Material point method: basics and applications

Data · May 2014

CITATIONS
2

READS
4,337

1 author:

Vinh Phu Nguyen
Monash University (Australia)
76 PUBLICATIONS **1,895** CITATIONS

SEE PROFILE

Material point method: basics and applications

Vinh Phu Nguyen
Cardiff Unitversity
Department of Civil Engineering
Institute of Advanced Mechanics and Materials

Contents

1 Lagrangian, Eulerian and Arbitrary Lagrangian-Eulerian descriptions **2**

2 Introduction **3**

3 Governing equations **5**

4 Weak form and discretisation **5**

5 Explicit time integration **7**

6 Implementation **10**
6.1 Eulerian grid 14
6.2 Shape functions 15
6.2.1 B-splines basis functions 16
6.3 Initial particle distribution 16
6.4 Visualisation 18

7 Examples **18**
7.1 1D examples 18
7.2 Axial vibration of a continuum bar 22
7.3 Impact of two elastic bodies 24
7.4 Impact of two elastic rings 31

8 Generalized Interpolation Material Point Method-GIMP **32**

9 Implicit time integration **34**

10 Fluid-structure interaction **36**

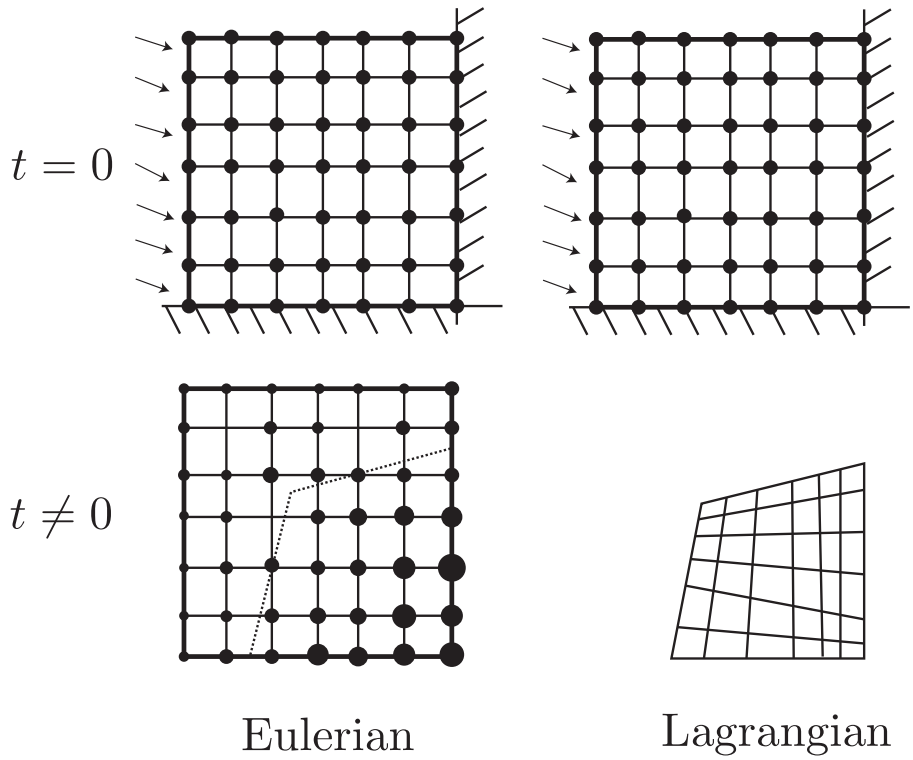

Figure 1: Eulerian vs Lagrangian description.

1 Lagrangian, Eulerian and Arbitrary Lagrangian-Eulerian descriptions

In Eulerian formulations the governing equations are solved on the fixed grid, and material moves through the mesh. In Lagrangian methods, the computational grid or mesh is attached to the material being simulated. The Lagrangian mesh moves and distorts with the material.

From Fig. 1 it is obvious that the most distinct advantage of the Eulerian description is the ability to deal with highly distorted motion since the grid is always fixed. However it suffers from the following disadvantages (1) difficulties with history dependent materials because material points are not tracked, (2) difficulties in defining material boundaries. The stress of fluids is independent of its history, the Eulerian description is then widely adopted in describing the motion of fluids. Furthermore, the material time derivative in an Eulerian description is the sum of a spatial time derivative and a convective term (this convective term is vanished in a Lagrangian formulation). This term is generally expensive to handle from a computational perspective.

There are also some mixed methods which combine the advantages of these two descriptions and to avoid their disadvantages, such as the arbitrary Lagrangian-Eulerian (ALE). In ALE, the computational mesh is continuously moved independently of the material deformation to optimize element shapes and describe the boundaries accurately. However, the convection term still exists in the ALE formulation, which may cause numerical difficulties. In addition, it

still remains a challenging task to design an efficient and effective mesh moving algorithm to maintain mesh regularity for three-dimensional complicated material domain. The Material Point Method (MPM) was born as a simple ALE method.

2 Introduction

The Material Point Method (MPM) is one of the latest developments in particle-in-cell (PIC) methods. The first PIC technique was developed in the early 1950s and was used primarily for applications in fluid mechanics. Early implementations suffered from excessive energy dissipation which was overcome in 1986, by Brackbill and Ruppel and they introduced FLIP-the Fluid Implicit Particle method. The FLIP technique was modified and tailored for applications in solid mechanics by Sulsky and her co-workers Sulsky et al. [1994, 1995] and has since been referred to as the material point method (MPM) Sulsky and Schreyer [1996].

The MPM discretizes a continuum body Ω with a finite set of n_p material points (or particles) in the original configuration that are tracked throughout the deformation process. The terms particle and material point will be used interchangeably throughout this manuscript. The mass and volume of subregions which the particles represent are memorised for these points, but changes in the shape of the subregions are not traced[1]. Let $\mathbf{x}_p^t$ $(p = 1, 2, \ldots, n_p)$ denote the current position of material point p at time t. Each material point has an associated mass M_p, density ρ_p, velocity $\mathbf{v}_p$, Cauchy stress tensor $\boldsymbol{\sigma}_p$ and any other internal state variables necessary for the constitutive model. Thus, these material points provide a Lagrangian description of the continuum body. Since each material point contains a fixed amount of mass for all time, mass conservation is automatically satisfied. In MPM, a fixed Eulerian grid is used where the equation of balance of momentum is solved. This is in contrast to other meshfree/particle methods where the momentum equations are solved on the particles. We refer to Fig 2 for a graphical illustration of the discretisation in MPM. The grid covers the solid in its initial configuration as well as the region in which the solid is expected to move. The particles interact with other particles in the same body, with other solid bodies, or with fluids through a background Eulerian grid. In general, after each time step the grid is discarded i.e., it is reset to the initial configuration. This is the key difference between MPM and the updated Lagrange FEM, cf. Fig. 3 and makes the method distortion-free. The MPM for solid mechanics conserves mass and momentum by construction, but energy conservation is not explicitly enforced. Advantages of the MPM algorithm include the absence of mesh-tangling problems and error-free advection of material properties via the motion of the material points. Yet another advantage is that fully coupled, fluid-structure interaction problems can be handled in a straightforward manner. A no slip contact algorithm is automatic to the method. That is, it comes at no additional computational expense.

Applications of the MPM are emerging in ice-sheet models for climate simulation Sulsky et al. [2007] as well as more traditional explosive-related simulations **?**. In 2004, numerical noise caused by cell-boundary crossing of material points was identified, and the generalized interpolation material point (GIMP) method was introduced to avoid the noise **?**. When

[1] Advanced MPM formulations do track the subregion shape.

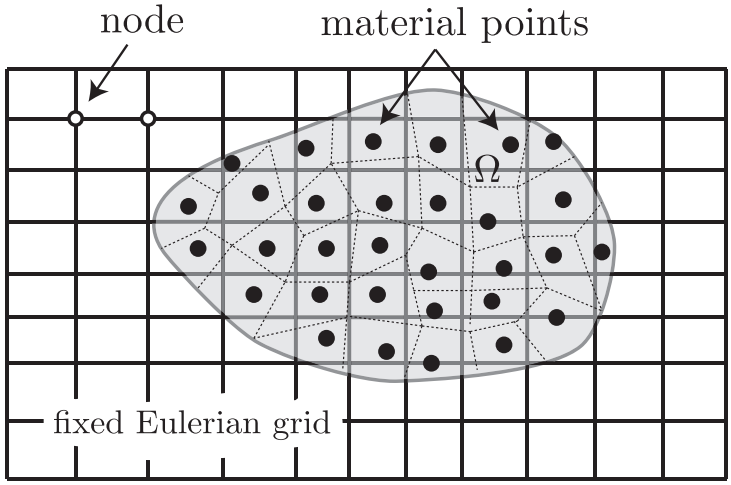

Figure 2: Material point method: Lagrangian material points overlaid over an Eulerian grid. The dotted lines denote the physical domains for each particle. These domains can be found by Voronoi tesselation.

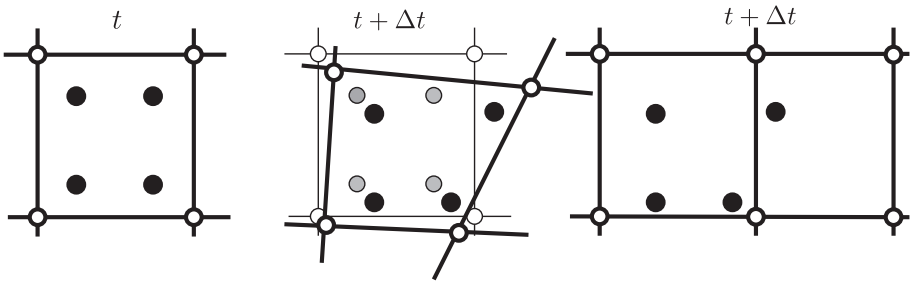

Figure 3: Material point method: Lagrangian material points overlaid over an Eulerian grid. Note that the material points can move to other elements. This is contrast to integration points in FEM which always stay at the same element.

the choice of characteristic function is the Dirac delta function, the traditional MPM formulation is recovered exactly. fluid-membrane interaction York et al. [1999, 2000]. **?** comparative study of SPH and MPM for hypervelocity impact problems. The conclusion was that the material point method is an efficient and promising method for simulating the hypervelocity impact problems. geo-engineering landslide.

3 Governing equations

4 Weak form and discretisation

The MPM also uses the weak formulation as in FEM which is given as

$$\int_\Omega \rho \delta u_i a_i \mathrm{d}\Omega + \int_\Omega \rho \frac{\partial \delta u_i}{\partial x_j} \sigma^s_{ij} \mathrm{d}\Omega = \int_\Omega \rho \delta u_i b_i \mathrm{d}\Omega + \int_{\Gamma_t} \rho \delta u_i \bar{t}_i \mathrm{d}\Gamma \tag{1}$$

where Ω denotes the current configuration, σ^s_{ij} is the specific stresses i.e., $\sigma^s_{ij} = \sigma_{ij}/\rho$.

The whole material domain is described with a set of material points, and it is assumed that the whole mass of a material subdomain is concentrated at the corresponding material point, which means that the mass density field is expressed as

$$\rho(\mathbf{x}, t) = \sum_{p=1}^{n_p} M_p \delta(\mathbf{x} - \mathbf{x}_p) \tag{2}$$

where δ is the Dirac delta function with dimension of the inverse of volume. Substitution of Equation (2) into Equation (1) results in

$$\sum_{p=1}^{n_p} M_p \delta u_i(\mathbf{x}_p) a_i(\mathbf{x}_p) + \sum_{p=1}^{n_p} M_p \left.\frac{\partial \delta u_i}{\partial x_j}\right|_{(\mathbf{x}_p)} \sigma^s_{ij}(\mathbf{x}_p) = \sum_{p=1}^{n_p} M_p \delta u_i(\mathbf{x}_p) b_i(\mathbf{x}_p) + \sum_{p=1}^{n_p} M_p \delta u_i(\mathbf{x}_p) \bar{t}_i(\mathbf{x}_p) \tag{3}$$

At this point, MPM is identical to FEM in which the material points are served as integration points. The background Eulerian grid serves as a finite element mesh with shape functions $\{N_I\}_{I=1}^{n_n}$ where n_n denotes the total number of nodes. Thanks to the use of a grid, evaluation of shape functions and derivatives are standard and does not involve neighbor search as in meshfree methods such as SPH. The position and displacement of particle p are then given by

$$x_i(\mathbf{x}_p) = \sum_{I=1}^{n_n} N_I(\mathbf{x}_p) x_{iI} \tag{4}$$

$$u_i(\mathbf{x}_p) = \sum_{I=1}^{n_n} N_I(\mathbf{x}_p) u_{iI} \tag{5}$$

where x_{iI} is the i component of the position vector of node I. In 3D, one writes $\mathbf{x}_I = (x_{1I}, x_{2I}, x_{3I})$. And u_{iI} is the i component of the displacement vector of node I. Subscript I denotes the value of grid node I, and subscript p denotes the value of particle p.

