Scientific Computation

Vinh Phu Nguyen
Alban de Vaucorbeil
Stephane Bordas

The Material Point Method

Theory, Implementations and Applications

Springer

The Material Point Method

Scientific Computation

Vinh Phu Nguyen · Alban de Vaucorbeil ·
Stephane Bordas

The Material Point Method

Theory, Implementations and Applications

Springer

Vinh Phu Nguyen
Department of Civil Engineering
Monash University
Clayton, VIC, Australia

Alban de Vaucorbeil
Institute for Frontier Materials
Deakin University
Geelong, VIC, Australia

Stephane Bordas
University of Luxembourg Campus
Kirchberg
Luxembourg, Luxembourg

ISSN 1434-8322 ISSN 2198-2589 (electronic)
Scientific Computation
ISBN 978-3-031-24069-0 ISBN 978-3-031-24070-6 (eBook)
https://doi.org/10.1007/978-3-031-24070-6

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Computer simulations have become an integral tool in engineering, ranging from civil and mechanical engineering to material sciences and beyond. While the finite element method (FEM) has long been the standard framework for simulations, it reaches its limits when dealing with problems involving large deformation and fractures. To overcome these limitations, meshless methods (MMs) such as smoothed-particle hydrodynamics (SPH) and the material point method (MPM) have emerged as a promising alternative. MPM, in particular, combines the advantages of FEM and MMs, representing the material by a set of particles overlaid on a background mesh. This approach has been successful in simulating a wide variety of large-deformation and complicated engineering problems.

The book not only re-examines previous contributions but also organizes them in a coherent fashion and anticipates new advancements. Sample algorithms for benchmark problems are available on the book's website, allowing researchers and graduate students to modify them and develop their own solution algorithms for specific problems. The goal of this book is to provide students and researchers with a theoretical and practical knowledge of the material point method for the simulation of engineering problems, and to promote further in-depth studies in the field.

This book aims at being a comprehensive guide to the Material Point Method that focuses on its use in solving problems in mechanics, physics and engineering. The book contains nine main chapters that build on each other to provide a detailed understanding of the MPM.

Chapter 1 provides an introduction to the Material Point Method and its advantages over other numerical methods. It also gives an overview of the topics covered in the book.

Chapter 2 covers the mathematical foundations of the MPM. It discusses the principles of continuum mechanics, the kinematic description of motion and the integration algorithms used in the MPM.

Chapter 3 presents different MPM versions that basically adopt different shape functions, e.g. hat functions, B-splines, Bernstein, GIMP and CPDI. Also treated is a new formulation called generalized particle in cell (GPIC) which combines the FEM and MPM that takes advantages of both methods.

Chapter 4 covers the constitutive models used in the MPM. This includes linear elastic isotropic materials, hyperelastic solids and elasto-plastic materials. The chapter also discusses the Johnson-Cook flow model and the algorithm used to compute damage.

Chapter 5 provides some implementation details such as particle generation for simple geometries and for images, initial and boundary conditions, MPM with unstructured grids and visualization of MPM results.

Chapter 6 presents a tutorial MPM code written in MATLAB. This code serves to illustrate the MPM algorithms discussed in the book and it can be used to solve one, two and three dimensional problems.

Chapter 7 describes Karamelo—an open-source parallel C++ package for the material point method. This code can be used to solve large-scale problems as it can be run on multiple processors using MPI. With such an efficient code, we present three dimensional simulations to demonstrate the capability of the MPM in solving large deformation solid mechanics problems.

Chapter 8 presents some advanced topics such as contacts and fracture. Both frictionless and frictional contact are discussed. One notable application of these contact algorithms is the simulation of scratch test—a popular mechanical test to measure a solid's hardness. We then discuss fracture modeling in the framework of the MPM. Application of the MPM to model large strain ductile fracture of metals is then provided.

Chapter 9 discusses the mathematical analysis of the MPM regarding its stability and accuracy. We discuss the conservation of energy and momenta in various MPM variants. We study the convergence behavior of all MPM variants discussed in this book for a problem involving only compression/tension deformation and another problem involving simple shear with superimposed rotation.

Chapter 10 discusses fluid/gases modeling, membrane modeling and heat conduction. With the information provided in this chapter, one can carry out fluid-structure interaction simulations, air-bag simulations and thermo-mechanical simulations.

