356880$ for the majority of values of c_F. For the cubic hexahedral elements, the reason for the missing values can be found in the memory management, see Section 4.3.3. On one hand, the required memory space is calculated by the size of the required data arrays. On the other hand, memory space is occupied by Cuda itself, e.g. for spilled registers. Hence the amount of the available memory space, including the memory usage of Cuda kernels, might be lower than the physically available space. However, the request and the calculation of the occupied memory by Cuda is not possible. Therefore, in some uncommon cases of a high demand of memory space, the amount of the required space might exceed the amount of available space, even after the pipeline split which leads to the termination of the program. With further improvements this problem might be solved in the future. On the contrast, the cubic serendipity hexahedral elements with $n_{dof} = 356880$ suffer from an adverse pipeline split where a few elements remain for the last iteration, as mentioned before.

4.5. Conclusions to the GPU-implementation

The main results are speedups between 3.1 and 5.7 for an efficient implementation of the tangent $\mathbf{T}_q^{glo,e}$ in Cuda. The exact speedup depends heavily on the choice of the Cuda factor. A general statement regarding the speedup can be made to the number of quadrature points for every element type, compare Table 6. The linear hexahedral element with eight nodes and a linear approximation in time achieves the highest speedup followed by the quadratic approximation in time for this element type. The larger the number of quadrature points in space and time and the number of nodes are, the smaller is the speedup and Cuda factor.

Table 6
Maximal speedup for different element types.

Element type	H8	H20	H27	H32	H64	H8	H20	H27	H32	H64
n_{pg}	1	1	1	1	1	2	2	2	2	2
n_{no} with n_{pg}	8	20	27	32	64	16	40	54	64	128
quadrature points	8	27	27	64	64	8	27	27	64	64
max speedup	5.67	4.06	3.67	3.59	3.16	4.58	3.37	3.17	3.23	3.08
Cuda factor c_F	0.83	0.75	0.73	0.73	0.68	0.78	0.71	0.68	0.69	0.68

In this section several possibilities have been described for an implementation of a finite element routine in Cuda. In general, a pipeline design was introduced for an efficient GPU implementation, where the pipeline consists of multiple stages which are handled by the implementation. Also various optimizations like pipeline splitting were presented, which adapt the implementation to each testing scenario and allow to exceed the memory limitation. On the other hand these methods lead to variations in the reached speedup so choosing the ideal settings based on a short pre-simulation test is recommended. Nevertheless, the Cuda factor c_F provides a useful constraint for reaching the maximum speedup and it can be stated that for hyperelastic materials without an evolution equation the speedup and the Cuda factor may shift further.

A higher speedup can also be achieved by the usage of a multi-GPU system. Our testing systems were limited to a single GPU and therefore the implementation is only capable of handling one GPU. However, due to the split pipeline design the implementation only has to be adapted in a way that pipeline iterations are distributed among multiple GPUs. Especially for a rising number of quadrature points in space and time, where the calculation time is significantly higher than the induced overhead, further tests might state out whether an almost linear growth of the speedup depending on the GPU count could be possible or is restricted by unknown criterions.

5. Outlook

The aim of this paper is to present a new time stepping scheme for a viscoelastic material formulation which is based on variational integrators combining the preservation of the total linear momentum, the total angular momentum, and the balance of total energy. This concept should be used for further works in the field of thermo-viscoelastic materials. The introduction of the temperature in the variational principal is also possible as for position and linear momentum.

As already mentioned we recommend to use a combination of different NEWTON criteria in the NEWTON iteration used to guarantee the fulfillment of all physically relevant balances exactly under the NEWTON tolerance.

The third continuation of this work could be the application of multiple GPUs to investigate the influence of more Cuda threads and memory on the speedup. Just as interesting is the application of other elements for the discretization in space. As a conclusion of Section 4 the linear hexahedral elements have the smallest number of nodes and the smallest number of quadrature points but the highest speedup can be achieved by this element type. It is known that tetrahedral elements have a smaller number of nodes and a less number of quadrature points compared to hexahedral elements. It should be tested if tetrahedral elements reach a higher speedup compared to hexahedral elements with the same number of degrees of freedom.

An other interesting point could be the speedup factor with the GPU implementation for hyperelastic materials, because the calculation effort is much smaller compared to a viscoelastic material for every element.

Acknowledgments

The authors thank the ‘Deutsche Forschungsgesellschaft (DFG)’ for financial support of this work under the grants GR 3297/2-1, GR 3297/2-2 and GR 3297/4-1.

Appendix A. Operator definitions

For simplicity of notation, we introduce some operators to describe the mathematical operations. The construction of the operators goes back to the work of [2]. We introduced these special operators to reduce the mathematical

overhead and the computational time. $\mathbf{A}$ and $\mathbf{D}$ in $\mathbb{R}^{3\times3}$ are the input for the operator $\overset{\text{AD}}{\mathcal{P}}_{\text{M}}\{\bullet\ ,\ \bullet\}$. The result is $\mathbb{R}^{3\times3}$ again.

$$\overset{\text{AD}}{\mathcal{P}}_{\text{M}}\{\mathbf{A}\ ,\ \mathbf{D}\} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \cdot \begin{bmatrix} d_{1,1} & d_{1,2} & d_{1,3} \\ d_{2,1} & d_{2,2} & d_{2,3} \\ d_{3,1} & d_{3,2} & d_{3,3} \end{bmatrix} \tag{A.1}$$

or in the symmetric case

$$\overset{\text{AD}}{\mathcal{P}}_{\text{M}}\{\mathbf{A}\ ,\ \mathbf{D}\} = \begin{bmatrix} a_1 & a_4 & a_6 \\ a_4 & a_2 & a_5 \\ a_6 & a_5 & a_3 \end{bmatrix} \cdot \begin{bmatrix} d_1 & d_4 & d_6 \\ d_4 & d_2 & d_5 \\ d_6 & d_5 & d_3 \end{bmatrix} \tag{A.2}$$

The JACOBI matrix $\mathbf{J}^e$ on the reference element is defined by:

$$\mathbf{J}^e = \sum_{A=1}^{n_{\text{no}}} \mathbf{X}^{eA} \otimes \text{Grad}\left[\text{N}^A(\boldsymbol{\xi})\right]. \tag{A.3}$$

$\mathbf{X}^{eA} \in \mathcal{B}_0$ denotes the x, y, z-values of the node A of element e. The shape function in space N^A is defined for node A with the local reference coordinates $\boldsymbol{\xi} = [\xi, \eta, \zeta]^{\text{T}} \in [-1, 1] \times [-1, 1] \times [-1, 1]$. The gradient of the shape functions $\tilde{\boldsymbol{\nabla}}^{eA}$ is given by:

$$\tilde{\boldsymbol{\nabla}}^{eA} = \left(\mathbf{J}^e\right)^{-\text{T}} \text{Grad}\left[\text{N}^A(\boldsymbol{\xi})\right] = \left[\widetilde{\nabla}_x^{eA}, \widetilde{\nabla}_y^{eA}, \widetilde{\nabla}_z^{eA}\right]^{\text{T}} \tag{A.4}$$

$$\tilde{\boldsymbol{\nabla}}_x^e = \left[\widetilde{\nabla}_x^{e\,1} \quad \widetilde{\nabla}_x^{e\,2} \quad \dots \quad \widetilde{\nabla}_x^{e\,n_{\text{no}}}\right] \tag{A.5}$$

$$\tilde{\boldsymbol{\nabla}}_y^e = \left[\widetilde{\nabla}_y^{e\,1} \quad \widetilde{\nabla}_y^{e\,2} \quad \dots \quad \widetilde{\nabla}_y^{e\,n_{\text{no}}}\right] \tag{A.6}$$

$$\tilde{\boldsymbol{\nabla}}_z^e = \left[\widetilde{\nabla}_z^{e\,1} \quad \widetilde{\nabla}_z^{e\,2} \quad \dots \quad \widetilde{\nabla}_z^{e\,n_{\text{no}}}\right] \tag{A.7}$$

The operator $\overset{\text{B}}{\mathcal{P}}_{\text{M}}\{\bullet\}$ can be used as a constructor of the well known "B-matrix".

$$\overset{\text{B}}{\mathcal{P}}_{\text{M}}\{\mathbf{A}\} = \begin{bmatrix} \tilde{\boldsymbol{\nabla}}_x^e \boxtimes (\mathbf{A}[:,1])^{\text{T}} \\ \tilde{\boldsymbol{\nabla}}_y^e \boxtimes (\mathbf{A}[:,2])^{\text{T}} \\ \tilde{\boldsymbol{\nabla}}_z^e \boxtimes (\mathbf{A}[:,3])^{\text{T}} \\ \tilde{\boldsymbol{\nabla}}_x^e \boxtimes (\mathbf{A}[:,2])^{\text{T}} + \tilde{\boldsymbol{\nabla}}_y^e \boxtimes (\mathbf{A}[:,1])^{\text{T}} \\ \tilde{\boldsymbol{\nabla}}_y^e \boxtimes (\mathbf{A}[:,3])^{\text{T}} + \tilde{\boldsymbol{\nabla}}_z^e \boxtimes (\mathbf{A}[:,2])^{\text{T}} \\ \tilde{\boldsymbol{\nabla}}_z^e \boxtimes (\mathbf{A}[:,1])^{\text{T}} + \tilde{\boldsymbol{\nabla}}_x^e \boxtimes (\mathbf{A}[:,3])^{\text{T}} \end{bmatrix} \tag{A.8}$$