The velocity and acceleration fields are given by

$$v_i(\mathbf{x}_p) = \sum_{I=1}^{n_n} N_I(\mathbf{x}_p) v_{iI} \tag{6}$$

$$a_i(\mathbf{x}_p) = \sum_{I=1}^{n_n} N_I(\mathbf{x}_p) a_{iI} \tag{7}$$

where v_{iI}, a_{iI} are the i component of the velocity and acceleration vectors of node I.

Using the Bubnov-Galerkin method, the virtual displacement field is approximated as

$$\delta u_i(\mathbf{x}_p) = \sum_{I=1}^{n_n} N_I(\mathbf{x}_p) \delta u_{iI} \tag{8}$$

thus

$$\left.\frac{\partial \delta u_i}{\partial x_j}\right|_{(\mathbf{x}_p)} = \sum_{I=1}^{n_n} \left.\frac{\partial N_I}{\partial x_j}\right|_{(\mathbf{x}_p)} \delta u_{iI} \tag{9}$$

Substituting Equations (8), (7) and (9) into Equation (3) leads to

$$\sum_{p=1}^{n_p} M_p \left[\sum_{I=1}^{n_n} N_I(\mathbf{x}_p)\delta u_{iI}\right]\left[\sum_{J=1}^{n_n} N_J(\mathbf{x}_p) a_{iJ}\right] - \sum_{p=1}^{n_p} M_p \left[\sum_{I=1}^{n_n} \left.\frac{\partial N_I}{\partial x_j}\right|_{(\mathbf{x}_p)} \delta u_{iI}\right] \sigma^s_{ij}(\mathbf{x}_p) = \sum_{p=1}^{n_p} M_p \left[\sum_{I=1}^{n_n} N_I(\mathbf{x}_p)\delta u_{iI}\right] b_i(\mathbf{x}_p) + \sum_{p=1}^{n_p} M_p \left[\sum_{I=1}^{n_n} N_I(\mathbf{x}_p)\delta u_{iI}\right] \bar{t}_i(\mathbf{x}_p) \tag{10}$$

Since δu_{iI} are arbitrary, we obtain the following set of equations (i equations for each node $I, I = 1, \ldots, n_n$)

$$\sum_{p=1}^{n_p} M_p N_I(\mathbf{x}_p)\left(\sum_{J=1}^{n_n} N_J(\mathbf{x}_p) a_{iJ}\right) - \sum_{p=1}^{n_p} M_p \left.\frac{\partial N_I}{\partial x_j}\right|_{(\mathbf{x}_p)} \sigma^s_{ij}(\mathbf{x}_p) = \sum_{p=1}^{n_p} M_p N_I(\mathbf{x}_p) b_i(\mathbf{x}_p) + \sum_{p=1}^{n_p} M_p N_I(\mathbf{x}_p) \bar{t}_i(\mathbf{x}_p) \tag{11}$$

which can be written in the following compact form

$$m_{IJ}\mathbf{a}_J = \mathbf{f}^{\text{ext}}_I + \mathbf{f}^{\text{int}}_I \tag{12}$$

where m_{IJ}, $\mathbf{f}^{\text{ext}}_I$, $\mathbf{f}^{\text{int}}_I$ are the consistent mass matrix, the external force vector and the internal force vector. This equation is exactly identical to FEM.

The consistent mass matrix is given by

$$m_{IJ} = \sum_{p=1}^{n_p} M_p N_I(\mathbf{x}_p) N_J(\mathbf{x}_p) \tag{13}$$

Note that the mass matrix is not constant as in FEM but changes in time because the material points move while the grid nodes are fixed. Therefore, for every time step, an inversion of the mass matrix must be carried out. Thus, explicit integration is usually employed in MPM with the use of a diagonal lumped mass matrix. The diagonal mass matrix is the standard row sum matrix in which the diagonal terms are given by

$$m_I = \sum_{J=1}^{n_n} m_{IJ} = \sum_{J=1}^{n_n} \sum_{p=1}^{n_p} M_p N_I(\mathbf{x}_p) N_J(\mathbf{x}_p) = \sum_{p=1}^{n_p} M_p N_I(\mathbf{x}_p) \tag{14}$$

where Equation (13) and the partition of unity property of FE shape functions were used.

The external force vector is written as

$$\mathbf{f}_I^{\text{ext}} = \sum_{p=1}^{n_p} M_p N_I(\mathbf{x}_p) \mathbf{b}(\mathbf{x}_p) + \sum_{p=1}^{n_p} M_p N_I(\mathbf{x}_p) \bar{\mathbf{t}}(\mathbf{x}_p) \tag{15}$$

and the internal force vector as

$$\mathbf{f}_I^{\text{int}} = -\sum_{p=1}^{n_p} M_p / \rho_p \boldsymbol{\sigma}_p \nabla N_I(\mathbf{x}_p) = -\sum_{p=1}^{n_p} V_p \boldsymbol{\sigma} \nabla N_I(\mathbf{x}_p) \tag{16}$$

where $\nabla N_I = (\frac{\partial N_I}{\partial x_1}, \frac{\partial N_I}{\partial x_2}, \frac{\partial N_I}{\partial x_3})^{\mathrm{T}}$ denotes the gradient of the shape function; V_p is the volume of particle p. Note that particle densities are defined as the ratio of particle mass to particle volume. Note also that $\boldsymbol{\sigma}_p$ is a 3×3 matrix in 3D. This is different from the usual stress vector written in Voigt notation in FEM.

5 Explicit time integration

If a lumped mass matrix is used, Equation (12) becomes

$$m_I^t \mathbf{a}_I^t = \mathbf{f}_I^{\text{ext}} + \mathbf{f}_I^{\text{int}} \equiv \mathbf{f}_I^t \tag{17}$$

and after being multiplied with Δt, one gets

$$m_I^t (\mathbf{v}_I^{t+\Delta t} - \mathbf{v}_I^t) = \mathbf{f}_I^t \Delta t \tag{18}$$

which permits the computation of the updated nodal momenta $(m\mathbf{v})_I^{t+\Delta t}$. Note that essential boundary conditions must be applied before (18). It was shown in Ref. [39] that the explicit procedure of time integration of the dynamic equations required shorter time increment when the MPM is utilized than in the case of using the finite element method. However, before

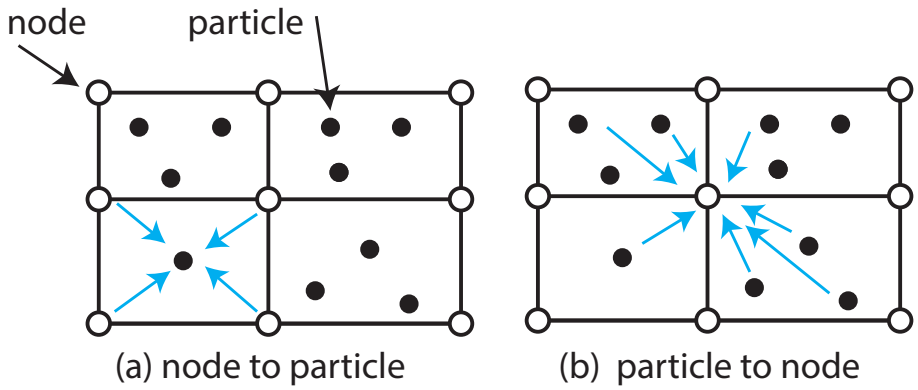

Figure 4: FE mapping in the material point method: (a) nodes to particles mapping and (b) particles to nodes mapping. The former is the standard FE mapping (from nodal displacements to integration points' strains). The latter is something unpopular in FEM although this is also used for FE visualization (mapping the stresses of integration points to the nodes). Note also that the former mapping is local and the latter is non-local (for C^0 shape functions). It should be emphasized that some high order Lagrange basis functions are negative and therefore they are not used in the particles to nodes mapping. High order B-splines are always positive and hence can be used.

solving Equation (17), one needs the nodal masses and momenta which are not stored at the nodes. Therefore, they are mapped from the particles using the shape functions

$$\begin{aligned} m_I^t &= \sum_p N_I(\mathbf{x}_p^t) M_p \\ (m\mathbf{v})_I^t &= \sum_p N_I(\mathbf{x}_p^t)(M\mathbf{v})_p^t \end{aligned} \tag{19}$$

Note that this mapping, graphically illustrated in Fig. (4), is a kind of extrapolation and is a non-local operation.

The particle velocities and positions are updated as follows

$$\mathbf{v}_p^{t+\Delta t} = \mathbf{v}_p^t + \Delta t \sum_I N_I(\mathbf{x}_p)\mathbf{f}_I^t/m_I^t \tag{20}$$

$$\mathbf{x}_p^{t+\Delta t} = \mathbf{x}_p^t + \Delta t \sum_I N_I(\mathbf{x}_p)(m\mathbf{v})_I^{t+\Delta t}/m_I^t \tag{21}$$

which are standard FE interpolations, cf. Fig. (4). Note that this is the momentum formulation in which momentum is used instead of velocity as much as possible, thus avoiding divisions by potentially small nodal masses. Because the velocity field is single-valued, interpenetration of material is precluded. It should be emphasized that this is totally an updated Lagrangian formulation since the grid nodes are moved to new positions $\mathbf{x}_I^{t+\Delta t} = \mathbf{v}_I^t + \Delta t \mathbf{v}_I^{t+\Delta t}$. However

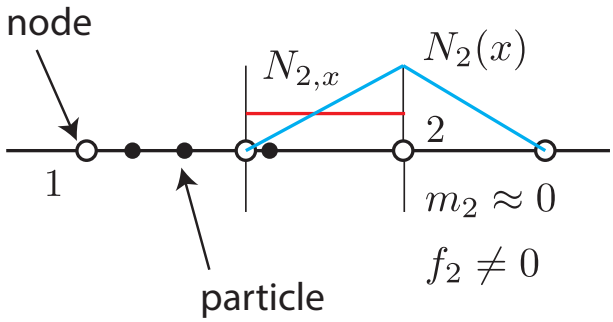

Figure 5: Troubled nodes with nearly zero mass resulting in infinite acceleration (node 2).

the grid nodes are not explicitly moved because it will be discarded anyway at the end of the time step.

Next, one needs to compute the updated particle gradient velocities using the nodal velocities $\mathbf{v}_I^{t+\Delta t}$. However if the velocities were computed as $\mathbf{v}_I^{t+\Delta t} = (m\mathbf{v})_I^{t+\Delta t}/m_I^t$ from Equation (18) the velocities would then be infinite if the mass m_I^t is small. This happens when a particle is very closed to a node having only one particle within its support, Fig. 5. There are at least three ways to treat this issue: (i) using a cutoff value to detect small nodal masses, (ii) a modified USL and (iii) USF. In the first approach, the nodal velocities are computed as

$$\mathbf{v}_I^{t+\Delta t} = \begin{cases} \dfrac{(m\mathbf{v})_I^{t+\Delta t}}{m_I^t} & \text{if } m_I^t > tol \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

which requires an extra parameter (a cutoff value) in the algorithm which is not clear how to choose. Even a good cutoff value can be chosen, it produces an undesirable constraint which should not be in the system. For example it results in non-zero stresses (albeit small) in particles which are moving with the same speed.