In addition to the content of the main chapters, the book has several appendices that provide supplementary information. Appendix A discusses the strong and weak form of the momentum equation and their equivalence. Appendix B presents derivation of various CPDI basis functions. Some useful utilities such as how to use an open-source computer algebra system (SageMath and SymPy) to derive CPDI functions, how to use remote machines to run large-scale simulations and consistent units are given in Appendix C. Appendix D gives a short but practical presentation of updated and total Lagrangian, explicit dynamics FEM for nonlinear solid mechanics. Appendix E treats implicit dynamics FEM so that it is easier to understand implicit MPM (even though this is not discussed in this book). Finally, we describe another MPM code in Appendix F, now written in Julia—a new high-level dynamic programming language which is easy to use as Python and as fast as C.

The book also includes several simulations related to these topics. The book provides sample algorithms for benchmark problems, which are available on the

book's website. These algorithms can be modified and used to develop custom solution algorithms for specific problems. The book includes MATLAB, Julia and C++ codes, derivations, and references to other studies in the field.

We would like to thank Prof. Deborah Sulsky at University of New Mexico for reading through the first draft and giving comments. Also, the first author acknowledges the fruitful discussions with Dr. Rebecca Brannon at University of Utah when he started working on the MPM.

Clayton, Australia — Vinh Phu Nguyen
Geelong, Australia — Alban de Vaucorbeil
Luxembourg, Luxembourg — Stephane Bordas

Reference

Zhang, X., Chen, Z., Liu, Y.: The Material Point Method-A Continuum-Based Particle Method for Extreme Loading Cases. Academic Press, Cambridge (2016a)

Contents

Chapter 1
Introduction

The aim of this introductory chapter is to provide an overview of what is the material point method—the topic of the book—and its application domains. We start with a brief introduction to the field of computational sciences and engineering to explain the role of computer simulations using a computational model (Sect. 1.1). The MPM is one of such computational models. The role of experiments cannot be underestimated even for a computational scientist and engineer (Sect. 1.2). Then, we briefly discuss initial-boundary value problems and numerical methods in Sect. 1.3. Next, we talk about mesh-based and mesh-free methods in Sect. 1.4 as the MPM adopts tools from these two classes. This is followed by Sect. 1.5 where we will present a picture of the MPM. Applications of the MPM in various engineering and sciences fields are given in Sect. 1.6. Open source and commercial MPM codes are presented in Sect. 1.7. The layout of the book is given in Sect. 1.8. Finally, our notations are explained in Sect. 1.9.

1.1 Computational Sciences and Engineering

The topic of this book falls within the scope of computational sciences and engineering (CSE). CSE is a relatively new discipline that deals with the development and application of *computational models*, often coupled with high-performance computing, to solve complex problems arising in engineering analysis and design (computational engineering) as well as in natural phenomena (computational science). CSE has been described as the "third mode of discovery" next to theory and experimentation.

Within the realm of CSE these are steps to solve a problem. First, a mathematical model that best describes the problem is selected or developed. This step of model development is done manually by people with sufficient mathematical skills. A majority of mathematical model is developed using calculus (see Remark 1 for a history account) and thus they are continuous models not suitable for digital computers.

V. P. Nguyen et al., *The Material Point Method*, Scientific Computation,
https://doi.org/10.1007/978-3-031-24070-6_1

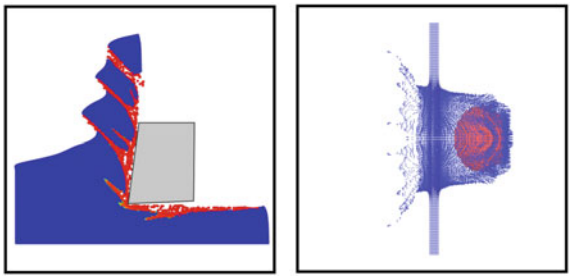


Fig. 1.1 Computer experiments: experiments done with computational models on a digital computer

Second, a computational model of this mathematical model is derived. A computational model is an approximation to the mathematical model and is in a discrete form which can be solved using computers. Third, this discrete model is implemented in a programming language (Fortran in the past and C++ and Python nowadays) to have a computational code or platform. For solid mechanics, popular computational platforms are Abaqus and LS-Dyna. Finally, these computational platforms are used to perform *computer simulations* or *computer experiments* (Fig. 1.1).

Computer simulations are not only useful to solve problems too complex to be resolved analytically, but are also increasingly replacing costly and time consuming experiments. Furthermore, they can provide tremendous information at scales of space and time where experimental visualization is difficult or impossible. And finally, simulations also have a value in their ability to predict the behavior of materials and structures that are yet to be created; experiments are limited to materials and structures that have already been created.