The combination of $\overset{\text{AD}}{\mathcal{P}}_{\text{M}}\{\mathbf{A}\ ,\ \mathbf{D}\}$ and $\overset{\text{B}}{\mathcal{P}}_{\text{M}}\{\bullet\}$ is to describe the double dot product of a symmetric tensor with the "B-matrix". This is more efficient than first create a B-matrix and build the double dot product.

$$\begin{aligned} \overset{\text{BS}}{\mathcal{P}}_{\text{M}}\{\mathbf{A}\ ,\ \mathbf{D}\} = \left(\overset{\text{B}}{\mathcal{P}}_{\text{M}}\{\mathbf{A}\}\right)^{\text{T}} \cdot \mathbf{D} &= \left(\tilde{\boldsymbol{\nabla}}_1^e\right)^{\text{T}} \boxtimes \left(\overset{\text{AD}}{\mathcal{P}}_{\text{M}}\{\mathbf{A}\ ,\ \mathbf{D}\}[:,1]\right) \\ &+ \left(\tilde{\boldsymbol{\nabla}}_2^e\right)^{\text{T}} \boxtimes \left(\overset{\text{AD}}{\mathcal{P}}_{\text{M}}\{\mathbf{A}\ ,\ \mathbf{D}\}[:,2]\right) \\ &+ \left(\tilde{\boldsymbol{\nabla}}_3^e\right)^{\text{T}} \boxtimes \left(\overset{\text{AD}}{\mathcal{P}}_{\text{M}}\{\mathbf{A}\ ,\ \mathbf{D}\}[:,3]\right) \end{aligned} \tag{A.9}$$

The last operator is defined as follows

$$\overset{\text{SST}}{\mathcal{P}}_{\text{M}}\{\mathbf{D}\} = \begin{bmatrix} \left[\left(\tilde{\boldsymbol{\nabla}}_x^e\right)^{\text{T}} \cdot \tilde{\boldsymbol{\nabla}}_x^e\right] d_1 \\ + \left[\left(\tilde{\boldsymbol{\nabla}}_y^e\right)^{\text{T}} \cdot \tilde{\boldsymbol{\nabla}}_y^e\right] d_2 \\ + \left[\left(\tilde{\boldsymbol{\nabla}}_z^e\right)^{\text{T}} \cdot \tilde{\boldsymbol{\nabla}}_z^e\right] d_3 \\ + \left[\left(\tilde{\boldsymbol{\nabla}}_x^e\right)^{\text{T}} \cdot \tilde{\boldsymbol{\nabla}}_y^e + \left(\tilde{\boldsymbol{\nabla}}_y^e\right)^{\text{T}} \cdot \tilde{\boldsymbol{\nabla}}_x^e\right] d_4 \\ + \left[\left(\tilde{\boldsymbol{\nabla}}_y^e\right)^{\text{T}} \cdot \tilde{\boldsymbol{\nabla}}_z^e + \left(\tilde{\boldsymbol{\nabla}}_z^e\right)^{\text{T}} \cdot \tilde{\boldsymbol{\nabla}}_y^e\right] d_5 \\ + \left[\left(\tilde{\boldsymbol{\nabla}}_z^e\right)^{\text{T}} \cdot \tilde{\boldsymbol{\nabla}}_x^e + \left(\tilde{\boldsymbol{\nabla}}_x^e\right)^{\text{T}} \cdot \tilde{\boldsymbol{\nabla}}_z^e\right] d_6 \end{bmatrix}^{\text{sym } n_{\text{no}} \times n_{\text{no}}} \boxtimes \mathbf{I}^{3\times 3} \tag{A.10}$$

to create the well known "stress-stiffness-matrix".

The kron symbols in the tangent are defined by:

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} a_0b_0 & a_3b_3 & a_5b_5 & \frac{1}{2}(a_0b_3 + a_3b_0) & \frac{1}{2}(a_3b_5 + a_5b_3) & \frac{1}{2}(a_0b_5 + a_5b_0) \\ a_3b_3 & a_1b_1 & a_4b_4 & \frac{1}{2}(a_1b_3 + a_3b_1) & \frac{1}{2}(a_1b_4 + a_4b_1) & \frac{1}{2}(a_3b_4 + a_4b_3) \\ a_5b_5 & a_4b_4 & a_2b_2 & \frac{1}{2}(a_4b_5 + a_5b_4) & \frac{1}{2}(a_2b_4 + a_4b_2) & \frac{1}{2}(a_2b_5 + a_5b_2) \\ a_0b_3 & a_3b_1 & a_5b_4 & \frac{1}{2}(a_0b_1 + a_3b_3) & \frac{1}{2}(a_5b_1 + a_3b_4) & \frac{1}{2}(a_0b_4 + a_5b_3) \\ a_3b_5 & a_1b_4 & a_4b_2 & \frac{1}{2}(a_1b_5 + a_3b_4) & \frac{1}{2}(a_1b_2 + a_4b_4) & \frac{1}{2}(a_3b_2 + a_4b_5) \\ a_0b_5 & a_3b_4 & a_5b_2 & \frac{1}{2}(a_0b_4 + a_3b_5) & \frac{1}{2}(a_3b_2 + a_5b_4) & \frac{1}{2}(a_0b_2 + a_5b_5) \end{bmatrix} \tag{A.11}$$

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} b_0 & a_0b_1 & a_0b_2 & a_0b_3 & a_0b_4 & a_0b_5 \\ b_0 & a_1b_1 & a_1b_2 & a_1b_3 & a_1b_4 & a_1b_5 \\ b_0 & a_2b_1 & a_2b_2 & a_2b_3 & a_2b_4 & a_2b_5 \\ b_0 & a_3b_1 & a_3b_2 & a_3b_3 & a_3b_4 & a_3b_5 \\ b_0 & a_4b_1 & a_4b_2 & a_4b_3 & a_4b_4 & a_4b_5 \\ b_0 & a_5b_1 & a_5b_2 & a_5b_3 & a_5b_4 & a_5b_5 \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix}^{\text{T}} . \tag{A.12}$$

$\mathbf{V} = [1, 1, 1, 2, 2, 2]^{\text{T}}$ is used as correction term in the VOIGT notation. A more detailed description including the advantages can be found in [2].

Appendix B. Global residual

Based on the global residual that depends on the position vector $\mathbf{q}$ and the internal variable $\mathbf{C}_{\text{i}}$ there are two possibilities to solve this system. The first one is to define a vector of unknown variables that depends on the position vector and the internal variables. This leads to a very large system of algebraic nonlinear equations. The second method is to calculate locally internal variables for every element in the global residual. In contrast to the first method this leads to a system with a lot of small systems of equations that have to be solved. The small systems are solved by local NEWTON methods for the internal variables. Here, the size of the global system only depends on the LAGRANGE multiplier and the unknown positions. Compared to an elastic model the dimension of the solution is equal. This is called a multi-level NEWTON–RAPHSON method. The unknown variables are defined as:

$$\mathbf{C}_{\text{i}\iota} = \begin{bmatrix} \mathbf{C}_{\text{i}\frac{1}{n_{\text{pg}}}} & \cdots & \mathbf{C}_{\text{i}\frac{n_{\text{pg}}}{n_{\text{pg}}}} \end{bmatrix}^T \qquad \mathbf{q}_\iota = \begin{bmatrix} \mathbf{q}_{\frac{1}{n_{\text{pg}}}} & \cdots & \mathbf{q}_{\frac{n_{\text{pg}}}{n_{\text{pg}}}} \end{bmatrix}^T . \tag{B.1}$$