The second way Sulsky et al. [1994] is to map the particle velocities back to the nodal velocities using

$$(m\mathbf{v})_I^{t+\Delta t} = \sum_p N_I(\mathbf{x}_p)(M\mathbf{v})_p^{t+\Delta t} \tag{23}$$

and thus

$$\mathbf{v}_I^{t+\Delta t} = \frac{(m\mathbf{v})_I^{t+\Delta t}}{m_I^t} = \frac{\sum_p N_I(\mathbf{x}_p)(M\mathbf{v})_p^{t+\Delta t}}{\sum_p N_I(\mathbf{x}_p)M_p} = \frac{\sum_p N_I(\mathbf{x}_p)(M\mathbf{v})_p^{t+\Delta t}}{m_I^t} \tag{24}$$

The appearance of the shape functions in both numerator and denominator, in the second equality, cancel out its role and the numerical problem is thus cured. Our implementation uses the final equality because m_I^t was computed in Equation (19). Note, however, that this option is inefficient for Equation (23) is a particle-to-node map within the node-to-particle phase.

Next particle velocity gradients are computed

$$\mathbf{L}_p^{t+\Delta t} \equiv \nabla \mathbf{v}_p^{t+\Delta t} = \sum_I \nabla N_I(\mathbf{x}_p)\mathbf{v}_I^{t+\Delta t} \tag{25}$$

where $\mathbf{L}_p$ is a 3×3 matrix of which components are $L_{ij} = v_{i,j}$ (in three dimensions). It is a 2×2 matrix in two dimensions. From the velocity gradients one can compute the gradient deformation using the relation $\dot{\mathbf{F}} = \mathbf{L}\mathbf{F}$. Utilizing a forward Euler method, one can write

$$\frac{\mathbf{F}^{t+\Delta t} - \mathbf{F}^t}{\Delta t} = \mathbf{L}^{t+\Delta t}\mathbf{F}^t \tag{26}$$

And the gradient deformation tensor is thus computed

$$\mathbf{F}_p^{t+\Delta t} = (\mathbf{I} + \mathbf{L}_p^{t+\Delta t}\Delta t)\mathbf{F}_p^t \tag{27}$$

which allows to compute the updated particle volume $V_p^{t+\Delta t} = \det \mathbf{F}_p^{t+\Delta t} V_p^0$. In the above equation, $\mathbf{I}$ is the identity matrix.

Next, the particle stresses are updated. This depends on the constitutive model. For example, one might need to compute the strain increment

$$\Delta \mathbf{e}_p = \left(\text{sym}\mathbf{L}_p^{t+\Delta t}\right) \Delta t \tag{28}$$

and using it to compute the stress increment $\Delta \boldsymbol{\sigma}_p$. The updated particle stresses are given by

$$\boldsymbol{\sigma}_p^{t+\Delta t} = \boldsymbol{\sigma}_p^t + \Delta \boldsymbol{\sigma}_p \tag{29}$$

In Box 5.1 the step-by-step algorithm of an explicit MPM is given. In the literature it is referred to as USL (Update Stress Last). In Bardenhagen [2002], another MPM implementation named USF (Update Stress First) was presented and for completeness, the USF procedure is given in Box 5.2. For explicit dynamics, there is an restriction on the time step $dt < dx/c$, where dx is the grid spacing and $c = \sqrt{E/\rho}$ is the speed of sound in the material.

6 Implementation

In this section, we present a serial implementation of MPM that reuses existing FE resources. Before doing so, it is beneficial to recognize the resemblances and differences between MPM and FEM:

- MPM can be seen as the updated Lagrangian finite element method in which material points serve as integration points and uncoupled from the Eulerian grid. Practically material points, where constitutive equations are applied, move from elements to elements rather than remain at the Gauss points of an element. This requires to track the movement of the material points.

- Contrary to standard FEM where the mesh is fitted to the physical domain of interest, the Eulerian grid in MPM is dimensioned sufficiently large to cover the deformed solid. Therefore, at a certain time step, there are elements with no material points. They are labeled as inactive elements and skipped in the assembly process.

Box 5.1 Solution procedure of an explicit MPM code: USL formulation.

1. Initialisation phase
   (a) Particle distribution in the undeformed configuration
   (b) Grid set up
   (c) Initialise particle quantities such as mass, stress, strain etc.
2. Solution phase for time step t to $t + \Delta t$
   (a) Mapping from particles to nodes
      i. Compute nodal mass $m_I^t = \sum_p N_I(\mathbf{x}_p^t) M_p$
      ii. Compute nodal momentum $(m\mathbf{v})_I^t = \sum_p N_I(\mathbf{x}_p^t)(M\mathbf{v})_p^t$
      iii. Compute external force $\mathbf{f}_I^{\text{ext},t}$
      iv. Compute internal force $\mathbf{f}_I^{\text{int}} = -\sum_{p=1}^{n_p} V_p \boldsymbol{\sigma}_p \nabla N_I(\mathbf{x}_p)$
      v. Compute nodal force $\mathbf{f}_I = \mathbf{f}_I^{\text{ext}} + \mathbf{f}_I^{\text{int}}$
   (b) Update the momenta $(m\mathbf{v})_I^{t+\Delta t} = (m\mathbf{v})_I^t + \mathbf{f}_I \Delta t$
   (c) Mapping from nodes to particles
      i. Update particle velocities $\mathbf{v}_p^{t+\Delta t} = \mathbf{v}_p^t + \Delta t \sum_I N_I(\mathbf{x}_p)\mathbf{f}_I^t / m_I^t$
      ii. Update particle positions $\mathbf{x}_p^{t+\Delta t} = \mathbf{x}_p^t + \Delta t \sum_I N_I(\mathbf{x}_p)(m\mathbf{v})_I^{t+\Delta t} / m_I^t$
      iii. Update nodal velocities $\mathbf{v}_I^{t+\Delta t} = (m\mathbf{v})_I^{t+\Delta t} / m_I^t$
      iv. Update gradient velocity $\mathbf{L}_p^{t+\Delta t} = \sum_I \nabla N_I(\mathbf{x}_p)\mathbf{v}_I^{t+\Delta t}$
      v. Updated gradient deformation tensor $\mathbf{F}_p^{t+\Delta t} = (\mathbf{I} + \mathbf{L}_p^{t+\Delta t}\Delta t)\mathbf{F}_p^t$
      vi. Update volume $V_p^{t+\Delta t} = \det \mathbf{F}_p^{t+\Delta t} V_p^0$.
      vii. Update stresses $\boldsymbol{\sigma}_p^{t+\Delta t} = \boldsymbol{\sigma}_p^t + \Delta\boldsymbol{\sigma}_p$
3. Reset the grid (if it was updated) and advance to the next time step.

Box 5.2 Solution procedure of an explicit MPM code: USF formulation.

1. Initialisation phase
2. Solution phase for time step t to $t + \Delta t$
   (a) Mapping from particles to nodes
      i. Compute nodal mass $m_I^t = \sum_p N_I(\mathbf{x}_p^t) M_p$
      ii. Compute nodal momentum $(m\mathbf{v})_I^t = \sum_p N_I(\mathbf{x}_p^t)(M\mathbf{v})_p^t$
      iii. Compute nodal velocities $\mathbf{v}_I^t = (m\mathbf{v})_I^t / m_I^t$
      iv. Compute gradient velocity $\mathbf{L}_p^t = \sum_I \nabla N_I(\mathbf{x}_p)\mathbf{v}_I^t$
      v. Compute gradient deformation tensor $\mathbf{F}_p^t = (\mathbf{I} + \mathbf{L}_p^t \Delta t)\mathbf{F}_p^t$
      vi. Update volume $V_p^t = \det \mathbf{F}_p^{t+\Delta t} V_p^0$
      vii. Update stresses $\boldsymbol{\sigma}_p^t = \boldsymbol{\sigma}_p^t + \Delta\boldsymbol{\sigma}_p$
      viii. Compute external force $\mathbf{f}_I^{\text{ext},t}$
      ix. Compute internal force $\mathbf{f}_I^{\text{int}} = -\sum_{p=1}^{n_p} V_p \boldsymbol{\sigma}_p \nabla N_I(\mathbf{x}_p)$
      x. Compute nodal force $\mathbf{f}_I = \mathbf{f}_I^{\text{ext}} + \mathbf{f}_I^{\text{int}}$
   (b) Update the momenta $(m\mathbf{v})_I^{t+\Delta t} = (m\mathbf{v})_I^t + \mathbf{f}_I \Delta t$
   (c) Mapping from nodes to particles
      i. Update particle velocities $\mathbf{v}_p^{t+\Delta t} = \mathbf{v}_p^t + \Delta t \sum_I N_I(\mathbf{x}_p)\mathbf{f}_I^t / m_I^t$
      ii. Update particle positions $\mathbf{x}_p^{t+\Delta t} = \mathbf{x}_p^t + \Delta t \sum_I N_I(\mathbf{x}_p)(m\mathbf{v})_I^{t+\Delta t} / m_I^t$
3. Reset the grid (if it was updated) and advance to the next time step.

- Unlike FEM, in MPM there is a mapping from Gauss points (particles) to nodes.

Some advantages of MPM are listed as follows

- Mesh distortion of Lagrangian FEM is eliminated.
- The problem of free surface is easy to solve.
- The problem of no-slip self-contact is solved automatically in the case of granular material: there is no penetration of particles because the velocities of the material point are single valued.
- Boundary conditions can be applied as easily as in the case of standard FEM.
- Adding material points during calculations is relatively easy as there is no coupling between them and the grid.
- The MPM can be implemented in a FEM program in a relatively easy way in constrast to meshfree methods or ALE methods.
- Solving fluid-structure interaction is relatively straightforward: the background grid is also used as an Eulerian grid for the fluid.

The MPM suffers from the following disadvantages

- Efficiency of MPM is lower than that of FEM due to the mappings between the background grid and the particles and the accuracy of particles quadrature used in MPM is lower than that of Gauss quadrature used in FEM.
- This method, in analogy with meshless methods, also introduces new visualization challenges.
- Whereas the finite element method (FEM) has long-established basic verification standards (patch tests, convergence testing, etc.), no such standards have been universally adopted within the particle method community.

Available MPM codes

- Uintah, http://www.uintah.utah.edu
- MPM-GIMP, http://sourceforge.net/p/mpmgimp/home/Home/
- Mechsys, http://mechsys.nongnu.org/index.shtml
- NairnMPM, http://osupdocs.forestry.oregonstate.edu/index.php/Main_Pag
- CartaBlanca

All are written in C++ except the last one which is written in Java.

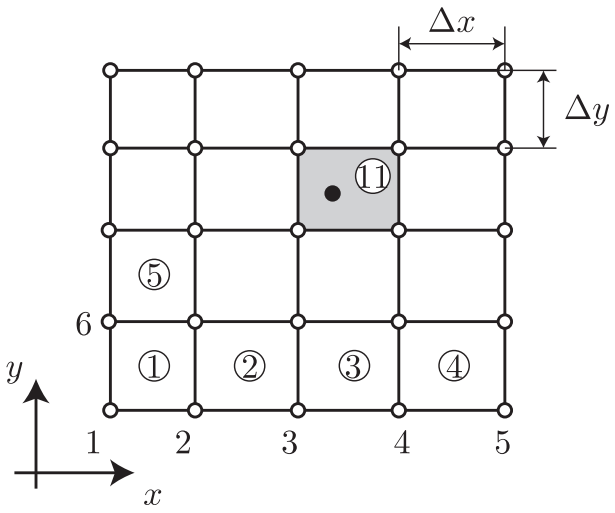

Figure 6: A two dimensional structured grid.

6.1 Eulerian grid

In MPM a structured Eulerian grid is usually used for its advantages. Among other benefits, a uniform Cartesian grid eliminates the need for computationally expensive neighborhood searches during particle-mesh interaction. A structured mesh is illustrated in Fig. 6 for 2D cases. For a particle p with coordinates (x_p, y_p), it is straightforward to track which element it belongs to using the following equation

$$e = [\text{floor}((x_p - x_{\text{min}})/\Delta_x) + 1] + nel_x[\text{floor}((y_p - y_{\text{min}})/\Delta y)] \tag{30}$$

where nel_x denotes the number of elements along the x direction and $(x_{\text{min}}, y_{\text{min}})$ are the minimum node coordinates and $\Delta x, \Delta y$ are the nodal spacing in the x and y directions, respectively.