This book presents MPM models i.e., discrete models based on the material point method for the problem of understanding and prediction of the deformation of solids and fluids that undergo very large deformation. The mathematical model for this problem is based on the theory of *continuum mechanics*, see e.g. Malvern (1969). We also discuss computer implementation of these MPM models and provide tutorial MPM codes written in Matlab and Julia and an efficient MPM platform named `Karamelo` which can replace contemporary FE packages such as LS-Dyna and Abaqus for certain problems.

Remark 1 We would like to discuss briefly why calculus is so dominant in sciences and engineering. It all started with the works of Galilei and Newton who discovered that the laws of nature can be unreasonably well described by mathematics, particularly calculus. If the motion of heavenly bodies can be modeled using mathematics, then it is logical to apply it to humanity problems. This was exactly what the geniuses like Bernoulli brothers, Euler, Lagrange, Cauchy had done some 300 years ago. These men developed partial differential equations that can model a wide range of phenomena such as the deformation of fluids, gases and solids. It is the models described by these PDEs that put men on the moon, give us cell phones, computers, radio. Or television. Or ultrasound for expectant mothers, or GPS for lost travelers.

Remark 2 Calculus has two parts: differential calculus and integral calculus. The latter is interesting as we go from finiteness to infinitum. And to solve it numerically, we do the reverse: from infinitum back to finiteness. We replace a solid with infinite number of degrees of freedom by a mesh consisting of just a finite number of degrees of freedom.

1.2 The Role of Experiments in CSE

It is certain that a computational model requires experiments to obtain parameters used in the model. To emphasize the vital role of experiments in sciences and engineering, we consider the interesting article of Boyce et al. (2016) that presents the Sandia fracture challenge to the computational fracture community. The challenge involves the simulation of the fracture of a steel sample of complex geometry. Only a minimum experimental data (tensile test of a steel coupon) was provided to the analysist. Different research groups, who participated in the challenge, used all existing fracture models and none provided a match with the experiment.

Therefore simulations are simply insufficient and thus a combined experiment-simulation programme should be pursued for any problem. It is interesting to know that R. W. Clough, the exact man who coined the term 'finite element method' some 70 years ago, stopped working on the method and switched to experiments (Clough 1980).

1.3 One Dimensional Wave Equation

In science and engineering, one commonly seeks the response to some excitations of a certain kind of system. This system can be mechanical, chemical, biological …To this end, it is common practice to adopt a mathematical model for the system and try to solve it. Usually the model equations are partial differential equations (PDE); for example, the Navier-Stokes equations in fluid mechanics, or the momentum conservation equations in solid mechanics. These differential equations together with both the initial and boundary conditions constitute an initial-boundary value problem (IBVP). In general, solving a boundary value problem by classical analytical methods is almost impossible. Therefore, an approximate solution to the IBVP is sought.

Approximate solutions to an IBVP are obtained by transforming the PDE into a set of algebraic equations. This is achieved by discretizing the space and time domain. Common spatial discretization methods include mesh-based methods such as the finite element method (FEM), the finite volume method (FVM), the finite difference method (FDM) and meshless or meshfree methods (MMs). Time discretization is mostly based on finite differences e.g. the forward Euler method and the leaf-frog method.

In this section, we provide an overview of how to use a numerical method to obtain approximate solutions to IBVPs. This discussion is not meant to be rigorous, but rather to present the basic concepts of numerical methods and a general procedure to go from PDEs, which are difficult to solve, to algebraic equations, which can be handled quite well by nowadays computers. We have decided to present methods using a weak form as the MPM follows this so-called Galerkin method.

For a simple demonstration of the basic concepts in numerical methods, let's consider the one dimensional momentum equation, that governs the deformation of a solid object due to applied external forces:

$$\rho \frac{\partial^2 u}{\partial t^2} = E \frac{\partial^2 u}{\partial x^2} + \rho b \tag{1.1}$$

where $u(x, t)$ is the displacement field, E the Young modulus of the material, ρ the density and b the body force. The spatial domain is $0 \leq x \leq L$ and the time domain is $0 \leq t \leq T$.

For the case of zero body force (i.e. $b = 0$) the above equation becomes the well known one dimensional wave equation written as:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \quad c = \sqrt{\frac{E}{\rho}} \tag{1.2}$$

Remark 3 Solving Eq. (1.2) for $u(x, t)$ with a given c is called *a forward problem*. Inversely, determining c so that Eq. (1.2) has a solution matching a predefined $\bar{u}(x, t)$ is coined *an inverse problem*.