$\mathbf{q}_0$ and $\mathbf{C}_{\text{i}0}$ are set as given. The global NEWTON–RAPHSON method follows by:

$$\mathbf{R}^{\text{glo}}(\mathbf{C}_{\text{i}\iota} + \Delta\mathbf{C}_{\text{i}}, \mathbf{q}_\iota + \Delta\mathbf{q}) = \mathbf{R}^{\text{glo}}(\mathbf{C}_{\text{i}\iota}, \mathbf{q}_\iota) + \frac{\partial \mathbf{R}^{\text{glo}}(\mathbf{C}_{\text{i}\iota}, \mathbf{q}_\iota)}{\partial \mathbf{C}_{\text{i}\iota}} \Delta\mathbf{C}_{\text{i}} + \frac{\partial \mathbf{R}^{\text{glo}}(\mathbf{C}_{\text{i}\iota}, \mathbf{q}_\iota)}{\partial \mathbf{q}_\iota} \Delta\mathbf{q} = \mathbf{0} \tag{B.2}$$

$$\mathbf{C}_{\text{i}\iota}^{m+1} = \mathbf{C}_{\text{i}\iota}^{m} - \left(\mathbf{T}_{\text{Ci}}^{\text{loc}}(\mathbf{C}_{\text{i}\iota}^{m}, \mathbf{q}_\iota)\right)^{-1} \mathbf{R}^{\text{loc}}(\mathbf{C}_{\text{i}\iota}^{m}, \mathbf{q}_\iota) \tag{B.3}$$

$$\Delta \mathbf{C}_\mathrm{i} = -\left(\mathbf{T}^{\mathrm{loc}}_{\mathrm{Ci}}(\mathbf{C}_{\mathrm{i}\iota}, \mathbf{q}_\iota)\right)^{-1} \mathbf{T}^{\mathrm{loc}}_{\mathrm{q}}(\mathbf{C}_{\mathrm{i}\iota}, \mathbf{q}_\iota)\Delta\mathbf{q} \tag{B.4}$$

$$-\mathbf{R}^{\mathrm{glo}}(\mathbf{C}_{\mathrm{i}\iota}, \mathbf{q}_\iota) = \mathbf{T}^{\mathrm{glo}}_{\mathrm{Ci}}(\mathbf{C}_{\mathrm{i}\iota}, \mathbf{q}_\iota)\Delta\mathbf{C}_\mathrm{i} + \mathbf{T}^{\mathrm{glo}}_{\mathrm{q}}(\mathbf{C}_{\mathrm{i}\iota}, \mathbf{q}_\iota)\Delta\mathbf{q} \tag{B.5}$$

$$-\mathbf{R}^{\mathrm{glo}}(\mathbf{C}_{\mathrm{i}\iota}, \mathbf{q}_\iota) = \left[\mathbf{T}^{\mathrm{glo}}_{\mathrm{q}}(\mathbf{C}_{\mathrm{i}\iota}, \mathbf{q}_\iota) - \mathbf{T}^{\mathrm{glo}}_{\mathrm{Ci}}(\mathbf{C}_{\mathrm{i}\iota}, \mathbf{q}_\iota)\left(\mathbf{T}^{\mathrm{loc}}_{\mathrm{Ci}}(\mathbf{C}_{\mathrm{i}\iota}, \mathbf{q}_\iota)\right)^{-1} \mathbf{T}^{\mathrm{loc}}_{\mathrm{q}}(\mathbf{C}_{\mathrm{i}\iota}, \mathbf{q}_\iota)\right]\Delta\mathbf{q} \tag{B.6}$$

$$-\mathbf{R}^{\mathrm{glo}}(\mathbf{C}_{\mathrm{i}\iota}, \mathbf{q}_\iota) = \mathbf{T}^{\mathrm{glo}}(\mathbf{C}_{\mathrm{i}\iota}, \mathbf{q}_\iota)\Delta\mathbf{q} \tag{B.7}$$

$$\mathbf{q}^{j+1}_\iota = \mathbf{q}^{j}_\iota - \left(\mathbf{T}^{\mathrm{glo}}(\mathbf{q}^{j}_\iota, \mathbf{C}_{\mathrm{i}\iota})\right)^{-1} \mathbf{R}^{\mathrm{glo}}(\mathbf{q}^{j}_\iota, \mathbf{C}_{\mathrm{i}\iota}). \tag{B.8}$$

The staggered NEWTON–RAPHSON method can be implemented as listed in Algorithm 1.

Algorithm 1: Simplified time stepping scheme

forall the *time steps k* **do**
- **set:** $j = 0$ and $\bar{\mathbf{q}}^j = \mathbf{q}^k$;
- calc global residuum with update of $\mathbf{C}_\mathrm{i}{}^k$ in local residuum;
- **for** *calc* $\mathbf{R}^{glo}(\bar{\mathbf{q}}^j, \mathbf{C}_\mathrm{i}{}^k)$ **do**
  - **set:** $m = 0$ and $\bar{\mathbf{C}}^m_\mathrm{i} = \mathbf{C}_\mathrm{i}{}^k$;
  - **while** $\|\mathbf{R}^{loc}(\bar{\mathbf{q}}^j, \bar{\mathbf{C}}^m_\mathrm{i})\| > TOL$ **do**
    - $\bar{\mathbf{C}}^{m+1}_\mathrm{i} = \bar{\mathbf{C}}^m_\mathrm{i} - \left(\mathbf{T}^{loc}_{Ci}(\bar{\mathbf{q}}^j, \bar{\mathbf{C}}^m_\mathrm{i})\right)^{-1} \mathbf{R}^{loc}(\bar{\mathbf{q}}^j, \bar{\mathbf{C}}^m_\mathrm{i})$;
    - $m++$;
  - $\mathbf{C}_\mathrm{i}{}^k = \bar{\mathbf{C}}^m_\mathrm{i}$;
- **while** $\|\mathbf{R}^{glo}(\bar{\mathbf{q}}^j, \mathbf{C}_\mathrm{i}{}^k)\| > TOL$ **do**
  - calc $\mathbf{T}^{glo}(\bar{\mathbf{q}}^j, \mathbf{C}_\mathrm{i}{}^k)$;
  - $\bar{\mathbf{q}}^{j+1} = \bar{\mathbf{q}}^j - \left(\mathbf{T}^{glo}(\bar{\mathbf{q}}^j, \mathbf{C}_\mathrm{i}{}^k)\right)^{-1} \mathbf{R}^{glo}(\bar{\mathbf{q}}^j, \mathbf{C}_\mathrm{i}{}^k)$;
  - $j++$;
  - calc $\mathbf{R}^{glo}(\bar{\mathbf{q}}^j, \mathbf{C}_\mathrm{i}{}^k)$ with $\mathbf{C}_\mathrm{i}{}^k$ update;
- **set:** $\mathbf{q}^{k+1} = \bar{\mathbf{q}}^j$ and $\mathbf{C}_\mathrm{i}{}^{k+1} = \mathbf{C}_\mathrm{i}{}^k$;
- **update:** $\mathbf{p}_{k+1}$

Appendix C. Consistent linearization

C.1. Material formulations

The material function ψ^{tot} can be split in several parts: A pure elastic part, a volumetric part for the elastic function, a viscous part, a volumetric part for the viscous part and at last an anisotropic part. For every material part different functions are available. The elastic and viscous functions depend on a NEO-HOOKIAN model for ψ^{ela}_1 and a SAINT-VENANT model ψ^{ela}_2. The volumetric parts can be found in [11]. The anisotropic part is listed in [1].

$$\psi^{\mathrm{tot}} = \psi^{\mathrm{ela}} + \psi^{\mathrm{vol}} + \psi^{\mathrm{vis}} + \psi^{\mathrm{vis,vol}} + \psi^{\mathrm{aniso}} \tag{C.1}$$

$$\psi^{\mathrm{ela}}_1 = \frac{\mu_1}{2}\left(\mathrm{tr}\,(\mathbf{C}) - 3 - 2\ln(J)\right) \qquad \psi^{\mathrm{ela}}_2 = \frac{\mu_2}{2}\left(\mathrm{tr}\left([\mathbf{C}-\mathbf{I}][\mathbf{C}-\mathbf{I}]^{\mathrm{T}}\right)\right) + \frac{\bar{\mu}_2}{8}\mathrm{tr}\,(\mathbf{C}-\mathbf{I})^2 \tag{C.2}$$

$$\psi^{\mathrm{vol}}_1 = \frac{\lambda_1}{2}\left(\ln(J)^2 + (J-1)^2\right) \qquad \psi^{\mathrm{vol}}_2 = \frac{\lambda_2}{2}\left(J^n - 2 + J^{-n}\right) \tag{C.3}$$

$$\psi^{\mathrm{vis}}_1 = \frac{\mu^{\mathrm{vis}}_1}{2}\left(\mathrm{tr}\left(\mathbf{C}\mathbf{C}_\mathrm{i}{}^{-1}\right) - 3 - 2\ln(J_\mathrm{e})\right) \tag{C.4}$$

$$\psi^{\mathrm{vis}}_2 = \frac{\mu^{\mathrm{vis}}_2}{2}\left(\mathrm{tr}\left([\mathbf{C}-\mathbf{C}_\mathrm{i}][\mathbf{C}-\mathbf{C}_\mathrm{i}]^{\mathrm{T}}\right)\right) + \frac{\bar{\mu}^{\mathrm{vis}}_2}{8}\mathrm{tr}\,(\mathbf{C}-\mathbf{C}_\mathrm{i})^2 \tag{C.5}$$

$$\psi^{\mathrm{vis,vol}}_1 = \frac{\lambda^{\mathrm{vis}}_1}{2}\left(\ln(J_\mathrm{e})^2 + (J_\mathrm{e}-1)^2\right) \qquad \psi^{\mathrm{vis,vol}}_2 = \frac{\lambda^{\mathrm{vis}}_2}{2}\left(J^n_\mathrm{e} - 2 + J^{-n}_\mathrm{e}\right) \tag{C.6}$$

$$\psi_k^{\text{aniso}} = \frac{\epsilon_1}{2\epsilon_2}\left(\exp\left[\epsilon_2(\text{tr}\,(\mathbf{C}\mathbf{M}_\text{F}) - 1)^2\right] - 1\right) \tag{C.7}$$

$$\mathbf{M}_\text{F} = \mathbf{a}_{0_k} \otimes \mathbf{a}_{0_k} \qquad \mathbf{a}_{0_k} = \frac{\mathbf{a}_k}{\|\mathbf{a}_k\|} \qquad \mathbb{V} = 2V^{\text{dev}}\mathbb{I}_{\text{dev}}^{\text{T}} + 3V^{\text{vol}}\mathbb{I}_{\text{vol}} \qquad \mathcal{E} = \frac{{\mathbf{C}_\text{i}}^{-1}}{4}\left(\mathbb{V} : [{\mathbf{C}_\text{i}}^{-1}\dot{\mathbf{C}}_\text{i}]\right) \tag{C.8}$$