In order to keep the implementation similar to FEM as much as possible, at every time step, the elements must know which particles they are storing. Listing 1 gives a simple data structure implemented in Matlab for this purpose.

Listing 1: Matlab data structures for mesh-particle interaction

```
pElems  = ones(pCount,1);
mpoints = cell(elemCount,1);

for p=1:pCount
    x = xp(p,1);
    y = xp(p,2);
    e = floor(x/deltax) + 1 + numx2*floor(y/deltay);
    pElems(p) = e; % particle "p" stays in element "e"
end

for e=1:elemCount
    id  = find(pElems==e);
    mpoints{e}=id; % mpoints{e}-> indices of particles in "e"
end
```

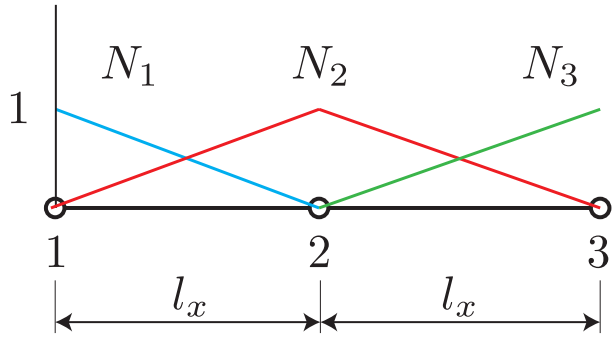

Figure 7: Material point method: One dimensional shape functions defined in global coordinate system.

6.2 Shape functions

Although any grid can be used in MPM, a Cartesian grid is usually chosen for computational convenience reasons. In order to avoid finding the natural coordinates of material points if shape functions are defined in the parameter space, in MPM shape functions are conveniently defined in the global coordinate system. In 1D, the shape functions are defined as

$$N_I^x(x) = \begin{cases} 1 - |x - x_I|/l_x & \text{if } |x - x_I| \leq l_x \\ 0 & \text{else} \end{cases} \tag{31}$$

where l_x denotes the nodal spacing or element size in the x direction. For 2D, the shape functions are simply the tensor-product of the two shape functions along the x and y directions

$$N_I(x, y) = N_I^x(x) N^y(y) \tag{32}$$

Remark 6.1. If shape functions are defined in the parent domain as most isoparametric elements are i.e., $N_I = N_I(\xi, \eta)$, then one has to perform one extra step–after getting the updated particle positions, determine the natural coordinates (ξ, η) of them. For a 2D structured grid with linear elements, it is straightforward to do so as

$$\begin{aligned} \xi &= \frac{2x - (x_1^e + x_2^e)}{x_2^e - x_1^e} \\ \eta &= \frac{2y - (y_1^e + y_2^e)}{y_2^e - y_1^e} \end{aligned} \tag{33}$$

where (x_1^e, y_1^e) and (x_2^e, y_2^e) denote the lower-left and upper-right nodes of element e, respectively. Extension to 3D is straightforward.

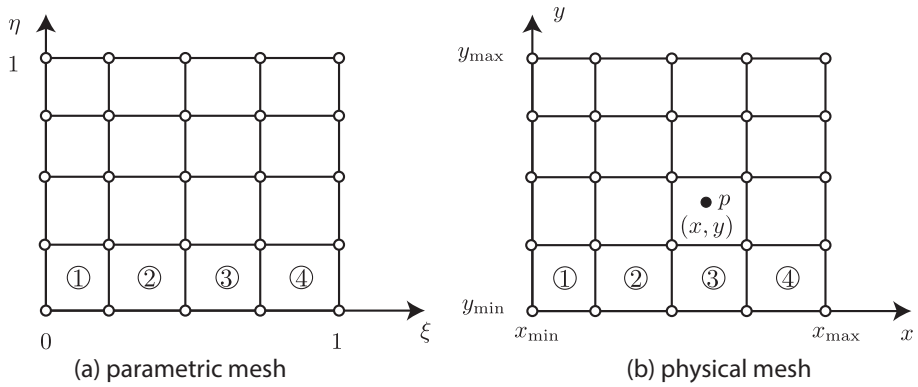

Figure 8: A two dimensional structured grid with B-splines.

6.2.1 B-splines basis functions

Steffen et al. [2008] showed that for simple problems, the use of cubic splines improves the spatial convergence properties of method as grid-crossing errors are reduced.

Given a knot vector $\Xi^1 = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$, the associated set of B-spline basis functions $\{N_{i,p}\}_{i=1}^{n}$ are defined recursively by the Cox-de-Boor formula, starting with the zeroth order basis function ($p = 0$)

$$N_{i,0}(\xi) = \begin{cases} 1 & \text{if } \xi_i \leq \xi < \xi_{i+1}, \\ 0 & \text{otherwise,} \end{cases} \tag{34}$$

and for a polynomial order $p \geq 1$

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi). \tag{35}$$

in which fractions of the form $0/0$ are defined as zero.

In two dimensions, the knot vectors are $\Xi^1 = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$ and $\Xi^2 = \{\eta_1, \eta_2, \ldots, \eta_{m+q+1}\}$ and the bi-variate B-spline basis functions are defined as

$$N_{i,j}(\xi, \eta) = N_{i,p}(\xi) M_{j,q}(\eta) \tag{36}$$

The mapping between the parameter space and the physical space is always linear regardless of the basis order and is given by

$$x = (x_{max} - x_{min})\xi + x_{min}, \quad y = (y_{max} - y_{min})\eta + y_{min} \tag{37}$$

High order B-spline basis functions are C^{p-1} not C^0, the connectivity of elements is different from standard finite elements. Fig. 9 illustrates this difference in case of cubic B-splines. Apart from this B-splines MPM is exactly identical to standard MPM.

6.3 Initial particle distribution

There are numerous ways to obtain the initial particle distribution depends on the geometry of the object and/or the available tools. For example, in order to generate particles for a circular

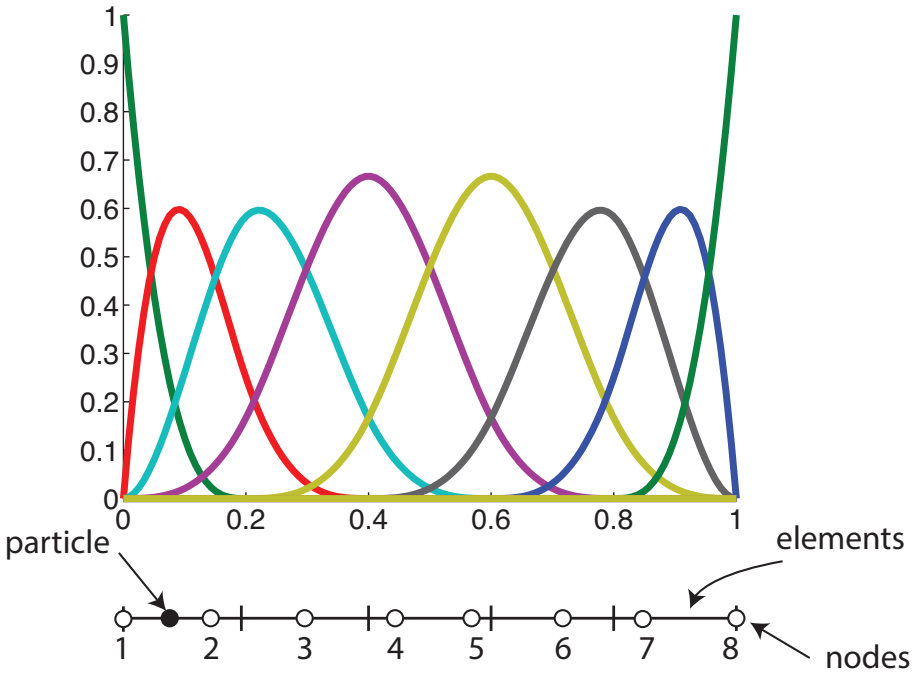

Figure 9: One dimensional cubic ($p = 3$) B-spline basis functions on a open uniform knot $\Xi = \{0, 0, 0, 0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1, 1\}$. There are 8 nodes (control points in CAD terminology) and 5 elements (or knot spans in CAD). At can be seen from the figure, at any point there are 4 ($= (p + 1)$) non-zero basis functions. Therefore each element has 4 nodes. The firstelement's connectivity is $[1, 2, 3, 4]$.

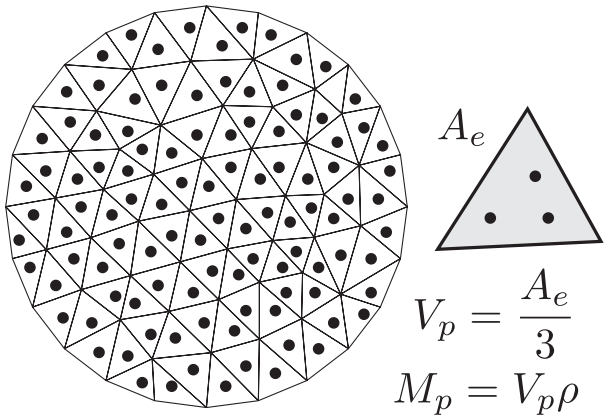

Figure 10: Initial particle distribution: obtained using the available FE mesh generators.

disk, one can use a mesh generator to partition the disk into a set of triangles. The particles can be taken as the centers of these triangles, Fig. 10. Alternatively, integration points of the triangular elements are mapped to the global coordinate systems and then are used as material points. The area of each triangle can be easily obtained and thus the particle volumes and masses can be determined.

6.4 Visualisation

In FEM visualisation is typically performed on the mesh. Information stored at the integration points are extrapolated to the nodes to this end. In MPM, the computational grid is fixed and thus not suitable for visualisation. There are at least two approaches to visualizing MPM results

- Particle visualization: the particles are visualized directly as spheres. Particle data (position, stresses etc.) are written to files (say VTK files) and can be processed by Paraview or Visit.
- Particle mesh visualization: in standard MPM, one can generate a mesh from the particle positions using a Delaunay triangulation and use this mesh for visualization. Note however that this method is expensive since the particles positions are evolving in time.

7 Examples

In this section various examples in one and two dimensions are provided to verify the implementation as well as to demonstrate the performance of the MPM.

7.1 1D examples

As the simplest MPM example, let us consider the vibration of a single material point as shown in Fig. 11. The bar is represented by a single point initially located at $X_p = L/2$, which has an initial velocity v_0. The material is linear elastic.

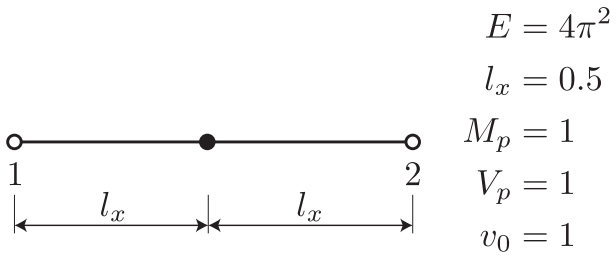

Figure 11: Vibration of a single material point (solid point).

The exact solution is given by

$$v(t) = v_0 \cos(\omega t), \quad \omega = \frac{1}{L}\sqrt{E/\rho} \tag{38}$$

for the velocity and

$$x(t) = x_0 \exp\left[\frac{v_0}{Lw}\sin(\omega t)\right] \tag{39}$$

for the position. The density ρ is constant and equals one. The constitutive equation is $\dot{\sigma} = E\dot{\epsilon}$, $\dot{\epsilon} = dv/dx$ where E is the Young's modulus. The grid consists of one two-noded element. The elastic wave speed is $c = \sqrt{E/\rho} = 2\pi$. Boundary conditions are imposed on the grid and demand that both the node velocity and the acceleration at $x = 0$ (the left node) be zero throughout the simulation. The Matlab implementation is given in Listing 2. Comparison between MPM and the exact solutions are given in Fig. 12. Kinetic, strain and total energies are plotted in Fig. 13 from which one observes that energy is conserved. The strain and kinetic energy are computed as

$$U = \frac{1}{2}\sigma_p \epsilon_p V_p, \quad K = \frac{1}{2} v_p^2 M_p \tag{40}$$

Listing 2: Matlab implementation

```
% Computational grid
nodes     = [0 L];
elements  = [1 2];
elemCount = size(elements,1);
nodeCount = size(nodes,2);
% Material points
xp = 0.5*L;              % position
Mp = 1;                  % mass
Vp = 1;                  % volume
vp = 0.1;                % velocity
s  = 0.;                 % stress
q  = Mp*vp;              % momentum
% Time loop
time  = 0.1;
dtime = 0.01;
t     = 0;
```

```
ta = []; va = []; xa = [];
while ( t < time )
    % shape functions and derivatives
    N1  = 1 - abs(xp-nodes(1))/L;
    N2  = 1 - abs(xp-nodes(2))/L;
    dN1 = -1/L;
    dN2 =  1/L;
    % particle mass and momentum to node
    m1   = N1*Mp;
    m2   = N2*Mp;
    mv1  = N1*q;
    mv2  = N2*q;

    mv1  = 0; % Boundary condition v=0 at node 1
    % internal force
    fint1 = - Vp*s*dN1;
    fint2 = - Vp*s*dN2;

    f1    = fint1 ;   % there is no external force
    f2    = fint2 ;
    % update nodal momenta
    f1  = 0; % Boundary conditions f1 = m1*a1, a1=0

    mv1 = mv1 + f1*dtime;
    mv2 = mv2 + f2*dtime;

    % update particle velocity and position
    vp  = vp + dtime * (N1*f1/m1 + N2*f2/m2);
    xp  = xp + dtime * (N1*mv1/m1 + N2*mv2/m2);

    q   = Mp*vp;                    % momentum

    v1 = N1*Mp*vp/m1;
    v2 = N2*Mp*vp/m2;
    v1 = 0;                         % boundary condition
    Lp = dN1 * v1 + dN2 * v2;       % gradient velocity
    dEps = dtime * Lp;              % strain increment
    s    = s + E * dEps;            % stress update
    % store time,velocty for plotting
    ta = [ta;t];
    va = [va;vp];
    xa = [xa;xp];
    % advance to the next time step
    t = t + dtime;
end
```

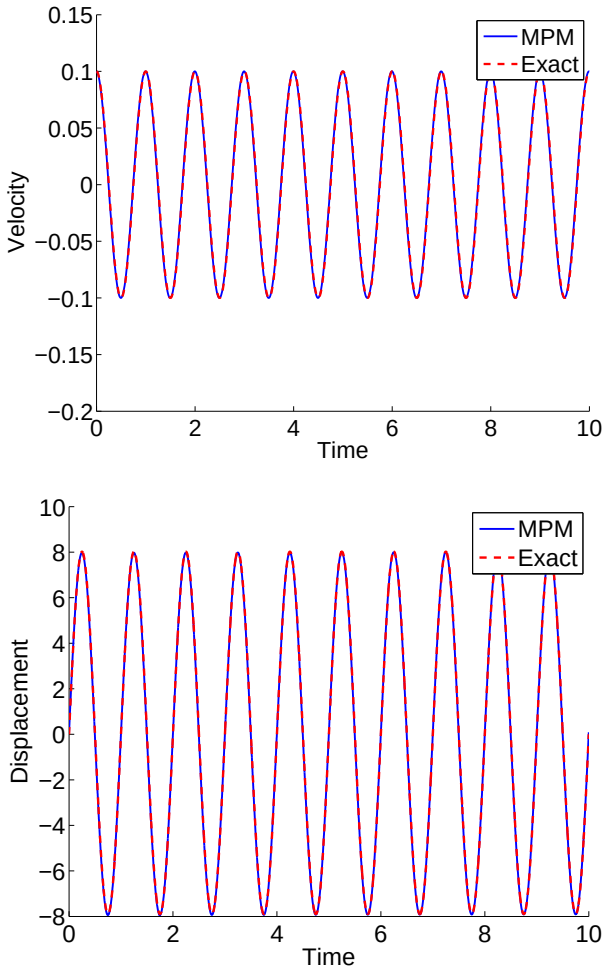

Figure 12: Vibration of a single material point.

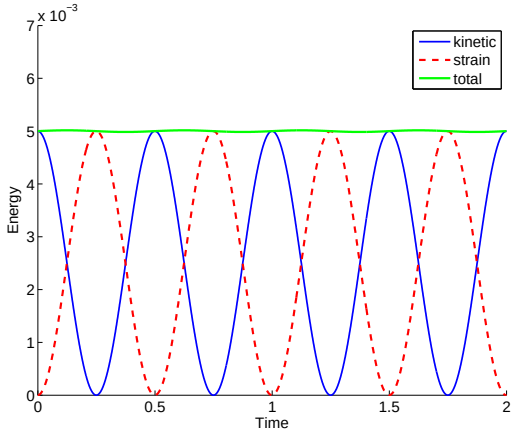

Figure 13: Vibration of a single material point: kinetic, strain and total energies.

7.2 Axial vibration of a continuum bar

In this example, we study the axial vibration of a continuum bar with Young's modulus $E = 100$ and the bar length $L = 25$.

Exact solutions for mode 1 are

$$v(x,t) = v_0 \cos(\omega_1 t) \sin(\beta_1 x) \tag{41}$$

$$u(x,t) = \frac{v_0}{\omega_1} \sin(\omega_1 t) \sin(\beta_1 x) \tag{42}$$

where $\omega_1 = \frac{\pi}{2L}\sqrt{E/\rho}$ and $\beta_1 = \frac{\pi}{2L}$.

The initial velocity is given by

$$v(x,0) = v_0 \sin(\beta_1 x) \tag{43}$$

The grid consists of 13 two-noded line elements (14 grid nodes) and 13 material points placed at the center of the elements are used. The initialisation step implemented in Matlab is given in Listing 3. The solution step implemented in Matlab is given in Listing 4. The results are given in Fig. 14 and the evolution of kinetic energy, strain energy and their sum (the total energy) is plotted in Fig. 15. The time increment is chosen as $\Delta t = 0.1\Delta x/c$ where Δx denotes the nodal spacing.

Listing 3: Vibration of a continuum bar: initialisation.

```
%  Computational grid: two-noded elements
elemCount = 13;
nodes     = linspace(0,L,elemCount+1);
elements  = zeros(elemCount,2);
for ie=1:elemCount
    elements(ie,:) = ie:ie+1;
end
nodeCount = size(nodes,2);

velo = zeros(nodeCount,1);
mId  = floor((elemCount)/2)+1; % id of the center mass particle
% Material points: centers of the elements  one particle per element
c      = sqrt(E/rho);
beta1  = pi/2/L;
omega1 = beta1*c;
deltax = L/elemCount; % nodal spacing
% material points at the center of elements
xp = zeros(elemCount,1);
for p=1:elemCount-1
    xp(p) = 0.5*(nodes(p) + nodes(p+1));
end
pCount = length(xp);

Mp  = deltax*ones(pCount,1);            % mass
Vp  = deltax*ones(pCount,1);            % volume
```

```
Fp   = ones(pCount,1);                    % gradient deformation
Vp0  = Vp;                                % initial volume
sp   = zeros(pCount,1);                   % stress
vp   = zeros(pCount,1);                   % velocity
% initial velocities
for p=1:pCount
    vp(p) = 0.1*sin(beta1*xp(p));
end
```

Listing 4: Vibration of a continuum bar: processing step.

```
dtime = 0.1*deltax/c;
time  = 100;
t     = 0;

ta = []; va = []; xa = [];

nmass     = zeros(nodeCount,1);  % nodal mass vector
nmomentum = zeros(nodeCount,1);  % nodal momentum vector
niforce   = zeros(nodeCount,1);  % nodal internal force vector
neforce   = zeros(nodeCount,1);  % nodal external force vector

while ( t < time )
    nmass(:)     = 0;
    nmomentum(:) = 0;
    niforce(:)   = 0;
    % loop over computational cells or elements
    for e=1:elemCount
        esctr = elements(e,:);
        enode = nodes(esctr);
        Le    = enode(2)-enode(1);
        mpts  = mpoints{e};
        % loop over particles
        for p=1:length(mpts)
            pid  = mpts(p);
            N1   = 1 - abs(xp(pid)-enode(1))/Le;
            N2   = 1 - abs(xp(pid)-enode(2))/Le;
            dN1 = -1/Le;
            dN2 =  1/Le;
            % particle mass and momentum to node
            nmass(esctr(1))      += N1*Mp(pid);
            nmass(esctr(2))      += N2*Mp(pid);
            nmomentum(esctr(1))  += N1*Mp(pid)*vp(pid);
            nmomentum(esctr(2))  += N2*Mp(pid)*vp(pid);
            % internal force
            niforce(esctr(1)) = niforce(esctr(1)) - Vp(pid)*sp(pid)*dN1;
            niforce(esctr(2)) = niforce(esctr(2)) - Vp(pid)*sp(pid)*dN2;
        end
    end
    % update nodal momenta
    nmomentum(1)  = 0; % Boundary conditions f1 = m1*a1, a1=0
    niforce(1)    = 0;
```

```
    for i=1:nodeCount
        nmomentum(i) = nmomentum(i) + niforce(i)*dtime;
    end
    % update particle velocity and position and stresses
    for e=1:elemCount
        esctr = elements(e,:);
        enode = nodes(esctr);
        Le    = enode(2)-enode(1);
        mpts  = mpoints{e};
        % loop over particles
        for p=1:length(mpts)
            pid  = mpts(p);
            N1   = 1 - abs(xp(pid)-enode(1))/Le;
            N2   = 1 - abs(xp(pid)-enode(2))/Le;
            dN1 = -1/Le; dN2 =  1/Le;
            if nmass(esctr(1)) > tol
               vp(pid)  += dtime * N1*niforce(esctr(1))/nmass(esctr(1));
            end
            if nmass(esctr(2)) > tol
               vp(pid)  +=  dtime * N2*niforce(esctr(2))/nmass(esctr(2));
            end
            xp(pid)  += dtime * (N1*nmomentum(esctr(1))/nmass(esctr(1)) +
                                 N2*nmomentum(esctr(2))/nmass(esctr(2)));
            v1 = nmomentum(esctr(1))/nmass(esctr(1));
            v2 = nmomentum(esctr(2))/nmass(esctr(2));
            %if ( esctr(1) == 1 ) v1 = 0; end
            Lp      = dN1 * v1 + dN2 * v2;          % gradient velocity
            Fp(pid) = (1 + Lp*dtime)*Fp(pid);       % gradient deformation
            Vp(pid) = Fp(pid)*Vp0(pid);             % volume
            dEps    = dtime * Lp;                   % strain increment
            sp(pid) = sp(pid) + E * dEps;           % stress update
        end
    end
    % store time,velocty for plotting
    ta = [ta;t]; va = [va;vp(mId)]; xa = [xa;xp(mId)];
    % advance to the next time step
    t = t + dtime;
end
```

7.3 Impact of two elastic bodies

Fig. 16 shows the problem of the impact of two identical elastic disks which move in opposite direction towards each other Sulsky et al. [1994]. The computational domain is a square which is discretised into 20×20 Q4 elements. A plane strain condition is assumed. The mesh and the initial particle distribution are shown in Fig. 17. There are 320 particles for two disks which are obtained by meshing (using Gmsh) the two disks and take the centers as the initial particle positions. Initial condition for this problem is the initial velocities of the particles, $\mathbf{v}_p = \mathbf{v}$ for lower-left particles and $\mathbf{v}_p = -\mathbf{v}$ for upper-right particles. There is no boundary conditions in this problem since the simulation stops before the particles after impact move out of the

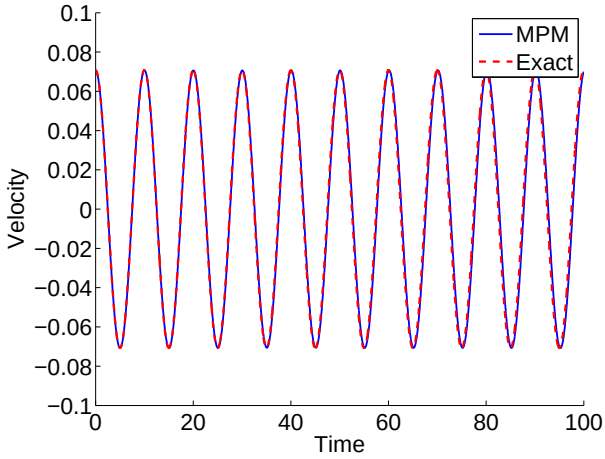

Figure 14: Vibration of a continuum bar: comparison of the MPM velocity solution and the exact solution.