$\mathcal{E}$ is described in [12] and represents a part of the evolution equation for the internal variable $\mathbf{C}_\text{i}$ and material parameters V^{vol} and V^{dev}.

$$\mathcal{E} = \frac{1}{4}\left(2V^{\text{dev}}{\mathbf{C}_\text{i}}^{-1}\dot{\mathbf{C}}_\text{i}{\mathbf{C}_\text{i}}^{-1} + \left[V^{\text{vol}} - \frac{2V^{\text{dev}}}{3}\right]{\mathbf{C}_\text{i}}^{-1}\text{tr}\left({\mathbf{C}_\text{i}}^{-1}\dot{\mathbf{C}}_\text{i}\right)\right) \tag{C.9}$$

C.2. Stress definitions

In this section the derivatives with respect to $\mathbf{C}$ and $\mathbf{C}_\text{i}$ are listed. From the derivatives the stress definition follows directly.

$$\frac{1}{2}\mathbf{S}_1^{\text{ela}} = \frac{\partial\psi_1^{\text{ela}}}{\partial\mathbf{C}} = \frac{\mu_1}{2}\left(\mathbf{I} - \mathbf{C}^{-1}\right) \qquad \frac{1}{2}\mathbf{S}_2^{\text{ela}} = \frac{\partial\psi_2^{\text{ela}}}{\partial\mathbf{C}} = \frac{\mu_2}{2}\left(\mathbf{C} - \mathbf{I}\right) + \frac{\bar{\mu}_2}{4}\text{tr}\left(\mathbf{C} - \mathbf{I}\right)\mathbf{I} \tag{C.10}$$

$$\frac{1}{2}\mathbf{S}_1^{\text{vol}} = \frac{\partial\psi_1^{\text{vol}}}{\partial\mathbf{C}} = \frac{\lambda_1}{2}\left(\ln(J) + J^2 - J\right)\mathbf{C}^{-1} \qquad \frac{1}{2}\mathbf{S}_2^{\text{vol}} = \frac{\partial\psi_2^{\text{vol}}}{\partial\mathbf{C}} = \frac{n\lambda_2}{4}\left(J^n - J^{-n}\right)\mathbf{C}^{-1} \tag{C.11}$$

$$\frac{1}{2}\mathbf{S}_1^{\text{vis}} = \frac{\partial\psi_1^{\text{vis}}}{\partial\mathbf{C}} = \frac{\mu_1^{\text{vis}}}{2}\left({\mathbf{C}_\text{i}}^{-1} - \mathbf{C}^{-1}\right) \tag{C.12}$$

$$\frac{1}{2}\mathbf{S}_2^{\text{vis}} = \frac{\partial\psi_2^{\text{vis}}}{\partial\mathbf{C}} = \frac{\mu_2^{\text{vis}}}{2}\left(\mathbf{C} - {\mathbf{C}_\text{i}}^{-1}\right) + \frac{\bar{\mu}_2^{\text{vis}}}{4}\text{tr}\left(\mathbf{C} - \mathbf{C}_\text{i}\right)\mathbf{I} \tag{C.13}$$

$$\frac{1}{2}\mathbf{S}_1^{\text{vis,vol}} = \frac{\partial\psi_1^{\text{vis,vol}}}{\partial\mathbf{C}} = -\frac{\lambda_1^{\text{vis}}}{2}\left(\ln(J_\text{e}) + J_\text{e}^2 - J_\text{e}\right)\mathbf{C}^{-1} \tag{C.14}$$

$$\frac{1}{2}\mathbf{S}_2^{\text{vis,vol}} = \frac{\partial\psi_2^{\text{vis,vol}}}{\partial\mathbf{C}} = -\frac{n\lambda_2^{\text{vis}}}{4}\left(J_\text{e}^n - J_\text{e}^{-n}\right)\mathbf{C}^{-1} \tag{C.15}$$

$$\frac{1}{2}\mathbf{S}_k^{\text{aniso}} = \frac{\partial\psi_k^{\text{aniso}}}{\partial\mathbf{C}} = \epsilon_1\left(\text{tr}\,(\mathbf{C}\mathbf{M}_\text{F}) - 1\right)\exp\left[\epsilon_2(\text{tr}\,(\mathbf{C}\mathbf{M}_\text{F}) - 1)^2\right]\mathbf{M}_\text{F} \tag{C.16}$$

$$\bar{\mathbf{S}}^e = \mathbf{S}_k^{\text{ela}} + \mathbf{S}_k^{\text{ela,vol}} + \mathbf{S}_k^{\text{vis}} + \mathbf{S}_k^{\text{vis,vol}} + \mathbf{S}_k^{\text{ansio}} \tag{C.17}$$

In the same way the derivatives with respect to $\mathbf{C}_\text{i}$ are defined as follows:

$$\hat{\mathbf{S}}_1^{\text{vis}} = \frac{\partial\psi_1^{\text{vis}}}{\partial\mathbf{C}_\text{i}} = \frac{\mu_1^{\text{vis}}}{2}\left({\mathbf{C}_\text{i}}^{-1} - {\mathbf{C}_\text{i}}^{-1}\mathbf{C}{\mathbf{C}_\text{i}}^{-1}\right) \tag{C.18}$$

$$\hat{\mathbf{S}}_2^{\text{vis}} = \frac{\partial\psi_2^{\text{vis}}}{\partial\mathbf{C}_\text{i}} = -\frac{\mu_2^{\text{vis}}}{2}\left(\mathbf{C} - \mathbf{C}_\text{i}\right) - \frac{\bar{\mu}_2^{\text{vis}}}{4}\text{tr}\left(\mathbf{C} - \mathbf{C}_\text{i}\right)\mathbf{I} \tag{C.19}$$

$$\hat{\mathbf{S}}_1^{\text{vis,vol}} = \frac{\partial\psi_1^{\text{vis,vol}}(\mathbf{C}_\text{i})}{\partial\mathbf{C}_\text{i}} = -\frac{\lambda_1^{\text{vis}}}{2}\left(\ln(J_\text{e}) + J_\text{e}^2 - J_\text{e}\right){\mathbf{C}_\text{i}}^{-1} \tag{C.20}$$

$$\hat{\mathbf{S}}_2^{\text{vis,vol}} = \frac{\partial\psi_2^{\text{vis,vol}}(\mathbf{C}_\text{i})}{\partial\mathbf{C}_\text{i}} = -\frac{n\lambda_2^{\text{vis}}}{4}\left(J_\text{e}^n - J_\text{e}^{-n}\right){\mathbf{C}_\text{i}}^{-1} \tag{C.21}$$

$$\hat{\mathbf{S}}^e = \hat{\mathbf{S}}_k^{\text{vis}} + \hat{\mathbf{S}}_k^{\text{vis,vol}}. \tag{C.22}$$

C.3. Second order derivatives

In this section we list the second order derivatives: At first, the second order derivatives with respect to $\mathbf{C}$ for the material function ψ^{tot}.

$$\mathbb{C}_1^{\text{ela}} = 4\frac{\partial^2\psi_1^{\text{ela}}}{\partial\mathbf{C}\partial\mathbf{C}} = 4\frac{\mu_1^{\text{ela}}}{2}\left(\mathbf{C}^{-1} \odot \mathbf{C}^{-1}\right) \tag{C.23}$$

$$\mathbb{C}_2^{\text{ela}} = 4\frac{\partial^2\psi_2^{\text{ela}}}{\partial\mathbf{C}\partial\mathbf{C}} = 4\frac{\mu_2^{\text{ela}}}{2}\left(\mathbf{I} \odot \mathbf{I}\right) + 4\frac{\bar{\mu}_2^{\text{ela}}}{4}\left(\mathbf{I} \otimes \mathbf{I}\right) \tag{C.24}$$

$$\mathbb{C}_1^{\text{ela,vol}} = 4\frac{\partial^2\psi_1^{\text{ela,vol}}}{\partial\mathbf{C}\partial\mathbf{C}} = 4\frac{\lambda_1^{\text{ela}}}{4}\left(2J^2 - J + 1\right)\mathbf{C}^{-1}\otimes\mathbf{C}^{-1} - 4\frac{\lambda_1^{\text{ela}}}{2}\left(\ln(J) + J^2 - J\right)\mathbf{C}^{-1}\odot\mathbf{C}^{-1} \tag{C.25}$$