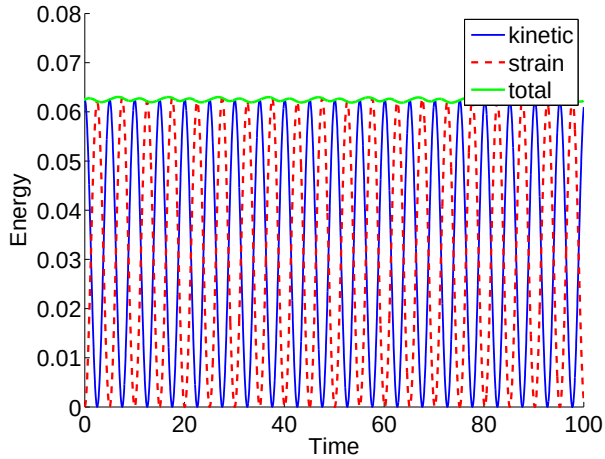

Figure 15: Vibration of a continuum bar: kinetic, strain and total energies.

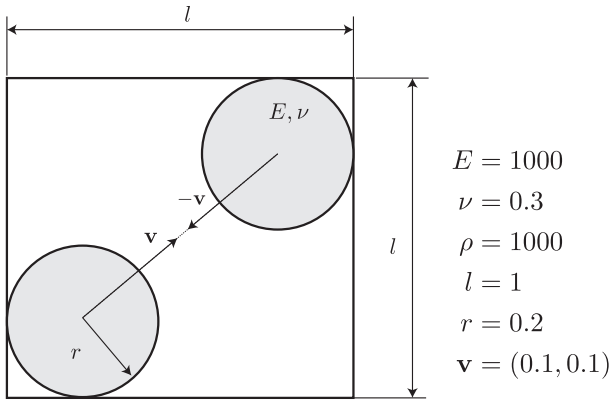

Figure 16: Impact of two elastic bodies: problem statement.

computational box.

In order to check the energy conservation, the strain and kinetic energy are computed for each time step. They are defined as

$$U = \sum_{p=1}^{n_p} u_p V_p, \quad K = \frac{1}{2} \sum_{p=1}^{n_p} \mathbf{v}_p \cdot \mathbf{v}_p M_p \tag{44}$$

where u_p denotes the strain energy density of particle p, $u_p = 1/2\sigma_{p,ij}\epsilon_{p,ij}$ or explicitly as

$$u_p = \frac{1}{4\mu}\left[\frac{\kappa+1}{4}(\sigma_{p,xx}^2 + \sigma_{p,yy}^2) - 2(\sigma_{p,xx}\sigma_{p,yy} - \sigma_{p,xy}^2)\right] \tag{45}$$

with μ and κ are the shear modulus and the Kolosov constant, respectively.

List 5 and 6 gives the code for initialising the particle data and nodal data. Lists 7, 8 and 9 presents the codes for the solution phase. Note that some C++ notations (+=) were adopted to save space. It should be emphasized that the code has not been optimized to make it easy to understand. One can, for example, store the shape functions and its derivatives computed in List 8 so that they can be reused in List 9.

Listing 5: Two dimensional MPM: particle initialisation.

```
pCount = numelem;                          % # of particles
Mp   = ones(pCount,1);                     % mass
Vp   = ones(pCount,1);                     % volume
Fp   = ones(pCount,4);                     % gradient deformation
s    = zeros(pCount,3);                    % stress
eps  = zeros(pCount,3);                    % strain
vp   = zeros(pCount,2);                    % velocity
xp   = zeros(pCount,2);                    % position
% initialise particle position, mass ,volume, velocity
for e = 1:numelem
    coord = node1(element1(e,:),:);
    a     = det([coord,[1;1;1]])/2;
    Vp(e) = a;
```

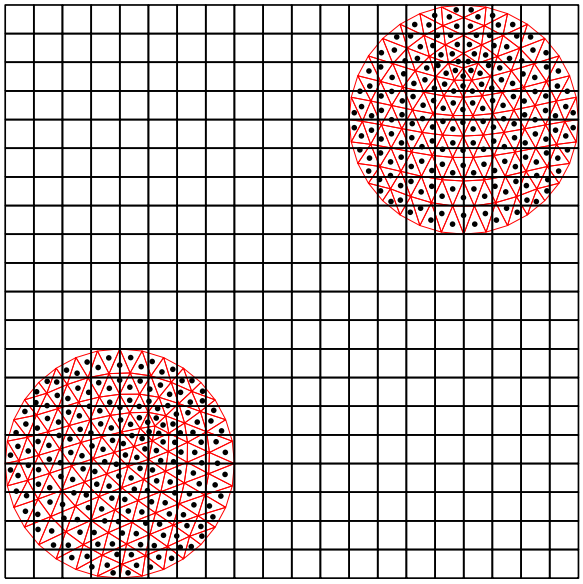

Figure 17: Impact of two elastic bodies: Eulerian mesh and initial particle distribution.

```
    Mp(e) = a*rho;
    xp(e,:) = mean(coord);
    % this is particular for the two impact disks example
    if xp(e,1) < 0.5
        vp(e,:) = [v v];
    else
        vp(e,:) = [-v -v];
    end
    Fp(e,:) = [1 0 0 1];
end
Vp0 = Vp;
```

Listing 6: Two dimensional MPM: nodal initialisation.

```
% build the structured grid ...
% initialise nodal data
nmass     = zeros(nodeCount,1);  % nodal mass vector
nmomentum = zeros(nodeCount,2);  % nodal momentum vector
niforce   = zeros(nodeCount,2);  % nodal internal force vector
neforce   = zeros(nodeCount,2);  % nodal external force vector
```

Listing 7: Two dimensional MPM: solution phase (explicit).

```
while ( t < time )
    % reset grid data
    nmass(:)     = 0;
    nmomentum(:) = 0;
    niforce(:)   = 0;
    % particle to grid nodes
```

```
    See List 8
    % update nodal momenta
    nmomentum = nmomentum + niforce*dtime;
    % nodes to particles
    See List 9
    % store time,velocty for plotting
    pos{istep} = xp;
    vel{istep} = vp;
    % update the element particle list
    for p=1:pCount
        x = xp(p,1);
        y = xp(p,2);
        e = floor(x/deltaX) + 1 + numx2*floor(y/deltaY);
        pElems(p) = e;
    end
    for e=1:elemCount
        id  = find(pElems==e);
        mpoints{e}=id;
    end
    % VTK output
    if ( mod(istep,interval) == 0 )
        xp = pos{istep};
        vtkFile = sprintf('../results/%s%d',vtkFileName,istep);
        VTKParticles(xp,vtkFile,s);
    end
    % advance to the next time step
    t     = t + dtime; istep = istep + 1;
end
```

Listing 8: Two dimensional MPM: particles to nodes.

```
for e=1:elemCount                % loop over elements
   esctr = element(e,:);         % element connectivity
   enode = node(esctr,:);        % element node coords
   mpts  = mpoints{e};           % particles inside element e
  % loop over particles
   for p=1:length(mpts)
       pid  = mpts(p);           % particle ID
       pt(1)= (2*xp(pid,1)-(enode(1,1)+enode(2,1)))/deltaX;
       pt(2)= (2*xp(pid,2)-(enode(2,2)+enode(3,2)))/deltaY;

       [N,dNdxi]=lagrange_basis('Q4',pt);   % element shape functions
       J0       = enode'*dNdxi;             % element Jacobian matrix
       invJ0    = inv(J0);
       dNdx     = dNdxi*invJ0;
      % particle mass and momentum to node
       stress = s(pid,:);
       for i=1:length(esctr)
           id    = esctr(i);   % node ID
           dNIdx = dNdx(i,1);
           dNIdy = dNdx(i,2);
           nmass(id)       += N(i)*Mp(pid);
```

```
            nmomentum(id,:) += N(i)*Mp(pid)*vp(pid,:);
            niforce(id,1)   -= Vp(pid)*(stress(1)*dNIdx + stress(3)*dNIdy);
            niforce(id,2)   -= Vp(pid)*(stress(3)*dNIdx + stress(2)*dNIdy);
        end
    end
end
```

Listing 9: Two dimensional MPM: nodes to particles.

```
for e=1:elemCount                % loop over elements
        esctr = element(e,:);
        enode = node(esctr,:);
        mpts  = mpoints{e};
        % loop over particles
        for p=1:length(mpts)
            pid  = mpts(p);
            pt(1)= (2*xp(pid,1)-(enode(1,1)+enode(2,1)))/deltaX;
            pt(2)= (2*xp(pid,2)-(enode(2,2)+enode(3,2)))/deltaY;
            [N,dNdxi]=lagrange_basis('Q4',pt);
            J0       = enode'*dNdxi;
            invJ0    = inv(J0);
            dNdx     = dNdxi*invJ0;
            Lp = zeros(2,2);
            for i=1:length(esctr)
                id = esctr(i);       % node ID
                vI = [0 0];
                if nmass(id) > tol
                    vp(pid,:) += dtime * N(i)*niforce(id,:)  /nmass(id);
                    xp(pid,:) += dtime * N(i)*nmomentum(id,:)/nmass(id);
                    vI        = nmomentum(id,:)/nmass(id);% nodal velocity
                end
                Lp = Lp + vI'*dNdx(i,:); % particle gradient velocity
            end

            F       = ([1 0;0 1] + Lp*dtime)*reshape(Fp(pid,:),2,2);
            Fp(pid,:)= reshape(F,1,4);
            Vp(pid) = det(F)*Vp0(pid);
            dEps    = dtime * 0.5 * (Lp+Lp');
            dsigma  = C * [dEps(1,1);dEps(2,2);2*dEps(1,2)] ;
            s(pid,:)  = s(pid,:) + dsigma';
            eps(pid,:)= eps(pid,:) + [dEps(1,1) dEps(2,2) 2*dEps(1,2)];
        end
end
```

The movement of two disks is given in Fig. 18. The collision occurs in a physically realistic fashion, although no contact law has been specified. Fig. 19 plots the evolution of the kinetic, strain and total energy.