$$\mathbb{C}_2^{\text{ela,vol}} = 4\frac{\partial^2\psi_2^{\text{ela,vol}}}{\partial\mathbf{C}\partial\mathbf{C}} = \frac{n^2\lambda_2^{\text{ela}}}{8}\left(J^n + J^{-n}\right)\mathbf{C}^{-1}\otimes\mathbf{C}^{-1} - \frac{n\lambda_2^{\text{ela}}}{4}\left(J^n - J^{-n}\right)\mathbf{C}^{-1}\odot\mathbf{C}^{-1} \tag{C.26}$$

$$\mathbb{C}_1^{\text{vis}} = 4\frac{\partial^2\psi_1^{\text{vis}}}{\partial\mathbf{C}\partial\mathbf{C}} = 4\frac{\mu_1^{\text{vis}}}{2}\left(\mathbf{C}^{-1}\odot\mathbf{C}^{-1}\right) \tag{C.27}$$

$$\mathbb{C}_2^{\text{vis}} = 4\frac{\partial^2\psi_2^{\text{vis}}}{\partial\mathbf{C}\partial\mathbf{C}} = 4\frac{\mu_2^{\text{vis}}}{2}\left(\mathbf{I}\odot\mathbf{I}\right) + 4\frac{\bar{\mu}_2^{\text{vis}}}{4}\left(\mathbf{I}\otimes\mathbf{I}\right) \tag{C.28}$$

$$\mathbb{C}_1^{\text{vis,vol}} = 4\frac{\partial^2\psi_1^{\text{vis,vol}}}{\partial\mathbf{C}\partial\mathbf{C}} = 4\frac{\lambda_1^{\text{vis}}}{4}\left(2J_{\text{e}}^2 - J_{\text{e}} + 1\right)\mathbf{C}_{\text{i}}{}^{-1}\otimes\mathbf{C}_{\text{i}}{}^{-1} - 4\frac{\lambda_1^{\text{vis}}}{2}\left(\ln(J_{\text{e}}) + J_{\text{e}}^2 - J_{\text{e}}\right)\mathbf{C}^{-1}\odot\mathbf{C}^{-1} \tag{C.29}$$

$$\mathbb{C}_2^{\text{vis,vol}} = 4\frac{\partial^2\psi_2^{\text{vis,vol}}}{\partial\mathbf{C}\partial\mathbf{C}} = \frac{n^2\lambda_2^{\text{vis}}}{8}\left(J_{\text{e}}^n + J_{\text{e}}^{-n}\right)\mathbf{C}_{\text{i}}{}^{-1}\otimes\mathbf{C}_{\text{i}}{}^{-1} - \frac{n\lambda_2^{\text{vis}}}{4}\left(J_{\text{e}}^n - J_{\text{e}}^{-n}\right)\mathbf{C}_{\text{i}}{}^{-1}\odot\mathbf{C}_{\text{i}}{}^{-1} \tag{C.30}$$

$$\mathbb{C}_k^{\text{aniso}} = 4\frac{\partial^2\psi_k^{\text{aniso}}}{\partial\mathbf{C}\partial\mathbf{C}} = 4\exp\left[\epsilon_2(\text{tr}\left(\mathbf{C}\mathbf{M}_{\text{F}}\right) - 1)^2\right]\left(\epsilon_1 + 2\epsilon_1\epsilon_2(\text{tr}\left(\mathbf{C}\mathbf{M}_{\text{F}}\right) - 1)^2\right)\mathbf{M}_{\text{F}}\otimes\mathbf{M}_{\text{F}} \tag{C.31}$$

$$\mathbb{C}^e = \mathbb{C}_k^{\text{ela}} + \mathbb{C}_k^{\text{ela,vol}} + \mathbb{C}_k^{\text{vis}} + \mathbb{C}_k^{\text{vis,vol}} + \mathbb{C}_k^{\text{ansio}} \tag{C.32}$$

The next equations are the second order derivatives with respect to $\mathbf{C}_{\text{i}}$ of the material function ψ^{tot}:

$$\hat{\mathbb{C}}_1^{\text{vis}} = \frac{\partial^2\psi_1^{\text{vis}}}{\partial\mathbf{C}_{\text{i}}\partial\mathbf{C}_{\text{i}}} = \frac{\mu_1^{\text{vis}}}{2}\left(\mathbf{C}_{\text{i}}{}^{-1}\odot\mathbf{C}_{\text{i}}{}^{-1}\mathbf{C}\mathbf{C}_{\text{i}}{}^{-1} + \mathbf{C}_{\text{i}}{}^{-1}\mathbf{C}\mathbf{C}_{\text{i}}{}^{-1}\odot\mathbf{C}_{\text{i}}{}^{-1} - \mathbf{C}_{\text{i}}{}^{-1}\odot\mathbf{C}_{\text{i}}{}^{-1}\right) \tag{C.33}$$

$$\hat{\mathbb{C}}_2^{\text{vis}} = \frac{\partial^2\psi_2^{\text{vis}}}{\partial\mathbf{C}_{\text{i}}\partial\mathbf{C}_{\text{i}}} = \frac{\mu_2^{\text{vis}}}{2}\left(\mathbf{I}\odot\mathbf{I}\right) + \frac{\bar{\mu}_2^{\text{vis}}}{4}\left(\mathbf{I}\otimes\mathbf{I}\right) \tag{C.34}$$

$$\hat{\mathbb{C}}_1^{\text{vis,vol}} = \frac{\partial^2\psi_1^{\text{vis,vol}}}{\partial\mathbf{C}_{\text{i}}\partial\mathbf{C}_{\text{i}}} = \frac{\lambda_1^{\text{vis}}}{4}\left(2J_{\text{e}}^2 - J_{\text{e}} + 1\right)\mathbf{C}_{\text{i}}{}^{-1}\otimes\mathbf{C}_{\text{i}}{}^{-1} + \frac{\lambda_1^{\text{vis}}}{2}\left(\ln(J_{\text{e}}) + J_{\text{e}}^2 - J_{\text{e}}\right)\mathbf{C}^{-1}\odot\mathbf{C}^{-1} \tag{C.35}$$

$$\hat{\mathbb{C}}_2^{\text{vis,vol}} = \frac{\partial^2\psi_2^{\text{vis,vol}}}{\partial\mathbf{C}_{\text{i}}\partial\mathbf{C}_{\text{i}}} = \frac{n^2\lambda_2^{\text{vis}}}{8}\left(J_{\text{e}}^n + J_{\text{e}}^{-n}\right)\mathbf{C}_{\text{i}}{}^{-1}\otimes\mathbf{C}_{\text{i}}{}^{-1} + \frac{n\lambda_2^{\text{vis}}}{4}\left(J_{\text{e}}^n - J_{\text{e}}^{-n}\right)\mathbf{C}_{\text{i}}{}^{-1}\odot\mathbf{C}_{\text{i}}{}^{-1} \tag{C.36}$$

$$\hat{\mathbb{C}}^e = \hat{\mathbb{C}}_k^{\text{vis}} + \hat{\mathbb{C}}_k^{\text{vis,vol}} \tag{C.37}$$

and the derivatives of the D'Alembert terms with respect to $\mathbf{C}_{\text{i}}$ of the evolution equation:

$$\hat{\mathbb{E}}_{\text{d}}^e = \frac{\partial\mathcal{E}(\mathbf{C}_{\text{i}},\dot{\mathbf{C}}_{\text{i}})}{\partial\dot{\mathbf{C}}_{\text{i}}} = \frac{2V^{\text{dev}}}{4}\left(\mathbf{C}_{\text{i}}{}^{-1}\odot\mathbf{C}_{\text{i}}{}^{-1}\right) + \frac{1}{4}\left(V^{\text{vol}} - \frac{2V^{\text{dev}}}{3}\right)\left(\mathbf{C}_{\text{i}}{}^{-1}\otimes\mathbf{C}_{\text{i}}{}^{-1}\right) \tag{C.38}$$

$$\begin{aligned}\hat{\mathbb{E}}^e = \frac{\partial\mathcal{E}(\mathbf{C}_{\text{i}},\dot{\mathbf{C}}_{\text{i}})}{\partial\mathbf{C}_{\text{i}}} = \mathbf{C}_{\text{i}}\odot\Bigg[&-\frac{2V^{\text{dev}}}{4}\mathbf{C}_{\text{i}}{}^{-1}\dot{\mathbf{C}}_{\text{i}}\mathbf{C}_{\text{i}}{}^{-1} + \\ &+\left(\frac{2}{4}V^{\text{dev}} - \frac{1}{4}\text{tr}\left(\mathbf{C}_{\text{i}}{}^{-1}\dot{\mathbf{C}}_{\text{i}}\right)\left(V^{\text{vol}} - \frac{2V^{\text{dev}}}{3}\right)\right)\mathbf{C}_{\text{i}}{}^{-1}\Bigg] \\ &+\left(-\frac{2V^{\text{dev}}}{4}\mathbf{C}_{\text{i}}{}^{-1}\dot{\mathbf{C}}_{\text{i}}\mathbf{C}_{\text{i}}{}^{-1}\right)\odot\mathbf{C}_{\text{i}}{}^{-1} \\ &+\mathbf{C}_{\text{i}}{}^{-1}\otimes\left(-\frac{1}{4}\left[V^{\text{vol}} - \frac{2V^{\text{dev}}}{3}\right]\mathbf{C}_{\text{i}}{}^{-1}\dot{\mathbf{C}}_{\text{i}}\mathbf{C}_{\text{i}}{}^{-1}\right)\end{aligned} \tag{C.39}$$

At last the mixed derivatives with respect to $\mathbf{C}$ and $\mathbf{C}$ are given by:

$$\overline{\mathbb{C}}_1^{\text{vis}} = 2\frac{\partial^2 \psi_1^{\text{vis}}}{\partial \mathbf{C} \partial \mathbf{C}_\text{i}} = -2\frac{\mu_1^{\text{vis}}}{2}\left(\mathbf{C}_\text{i}^{-1} \odot \mathbf{C}_\text{i}^{-1}\right) \qquad \tilde{\mathbb{C}}_1^{\text{vis}} = 2\frac{\partial^2 \psi_1^{\text{vis}}}{\partial \mathbf{C}_\text{i} \partial \mathbf{C}} = -2\frac{\mu_1^{\text{vis}}}{2}\left(\mathbf{C}_\text{i}^{-1} \odot \mathbf{C}_\text{i}^{-1}\right) \tag{C.40}$$

$$\overline{\mathbb{C}}_2^{\text{vis}} = 2\frac{\partial^2 \psi_2^{\text{vis}}}{\partial \mathbf{C} \partial \mathbf{C}_\text{i}} = -2\frac{\mu_2^{\text{vis}}}{2}\left(\mathbf{I} \odot \mathbf{I}\right) - 2\frac{\bar{\mu}_2^{\text{vis}}}{4}\left(\mathbf{I} \otimes \mathbf{I}\right) \tag{C.41}$$

$$\tilde{\mathbb{C}}_2^{\text{vis}} = 2\frac{\partial^2 \psi_2^{\text{vis}}}{\partial \mathbf{C}_\text{i} \partial \mathbf{C}} = -2\frac{\mu_2^{\text{vis}}}{2}\left(\mathbf{I} \odot \mathbf{I}\right) - 2\frac{\bar{\mu}_2^{\text{vis}}}{4}\left(\mathbf{I} \otimes \mathbf{I}\right) \tag{C.42}$$

$$\overline{\mathbb{C}}_1^{\text{vis,vol}} = 2\frac{\partial^2 \psi_1^{\text{vis,vol}}}{\partial \mathbf{C} \partial \mathbf{C}_\text{i}} = -2\frac{\lambda_1^{\text{vis}}}{2}\left(2J_\text{e}^2 - J_\text{e} + 1\right)\left(\mathbf{C}_\text{i}^{-1} \otimes \mathbf{C}^{-1}\right) \tag{C.43}$$

$$\tilde{\mathbb{C}}_1^{\text{vis,vol}} = 2\frac{\partial^2 \psi_1^{\text{vis,vol}}}{\partial \mathbf{C}_\text{i} \partial \mathbf{C}} = -2\frac{\lambda_1^{\text{vis}}}{2}\left(2J_\text{e}^2 - J_\text{e} + 1\right)\left(\mathbf{C}^{-1} \otimes \mathbf{C}_\text{i}^{-1}\right) \tag{C.44}$$

$$\overline{\mathbb{C}}_2^{\text{vis,vol}} = 2\frac{\partial^2 \psi_2^{\text{vis,vol}}}{\partial \mathbf{C} \partial \mathbf{C}_\text{i}} = -2\frac{n^2\lambda_2^{\text{vis}}}{8}\left(J_\text{e}^n - J_\text{e}^{-n}\right)\left(\mathbf{C}_\text{i}^{-1} \otimes \mathbf{C}^{-1}\right) \tag{C.45}$$

$$\tilde{\mathbb{C}}_2^{\text{vis,vol}} = 2\frac{\partial^2 \psi_2^{\text{vis,vol}}}{\partial \mathbf{C}_\text{i} \partial \mathbf{C}} = -2\frac{n^2\lambda_2^{\text{vis}}}{8}\left(J_\text{e}^n - J_\text{e}^{-n}\right)\left(\mathbf{C}^{-1} \otimes \mathbf{C}_\text{i}^{-1}\right) \tag{C.46}$$

$$\tilde{\mathbb{C}}^e = \tilde{\mathbb{C}}_k^{\text{vis}} + \tilde{\mathbb{C}}_k^{\text{vis,vol}} \qquad \overline{\mathbb{C}}^e = \overline{\mathbb{C}}_k^{\text{vis}} + \overline{\mathbb{C}}_k^{\text{vis,vol}} \tag{C.47}$$

In the previous equations we skipped the dependencies but at this point we introduce the dependencies for a better understanding.

$$\mathbb{C}^e = \mathbb{C}^e\left(\bar{\mathbf{C}}^{e,\text{h}}, \mathbf{C}_\text{i}^{e,\text{h}}\right) \qquad \tilde{\mathbb{C}}^e = \tilde{\mathbb{C}}^e\left(\bar{\mathbf{C}}^{e,\text{h}}, \mathbf{C}_\text{i}^{e,\text{h}}\right) \qquad \overline{\mathbb{C}}^e = \overline{\mathbb{C}}^e\left(\bar{\mathbf{C}}^{e,\text{h}}, \mathbf{C}_\text{i}^{e,\text{h}}\right) \tag{C.48}$$

$$\hat{\mathbb{C}}^e = \hat{\mathbb{C}}^e\left(\bar{\mathbf{C}}^{e,\text{h}}, \mathbf{C}_\text{i}^{e,\text{h}}\right) \qquad \hat{\mathbf{S}}^e = \hat{\mathbf{S}}^e\left(\bar{\mathbf{C}}^{e,\text{h}}, \mathbf{C}_\text{i}^{e,\text{h}}\right) \qquad \bar{\mathbf{S}}^e = \bar{\mathbf{S}}^e\left(\bar{\mathbf{C}}^{e,\text{h}}, \mathbf{C}_\text{i}^{e,\text{h}}\right) \tag{C.49}$$

$$\hat{\mathbb{E}}_\text{d}^e = \hat{\mathbb{E}}_\text{d}^e\left(\mathbf{C}_\text{i}^{e,\text{h}}, \mathbf{C}_\text{i}^{e,\text{d}}\right) \qquad \mathcal{E}^e = \mathcal{E}^e\left(\mathbf{C}_\text{i}^{e,\text{h}}, \mathbf{C}_\text{i}^{e,\text{d}}\right) \qquad \hat{\mathbb{E}}^e = \hat{\mathbb{E}}^e\left(\mathbf{C}_\text{i}^{e,\text{h}}\right) \tag{C.50}$$

$$\mathcal{D}^\text{e} = \mathcal{D}^\text{e}\left(\bar{\mathbf{C}}^{e,\text{h}}, \bar{\mathbf{C}}^{e,\text{d}}, \mathbf{C}^{e,\text{hd}}, \mathbf{C}_\text{i}^{e,\text{h}}, \mathbf{C}_\text{i}^{e,\text{d}}\right) \tag{C.51}$$

The following equations list the global and local residual equations with the associated tangent terms. Under every equation the associated dimension is given:

$$\mathbf{R}_{\text{Ci}}^{\text{loc}} = h_n \sum_{g=1}^{n_\text{pg}} w_g M_{\frac{i}{n_\text{pg}}}\left[\hat{\mathbf{S}}^e + \mathcal{E}^e\right]\Big|_{i=0}^{n_\text{pg}-1} \tag{C.52}$$

$$\mathbf{R}_{\text{Ci}}^{\text{loc}} = \mathbb{R}^{6n_\text{pg}\times 1} \tag{C.53}$$

$$\begin{aligned}\mathbf{R}_{\text{q}}^{\text{glo},e} = \mathbf{p}_i^e\Big|_{i=0}^{0} - \int_{\mathcal{B}_0^e} h_n \sum_{g=1}^{n_\text{pg}} w_g M_{\frac{i}{n_\text{pg}}}\left[\overset{\text{BS}}{\mathcal{P}}_\text{M}\left\{\mathbf{F}(\mathbf{q}^{e,\text{h}})\,,\ \bar{\mathbf{S}}^e + \mathcal{D}^\text{e}\bar{\mathbf{C}}^{e,\text{d}}\right\} - \rho_0\mathbf{g}^e\right]\text{d}V\Big|_{i=0}^{n_\text{pg}-1} \\ + \left[\sum_{g=1}^{n_\text{pg}} w_g h_n \dot{M}_{\frac{i}{n_\text{pg}}}\mathbf{M}^e \cdot \mathbf{q}^{e,\text{d}}\right]\Big|_{i=0}^{n_\text{pg}-1} + h_n\mathbf{F}^{\text{ext},e} - h_n\left(\frac{\partial\phi(\mathbf{q}^e)}{\partial\mathbf{q}^e}\right)^\text{T}\boldsymbol{\lambda}^e\end{aligned} \tag{C.54}$$

$$\mathbf{R}_{\text{q}}^{\text{glo},e} = \mathbb{R}^{3n_\text{no}n_\text{pg}\times 1} \tag{C.55}$$