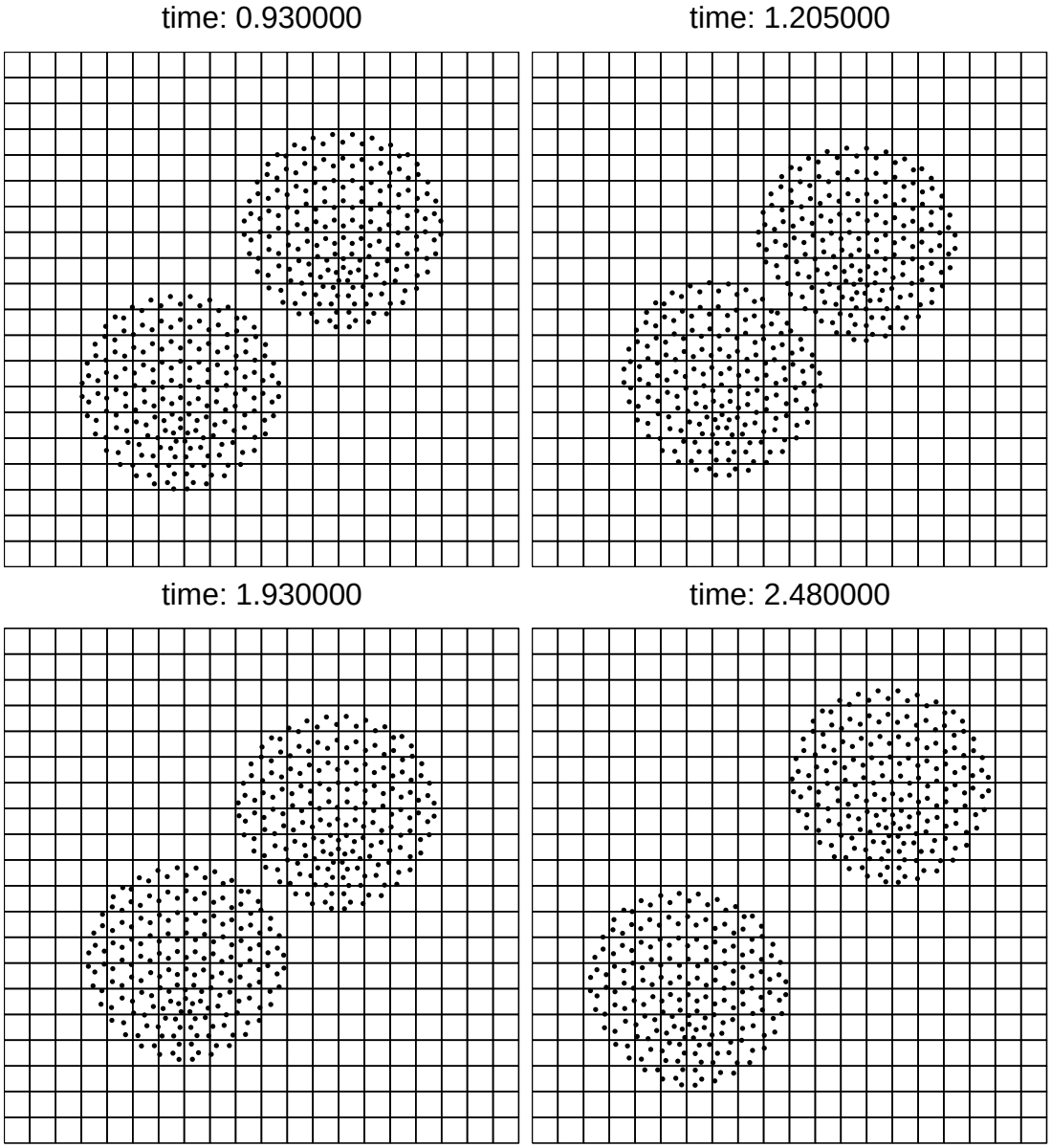

Figure 18: Impact of two elastic bodies: time snapshots of the bodies. Top figures: twp bodies move towards each other and collide. Bottom figures: they bounce back and move far from each other. These figures were created in Matlab using the *scatter* command.

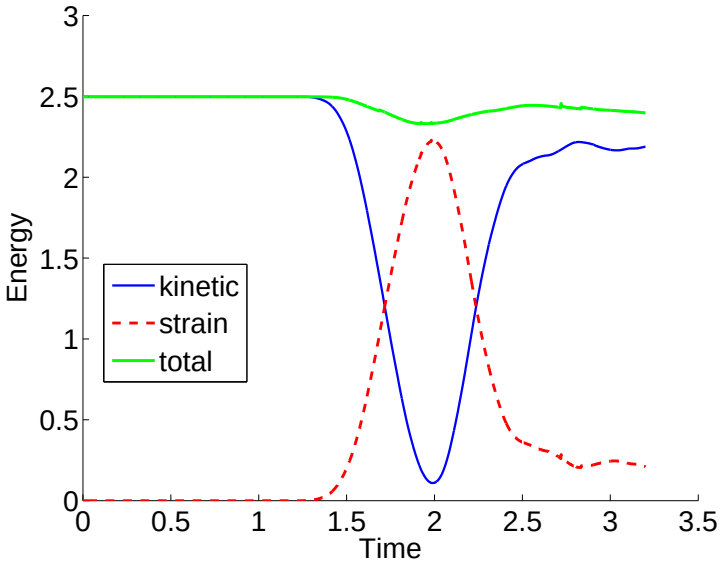

Figure 19: Impact of two elastic bodies: evolution of strain, kinetic and total energies in time.

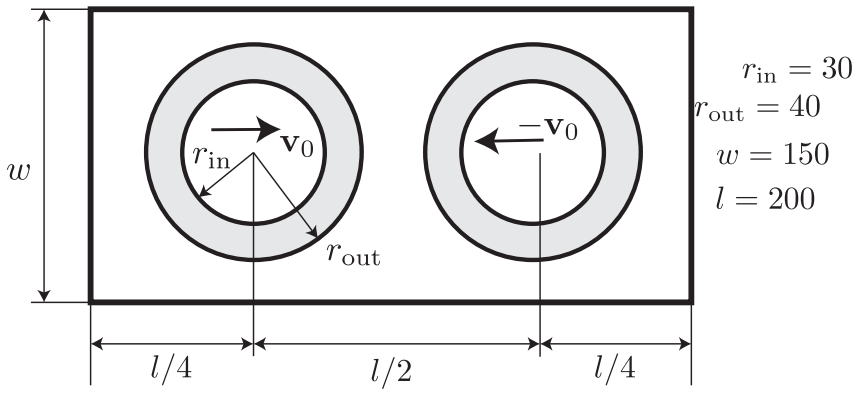

Figure 20: Impact of two elastic bodies: problem description. Dimensions are in millimeters.

7.4 Impact of two elastic rings

This example problem involves two hollow elastic cylinders, under the assumption of plane strain, impacting each other Fig. 20. The material is a compressible Neo-Hookean with bulk modulus $K = 121.7$ MPa, shear modulus $\mu = 26.1$ MPa, the density is $\rho = 1010 \times 10^{-12}$ kg/mm^3. The Cauchy stresses are computed as follows

$$\boldsymbol{\sigma} = \frac{1}{J}\left[\mu_0(\mathbf{b} - \mathbf{I}) + \lambda_0 \ln J\mathbf{I}\right] \tag{46}$$

where $J = \det \mathbf{F}$ and $\mathbf{b} = \mathbf{F}\mathbf{F}^{\mathrm{T}}$ is the left Cauchy strain tensor; $\lambda_0 = K - 2/3\mu_0$ is the first Lamé constant.

A grid of 100×60 elements is adopted and each cylinder is discretized by 5488 particles. The time step was chosen as $\Delta t = 0.2\sqrt{E/\rho} = 1.5 \times 10^{-6}$ s. For this problem, it is very

hard to find a cutoff value in Equation (22). We therefore used Equation (24) to compute the nodal velocities which are subsequently used to compute the particle velocity gradient $\mathbf{L}_p$. For visualization, the particle data (positions and stresses) are written to a VTP file (PolyData) and processed in Paraview using the filter Glyph.

8 Generalized Interpolation Material Point Method-GIMP

The original MPM with C^0 shape functions suffers from the so-called cell crossing instability. In order to illustrate this phenomenon, consider a 1D MPM discretisation shown in Fig. 22 in which all particles have the same stress (i.e., uniform stress state), and the same volume and a uniform element size. Each element has two particles, Fig. 22a. The internal force at node 2 is identically zero in the absence of body force. When a particle have just moved to a new cell, Fig. 22b, the internal force at node 2 is non-zero resulting in non-equilibrium. One solution is not to reset the grid at the end of the time step. However doing so can result in mesh distortion which makes MPM resemble to FEM. The cell-crossing error can be mitigated using high order B-splines basis functions or the generalized interpolation material point (GIMP) method.

These functions, which can be interpreted as averages of the basis function and its gradient over the particle domain, are used for mapping and interpolating data between particles and grid nodes as well as calculating internal and external forces.

$$\phi_{Ip} \equiv \phi_I(\mathbf{x}_p) = \bar{S}_{Ip} = \frac{1}{V_p}\int_{\Omega_\chi} \chi(\mathbf{x}-\mathbf{x}_p) N_I(\mathbf{x}) d\Omega \tag{47}$$

$$\nabla\phi_{Ip} \equiv \nabla\phi_I(\mathbf{x}_p) = \nabla\bar{S}_{Ip} = \frac{1}{V_p}\int_{\Omega_\chi} \chi(\mathbf{x}-\mathbf{x}_p) \nabla N_I(\mathbf{x}) d\Omega \tag{48}$$

where $\chi(\mathbf{x})$ is the particle characteristic function centered at the particle position and Ω_χ denotes the current support of this function; N_I are the standard linear shape functions. Typically piecewise constant particle characteristic function is used

$$\chi(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \Omega_p \\ 0 & \text{otherwise} \end{cases} \tag{49}$$

More precisely in 1D the following particle characteristic function is usually employed

$$\chi(x)_p = \begin{cases} 1 & \text{if } |x| \leq l_p/2 \\ 0 & \text{otherwise} \end{cases} \tag{50}$$

Here l_p is the current particle size. The initial particle size is determined by dividing the cell spacing L by the number of particles per cell. In order to have a closed form expression for the weighting functions given in Equation (48) so as to avoid numerical integration of these functions, it is assumed that Ω_p is always a rectangle which is aligned with the grid. In this case, one can explicitly compute the weighting function ϕ as

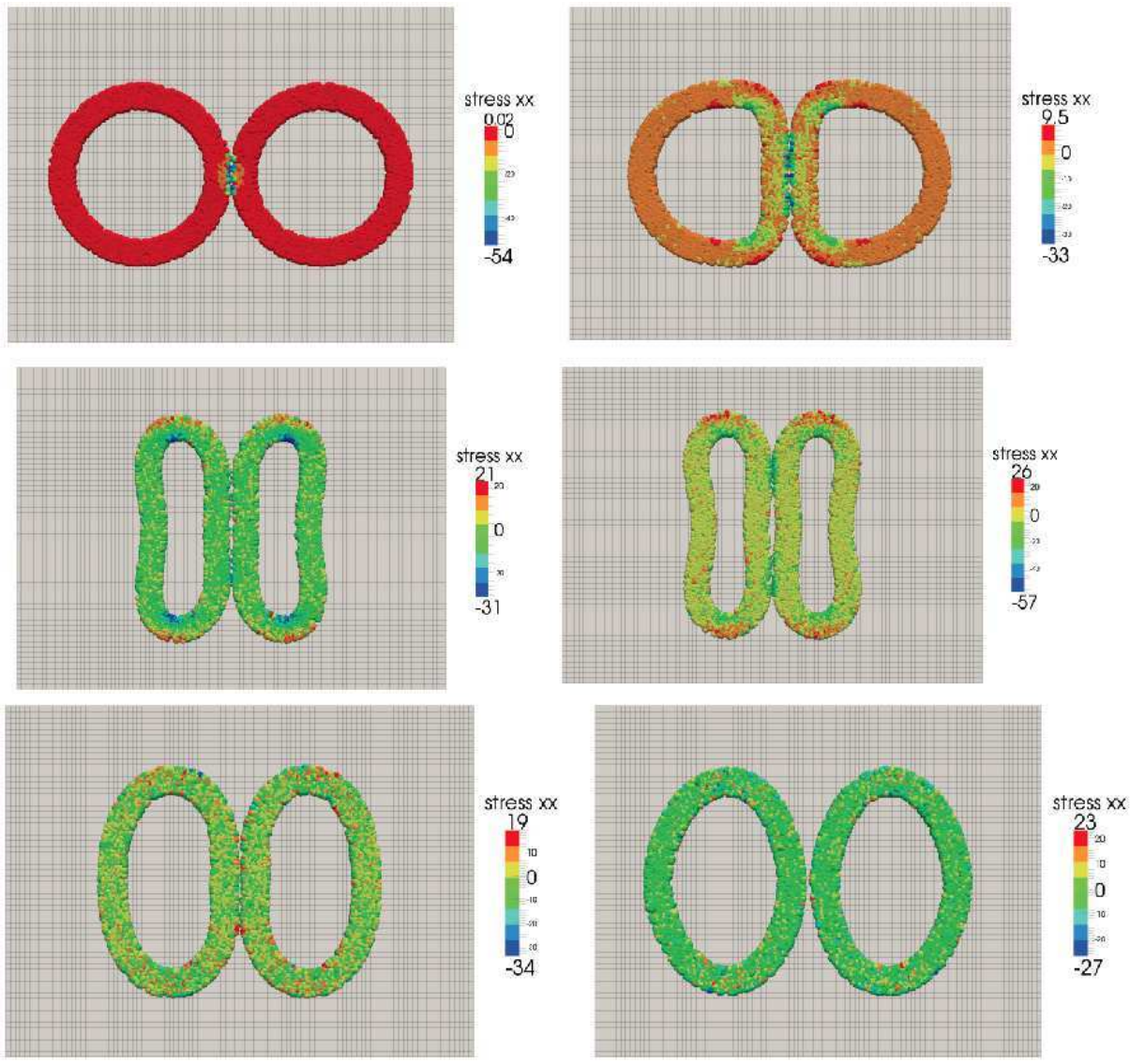

Figure 21: Impact of two elastic bodies: simulation snapshots.