$$\mathbf{T}_{\text{Ci}}^{\text{loc}} = h_n \sum_{g=1}^{n_\text{pg}} w_g M_{\frac{i}{n_\text{pg}}}\left[M_{\frac{j+1}{n_\text{pg}}}\left(\hat{\mathbb{C}}^e + \hat{\mathbb{E}}^e\right) + \dot{M}_{\frac{j+1}{n_\text{pg}}}\hat{\mathbb{E}}_\text{d}^e\right]\Big|_{i,j=0}^{n_\text{pg}-1} \tag{C.56}$$

$$\mathbf{T}_{\text{Ci}}^{\text{loc}} = \mathbb{R}^{6n_\text{pg}\times 6n_\text{pg}} \tag{C.57}$$

$$\mathbf{T}_{\mathrm{q}}^{\mathrm{loc}} = h_n \sum_{g=1}^{n_{\mathrm{pg}}} w_g M_{\frac{i}{n_{\mathrm{pg}}}} \left[M_{\frac{j+1}{n_{\mathrm{pg}}}} \overline{\mathbb{C}}^e \cdot \overset{\mathrm{B}}{\mathcal{P}}_{\mathrm{M}} \left\{ \mathbf{F}(\mathbf{q}^e{}_{\frac{j+1}{n_{\mathrm{pg}}}}) \right\} \right] \Big|_{i,j=0}^{n_{\mathrm{pg}}-1} \tag{C.58}$$

$$\mathbf{T}_{\mathrm{q}}^{\mathrm{loc}} = \mathbb{R}^{6n_{\mathrm{pg}} \times 3n_{\mathrm{no}} n_{\mathrm{pg}}} \tag{C.59}$$

$$\begin{aligned}
\mathbf{T}_{\mathrm{Ci}}^{\mathrm{glo}} = & -h_n \sum_{g=1}^{n_{\mathrm{pg}}} w_g M_{\frac{i}{n_{\mathrm{pg}}}} \left[M_{\frac{j+1}{n_{\mathrm{pg}}}} \overset{\mathrm{B}}{\mathcal{P}}_{\mathrm{M}} \left\{ \mathbf{F}(\mathbf{q}^{e,\mathrm{h}}) \right\}^{\mathrm{T}} \cdot \tilde{\mathbb{C}}^e \right] \Big|_{i,j=0}^{n_{\mathrm{pg}}-1} \\
& - h_n \sum_{g=1}^{n_{\mathrm{pg}}} w_g M_{\frac{i}{n_{\mathrm{pg}}}} \Bigg[\overset{\mathrm{BS}}{\mathcal{P}}_{\mathrm{M}} \left\{ \mathbf{F}\left(\mathbf{q}^{e,\mathrm{h}}\right),\ \bar{\mathbf{C}}^{e,\mathrm{d}} \right\} \cdot \Bigg(\\
& \quad - \frac{2}{\mathcal{N}^{\mathrm{e}}} \sum_{\hat{g}=1}^{n_{\mathrm{pg}}} w_{\hat{g}} M_{\frac{j}{n_{\mathrm{pg}}}} \left\{ \hat{\mathbb{C}}^e \cdot \mathbf{V} \cdot \mathbf{C}_{\mathrm{i}}^{e,\mathrm{d}} + \tilde{\mathbb{C}}^e \cdot \mathbf{V} \cdot \mathbf{C}^{e,\mathrm{hd}} \right\} \\
& \quad - \frac{2}{\mathcal{N}^{\mathrm{e}}} \sum_{\hat{g}=1}^{n_{\mathrm{pg}}} w_{\hat{g}} \dot{M}_{\frac{j}{n_{\mathrm{pg}}}} \hat{\mathbf{S}}^e \Bigg) \Bigg] \Bigg|_{i,j=0}^{n_{\mathrm{pg}}-1} \\
& - \frac{2}{\mathcal{N}^{\mathrm{e}}} h_n \sum_{g=1}^{n_{\mathrm{pg}}} w_g M_{\frac{i}{n_{\mathrm{pg}}}} \left[\overset{\mathrm{BS}}{\mathcal{P}}_{\mathrm{M}} \left\{ \mathbf{F}\left(\mathbf{q}^{e,\mathrm{h}}\right),\ \bar{\mathbf{C}}^{e,\mathrm{d}} \right\} \cdot \hat{\mathbf{S}}^e(\mathbf{C}_1, \mathbf{C}_{\mathrm{i}1}) \right] \Big|_{i,j=n_{\mathrm{pg}}-1}^{n_{\mathrm{pg}}-1}
\end{aligned} \tag{C.60}$$

$$\mathbf{T}_{\mathrm{Ci}}^{\mathrm{glo}} = \mathbb{R}^{3n_{\mathrm{no}} n_{\mathrm{pg}} \times 6n_{\mathrm{pg}}} \tag{C.61}$$

$$\begin{aligned}
\mathbf{T}_{\mathrm{q}}^{\mathrm{glo},e} = & - h_n \sum_{g=1}^{n_{\mathrm{pg}}} w_g M_{\frac{i}{n_{\mathrm{pg}}}} \Bigg(M_{\frac{j+1}{n_{\mathrm{pg}}}} \overset{\mathrm{SST}}{\mathcal{P}}_{\mathrm{M}} \left\{ \bar{\mathbf{S}}^e + \mathcal{D}^{\mathrm{e}} \bar{\mathbf{C}}^{e,\mathrm{d}} \right\} \\
& - \overset{\mathrm{B}}{\mathcal{P}}_{\mathrm{M}} \left\{ \mathbf{F}\left(\mathbf{q}^{e,\mathrm{h}}\right) \right\}^{\mathrm{T}} \cdot \left[M_{\frac{j+1}{n_{\mathrm{pg}}}} \mathbb{C}^e + 2\mathcal{D}^{\mathrm{e}} \dot{M}_{\frac{j+1}{n_{\mathrm{pg}}}} \mathbf{V}^{-1} \right] \cdot \overset{\mathrm{B}}{\mathcal{P}}_{\mathrm{M}} \left\{ \mathbf{F}\left(\mathbf{q}^e_{\frac{j+1}{n_{\mathrm{pg}}}} \right) \right\} \\
& - \overset{\mathrm{BS}}{\mathcal{P}}_{\mathrm{M}} \left\{ \mathbf{F}\left(\mathbf{q}^{e,\mathrm{h}}\right),\ \bar{\mathbf{C}}^{e,\mathrm{d}} \right\} \cdot \Bigg[\\
& - \sum_{\hat{g}=1}^{n_{\mathrm{pg}}} w_{\hat{g}} \overset{\mathrm{BS}}{\mathcal{P}}_{\mathrm{M}} \left\{ \mathbf{F}\left(\mathbf{q}^e_{\frac{j+1}{n_{\mathrm{pg}}}} \right),\ \frac{2\mathcal{D}^{\mathrm{e}}}{\mathcal{N}^{\mathrm{e}}} \dot{M}_{\frac{j+1}{n_{\mathrm{pg}}}} \mathbf{C}^{e,\mathrm{hd}} + \frac{1}{\mathcal{N}^{\mathrm{e}}} M_{\frac{j+1}{n_{\mathrm{pg}}}} \left(\left[\mathbb{C}^e \right]^{\mathrm{T}} \cdot \mathbf{V} \cdot \mathbf{C}^{e,\mathrm{hd}} + \left[\overline{\mathbb{C}}^e \right]^{\mathrm{T}} \cdot \mathbf{V} \cdot \mathbf{C}_{\mathrm{i}}^{e,\mathrm{d}} \right) \right\} \\
& - \sum_{\hat{g}=1}^{n_{\mathrm{pg}}} w_{\hat{g}} \overset{\mathrm{BS}}{\mathcal{P}}_{\mathrm{M}} \left\{ \mathbf{F}\left(\mathbf{q}^{e,\mathrm{h}}\right),\ \frac{2\mathcal{D}^{\mathrm{e}}}{\mathcal{N}^{\mathrm{e}}} \dot{M}_{\frac{j+1}{n_{\mathrm{pg}}}} \bar{\mathbf{C}}^{e,\mathrm{d}} + \frac{2}{\mathcal{N}^{\mathrm{e}}} \dot{M}_{\frac{j+1}{n_{\mathrm{pg}}}} \cdot \bar{\mathbf{S}}^e \right\} \\
& - \sum_{\hat{g}=1}^{n_{\mathrm{pg}}} w_{\hat{g}} \overset{\mathrm{BS}}{\mathcal{P}}_{\mathrm{M}} \left\{ \mathbf{F}\left(\mathbf{q}^{e,\mathrm{d}}\right),\ \frac{2\mathcal{D}^{\mathrm{e}}}{\mathcal{N}^{\mathrm{e}}} M_{\frac{j+1}{n_{\mathrm{pg}}}} \bar{\mathbf{C}}^{e,\mathrm{d}} + \frac{2}{\mathcal{N}^{\mathrm{e}}} M_{\frac{j+1}{n_{\mathrm{pg}}}} \cdot \bar{\mathbf{S}}^e \right\} \Bigg]^{\mathrm{T}} \Bigg) \Bigg|_{i,j=0}^{n_{\mathrm{pg}}-1} \\
& - \frac{2}{\mathcal{N}^{\mathrm{e}}} h_n \sum_{g=1}^{n_{\mathrm{pg}}} w_g M_{\frac{i}{n_{\mathrm{pg}}}} \overset{\mathrm{BS}}{\mathcal{P}}_{\mathrm{M}} \left\{ \mathbf{F}\left(\mathbf{q}^{e,\mathrm{h}}\right),\ \bar{\mathbf{C}}^{e,\mathrm{d}} \right\} \cdot \overset{\mathrm{BS}}{\mathcal{P}}_{\mathrm{M}} \left\{ \mathbf{F}\left(\mathbf{q}_1^e\right),\ \bar{\mathbf{S}}^e\left(\mathbf{q}_1^e\right) \right\}^{\mathrm{T}} \Big|_{i=0,j=n_{\mathrm{pg}}-1}^{n_{\mathrm{pg}}-1} \\
& + h_n \sum_{g=1}^{n_{\mathrm{pg}}} w_g \dot{M}_{\frac{i}{n_{\mathrm{pg}}}} \dot{M}_{\frac{j+1}{n_{\mathrm{pg}}}} \mathbf{M}^e \Big|_{i=0,j=0}^{n_{\mathrm{pg}}-1} + \mathbf{T}_{\mathrm{ext}}^{\mathrm{glo},e}
\end{aligned}$$

$$\mathbf{T}_{\mathrm{q}}^{\mathrm{glo},e} = \mathbb{R}^{3n_{\mathrm{no}} n_{\mathrm{pg}} \times 3n_{\mathrm{no}} n_{\mathrm{pg}}} \tag{C.62}$$

The global tangent of the system is given by:

$$\mathbf{T} = \begin{bmatrix} \bigcup_{e=1}^{n_{\text{el}}} \int_{\mathcal{B}_0^e} \mathbf{T}_{\text{q}}^{\text{glo},e} - \mathbf{T}_{\text{Ci}}^{\text{glo}} \left(\mathbf{T}_{\text{Ci}}^{\text{loc}}\right)^{-1} \mathbf{T}_{\text{q}}^{\text{loc}} \, \mathrm{d}V & \mathbf{G} \\ \tilde{\mathbf{G}} & \mathbf{0} \end{bmatrix} \tag{C.63}$$

with the parts of the derivatives of the LAGRANGE multiplier terms.

$$\mathbf{G} = -h_n \left(\frac{\partial \boldsymbol{\phi}(\check{\mathbf{q}}_{k+1})}{\partial \check{\mathbf{q}}_{k+1}} \right)^{\mathrm{T}} \tag{C.64}$$

$$\tilde{\mathbf{G}} = \frac{\partial \boldsymbol{\phi}(\check{\mathbf{q}}_{k+1})}{\partial \check{\mathbf{q}}_{k+1}} \tag{C.65}$$

Appendix D. Shape functions in time

The shape functions and the derivatives with respect to α for second order and a fourth order time step scheme are listed in Table D.7.

Table D.7
LAGRANGE polynomials for equidistant points in the interval between zero and one.

Polynomial degree 1		
Shape function	Derivative of the shape function	Points
$M_0 = 1 - \alpha$	$M_0' = -1$	$\xi_0 = 0$
$M_1 = \alpha$	$M_1' = 1$	$\xi_1 = 1$
Polynomial degree 2		
$M_0 = (2\alpha - 1)(\alpha - 1)$	$M_0' = 4\alpha - 3$	$\xi_0 = 0$
$M_{\frac{1}{2}} = -4\alpha(\alpha - 1)$	$M_{\frac{1}{2}}' = 4 - 8\alpha$	$\xi_1 = 0.5$
$M_1 = 2\alpha\left(\alpha - \frac{1}{2}\right)$	$M_1' = 4\alpha - 1$	$\xi_2 = 1$
Polynomial degree 3		
$M_0 = -1.5(3\alpha - 1)(\alpha - \frac{2}{3})(\alpha - 1)$	$M_0' = -13.5\alpha^2 + 18\alpha - 5.5$	$\xi_0 = 0$
$M_{\frac{1}{3}} = 13.5\alpha(\alpha - \frac{2}{3})(\alpha - 1)$	$M_{\frac{1}{3}}' = 40.5\alpha^2 - 45\alpha + 9$	$\xi_1 = \frac{1}{3}$
$M_{\frac{2}{3}} = -13.5\alpha(\alpha - \frac{1}{3})(\alpha - 1)$	$M_{\frac{2}{3}}' = -40.5\alpha^2 + 36\alpha - 4.5$	$\xi_2 = \frac{2}{3}$
$M_1 = 4.5\alpha(\alpha - \frac{1}{3})(\alpha - \frac{2}{3})$	$M_1' = 13.5\alpha^2 - 9\alpha + 1$	$\xi_3 = 1$

References

[1] D. Balzani, J. Schröder, P. Neff, Applications of anisotropic polyconvex energies: thin shells and biomechanics of arterial walls, in: Poly-, Quasi-and Rank-One Convexity in Applied Mechanics, Springer, 2010, pp. 131–175.

[2] M. Bartelt, J. Dietzsch, M. Groß, Efficient implementation of energy conservation for higher order finite elements with variational integrators, Math. Comput. Simulation 150 (2018) 83–121, http://dx.doi.org/10.1016/j.matcom.2018.03.002.

[3] R. Bell, G. Houlsby, H. Burd, Suitability of three-dimensional finite elements for modelling material incompressibility using exact integration, Int. J. Numer. Methods Biomed. Eng. 9 (4) (1993) 313–329.

[4] P. Betsch, P. Steinmann, Conservation properties of a time FE method—part II: Time-stepping schemes for non-linear elastodynamics, Internat. J. Numer. Methods Engrg. 50 (8) (2001) 1931–1955, http://dx.doi.org/10.1002/nme.103.

[5] CUBLAS Library, ninth ed., NVIDIA, 2018.

[6] CUDA C Programming Guide, ninth ed., NVIDIA, 2018.

[7] CUDA C Best Practices Guide, ninth ed., NVIDIA, 2018.

[8] F. Fritzen, M. Hodapp, M. Leuschner, GPU Accelerated computational homogenization based on a variational approach in a reduced basis framework, Comput. Methods Appl. Mech. Engrg. 278 (2014) 186–217, http://dx.doi.org/10.1016/j.cma.2014.05.006.

[9] O. Gonzalez, Time integration and discrete hamiltonian systems, J. Nonlinear Sci. 6 (5) (1996) 449, http://dx.doi.org/10.1007/BF02440162.

[10] M. Groß, P. Betsch, P. Steinmann, Conservation properties of a time FE method. Part IV: Higher order energy and momentum conserving schemes, Internat. J. Numer. Methods Engrg. 63 (13) (2005) 1849–1897, http://dx.doi.org/10.1002/nme.1339.

[11] S. Hartmann, P. Neff, Polyconvexity of generalized polynomial-type hyperelastic strain energy functions for near-incompressibility, Int. J. Solids Struct. 40 (11) (2003) 2767–2791, http://dx.doi.org/10.1016/S0020-7683(03)00086-6.
[12] M. Krüger, M. Groß, P. Betsch, An energy-entropy-consistent time stepping scheme for nonlinear thermo-viscoelastic continua, ZAMM Z. Angew. Math. Mech. (2014) 1–38, http://dx.doi.org/10.1002/zamm.201300268.
[13] J.E. Marsden, M. West, Discrete mechanics and variational integrators, Acta Numer. 10 (2001) 357–514, http://dx.doi.org/10.1017/S096249290100006X.
[14] X. Mei, X. Chu, Dissecting GPU memory hierarchy through microbenchmarking, IEEE Trans. Parallel Distrib. Syst. 28 (11) (2017) 72–86, http://dx.doi.org/10.1109/TPDS.2016.2549523.
[15] J.C. Nagtegaal, D.M. Parks, J. Rice, On numerically accurate finite element solutions in the fully plastic range, Comput. Methods Appl. Mech. Engrg. 4 (2) (1974) 153–177.
[16] S. Ober-Blöbaum, N. Saake, Construction and analysis of higher order Galerkin variational integrators, Adv. Comput. Math. (2014) 1–32, http://dx.doi.org/10.1007/s10444-014-9394-8.
[17] V. Volkov, Better performance at lower occupancy, in: GPU Technology Conference, UC Berkeley, 2010.
[18] P. Šolín, K. Segeth, I. Doleźel, Higher Order Finite Element Methods, Chapman & Hall CRC, Boca Raton, Fla. [u.a.], 2004, XX, 382 S.
[19] P. Wang, Fundamental optimizations in CUDA, in: GPU Technology Conference, NVIDIA, 2012.
[20] T. Wenger, S. Ober-Blöbaum, S. Leyendecker, Construction and analysis of higher order variational integrators for dynamical systems with holonomic constraints, Adv. Comput. Math. 43 (5) (2017) 1163–1195, http://dx.doi.org/10.1007/s10444-017-9520-5.
[21] P. Wriggers, Nichtlineare Finite-Element-Methoden, Springer Berlin Heidelberg, 2013.