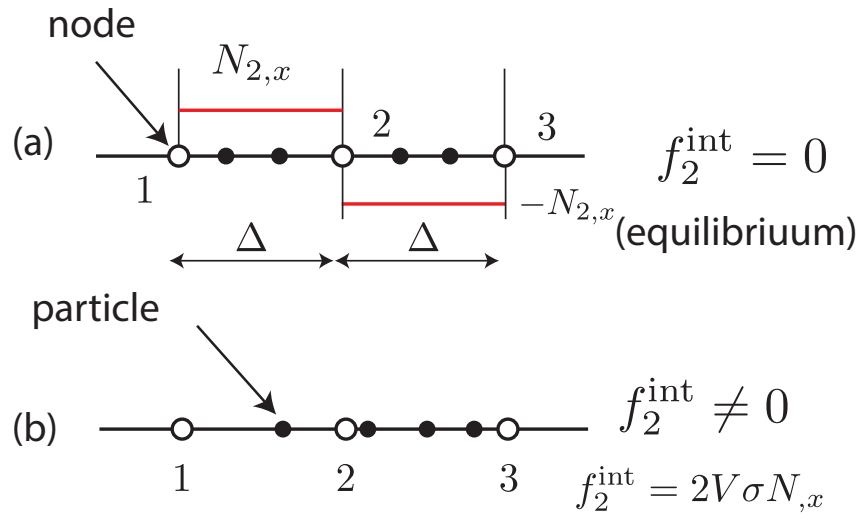

Figure 22: Cell crossing issue in MPM.

$$\phi(x) = \begin{cases} 1 - (4x^2 + l_p^2)/(4hl_p) & \text{if } |x| < 0.5l_p \\ 1 - |x|/h & \text{if } l_p \leq |x| \leq h - 0.5l_p \\ (h + l_p/2 - |x|)^2/(2hl_p) & \text{if } h - 0.5l_p \leq |x| < h + 0.5l_p \\ 0 & \text{otherwise} \end{cases} \tag{51}$$

of which an example is given in Fig. 23. Note that $\phi_I = \phi(x - x_I)$. As can be seen, if there are many particles per element (l_p is getting smaller), the GIMP functions resemble the standard FE hat functions.

Since the GIMP functions depend on the particle size which in turn evolves in time, the GIMP functions are particle specific and time-dependent. To simplify things, Ω_p is kept fixed and the resulting approach is referred to as uGIMP (unchanged GIMP). A more complicated approach, known as cpGIMP (contiguous particle GIMP), is to update the particle domain using the deformation gradient $\mathbf{F}$, cf. Fig. 24. Still, the updated particle domain is a rectangle in 2D and space cannot be tiled.

The implementation of uGIMP follows closely the MPM except two things: (i) the GIMP functions ϕ_I replace the FE hat functions N_I and (ii) a larger connectivity array of the elements i.e., the particles not only contribute to the nodes of the element they locate but also nodes of neighboring elements.

$$\chi(\mathbf{x})_p = V_p \delta(\mathbf{x} - \mathbf{x}_p) \tag{52}$$

9 Implicit time integration

The explicit code operates with a time step restricted by the usual Courant-Friedrichs-Lewy (CFL) condition, and thus the code resolves all waves. Simulating many manufacturing and

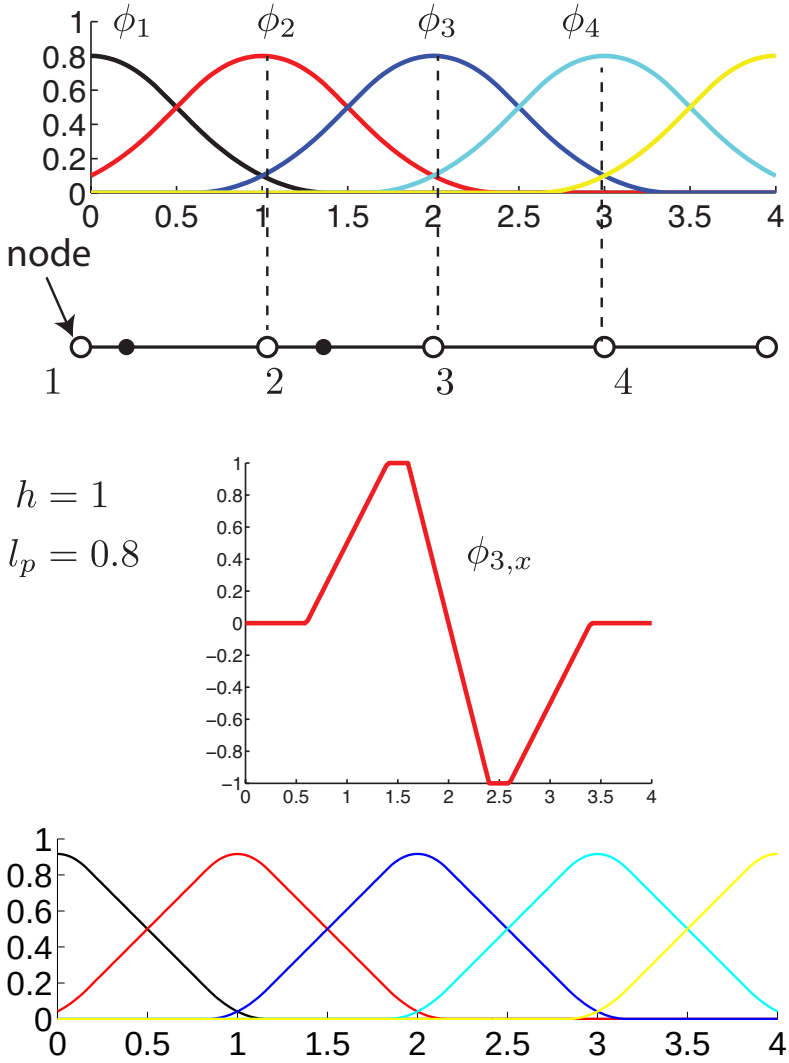

Figure 23: One dimensional GIMP basis functions.

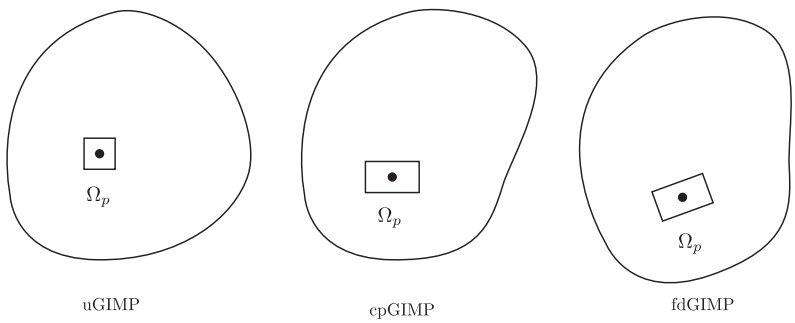

Figure 24: Tracking particle domain in GIPM: space cannot be tiled in a general multi-dimension domain using rectilinear Ω_p.

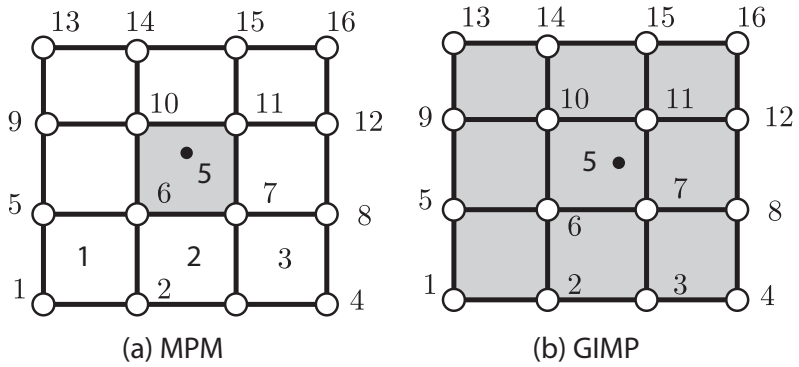

Figure 25: Larger element connectivity in GIMP (b) than in MPM (a). The connectivity of an element in GIMP can be determined by getting firstly its neighboring elements (very fast for structured grids used in MPM/GIMP) and then the nodes of these elements.

other problems with an implicit method can result in a considerable reduction in computational cost if one is interested only in the low-frequency, bulk motion, and not in the elastic wave motion.

10 Fluid-structure interaction

- Action of the air on aeronautic structures (aeroelasticity, flutter)
- Effect of the wind on civil structures (bridges, suspended cables, skyscrapers)
- Effect of water movement on dam or sloshing of a fluid in a container
- Fluid-membrane interaction (parachutes, vehicle airbags, blood vessels etc.)

The numerical procedures to solve the Fluid-structure interaction (FSI) problems may be broadly classified into two approaches

- Monolithic approach: the equations governing the fluid flow and the structure deformation are solved in a single solver;
- Partitioned approach: the fluid and solid equations are solved separately using different solvers.

The monolithic approach requires a code developed for this particular FSI problem whereas the partitioned approach preserves software modularity because an existing flow solver and solid solver are coupled. On the other hand, development of stable and accurate coupling algorithms is required in partitioned approaches. Particularly, the fluid-solid interface location is not known a priori and usually changed in time; thus, the partitioned approach requires the tracking of the new interface location and its related quantities, which can be cumbersome and error-prone.

The partitioned approach is also referred to as coupling method as there are two models which are coupled. Within the coupling method, one can differentiate between one-way and two-way coupling. The coupling is one-way if the motion of a fluid flow influences a solid structure but the reaction of a solid upon a fluid is negligible. The other way around is also possible. Moreover, the coupling can be loose or tight coupling.

$$\boldsymbol{\sigma}^f = 2\mu\dot{\boldsymbol{\epsilon}} - \frac{2}{3}\mu\text{tr}(\dot{\boldsymbol{\epsilon}})\mathbf{I} - \hat{p}\mathbf{I} \tag{53}$$

$$\hat{p} = (\gamma - 1)\rho e \tag{54}$$

e is the specific internal energy and γ is the ratio of specific heats.

References

S.G. Bardenhagen. Energy Conservation Error in the Material Point Method for Solid Mechanics. *Journal of Computational Physics*, 180(1):383–403, 2002.

M Steffen, PC Wallstedt, and JE Guilkey. Examination and analysis of implementation choices within the material point method (MPM). *Computer Modeling in Engineering and Sciences*, 31(2):107–127, 2008.

D Sulsky and HL Schreyer. Axisymmetric form of the material point method with applications to upsetting and Taylor impact problems. *Computer Methods in Applied Mechanics and Engineering*, 0457825(96), 1996.

D Sulsky, Z Chen, and HL Schreyer. A particle method for history-dependent materials. *Computer Methods in Applied Mechanics and Engineering*, 5, 1994.

D. Sulsky, S.J. Zhou, and H. L. Schreyer. Application of a particle-in-cell method to solid mechanics. *Computer Physics Communications*, 87(1-2):236–252, 1995.

D. Sulsky, H.L. Schreyer, K. Peterson, R. Kwok, and M. Coon. Using the material-point method to model sea ice dynamics. *Journal of Geophysical Research*, 112(C2):C02S90, 2007.

AR York, D. Sulsky, and HL Schreyer. The material point method for simulation of thin membranes. *International Journal for Numerical Methods in Engineering*, 1456(May 1998), 1999.

AR York, D. Sulsky, and HL Schreyer. FluidâĂŞmembrane interaction based on the material point method. *International Journal for Numerical Methods in Engineering*, pages 901–924, 2000.