In order for a PDE to have unique solutions, initial and boundary conditions have to be provided. For example, the so-called Dirichlet boundary conditions read

$$u(0, t) = a, \quad u(L, t) = b, \quad t > 0 \tag{1.3}$$

where a, b are some constants. Note that there exists other types of boundary conditions such as Neumann condition and Robin condition. As Eq. (1.2) involves second derivative with respect to t, two initial conditions are required which are given by

$$u(x, 0) = f(x), \quad \dot{u}(x, 0) = g(x) \tag{1.4}$$

where $\dot{u} := du/dt$ and f, g are some functions.

Putting all the above together we come up with the following initial-boundary value problem

$$\begin{aligned} &\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} && \text{(wave equation)} \\ &u(0, t) = a, \quad u(L, t) = b, \quad t > 0 && \text{(boundary conditions)} \\ &u(x, 0) = f(x), \quad \dot{u}(x, 0) = g(x) && \text{(initial conditions)} \end{aligned} \tag{1.5}$$

of which approximate solutions are sought for using a numerical method. Equation (1.5) is called a *strong form* of the wave equation. Some numerical methods such as FDM or collocation methods work directly with this strong form even though they are often less accurate compared with Galerkin methods—those that employ a weak formulation—and unstable. However, these methods are quite efficient as no numerical integration is needed.

The finite element methods (or generally Galerkin based methods) adopt a weak formulation where the partial differential equations are restated in an integral form called the *weak form*. A weak form of the differential equations is equivalent to the strong form. In many disciplines, the weak form has a physical meaning; for example, the weak form of the momentum equation is called the principle of virtual work in solid/structural mechanics.

To obtain the weak form, one multiplies the PDE i.e., the wave equation in this particular context, with an arbitrary function $w(x)$, called the *weight function*, and integrate the resulting equation over the entire domain. That is

$$\int_0^L \left[\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2}\right] w(x)\mathrm{d}x = 0, \quad \forall w(x) \text{ with } w(0) = w(L) = 0 \tag{1.6}$$

The arbitrariness of the weight function is crucial, as otherwise a weak form is not equivalent to the strong form. In this way, the weight function can be thought of as an enforcer: whatever it multiplies is enforced to be zero by its arbitrariness.

Using the integration by parts for the second term, the above equation becomes

$$\int_0^L \frac{\partial^2 u}{\partial t^2} w(x)\mathrm{d}x + c^2 \int_0^L \frac{\partial u}{\partial x}\frac{\partial w}{\partial x}\mathrm{d}x = 0 \tag{1.7}$$

where the spatial derivative of the unknown field, $u(x, t)$, was lowered from two to one. It is a great achievement by a simple derivation as constructing high order approximations of u, so that second derivatives are computable, is much more difficult than constructing linear approximations. Furthermore, the second term is now symmetric, this is significant as the resulting matrix will be symmetric. Symmetric matrices possess nice properties e.g. less storage and real eigenvalues.

The weak form of the wave equation is thus given by: find the smooth function $u(x, t)$ such that

$$\begin{aligned} \int_0^L \frac{\partial^2 u}{\partial t^2} w(x)\mathrm{d}x + c^2 \int_0^L \frac{\partial u}{\partial x}\frac{\partial w}{\partial x}\mathrm{d}x = 0 \\ u(0, t) = a, \quad u(L, t) = b \\ u(x, 0) = f(x), \quad u(x, 0) = g(x) \end{aligned} \tag{1.8}$$

for all $w(x)$ with $w(0) = w(L) = 0$.

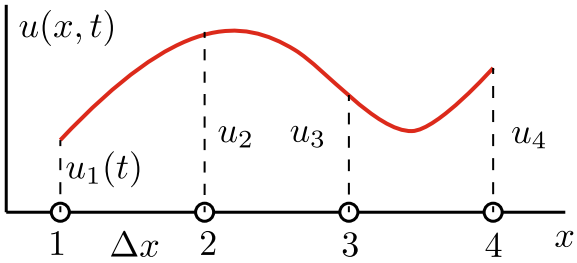


Fig. 1.2 Spatial discretization in one dimension

The basic idea of numerical methods in solving PDEs is to discretize the spatial and temporal domain i.e., instead of working with infinite number of points (or nodes) within the domain of interest $[0, L] \times [0, T]$, one first discretize the spatial domain into a finite number of points x_I, $I = 1, 2, \ldots, n$. Next, the unknown function $u(x, t)$ is approximated using the values of u evaluated at those discrete points x_I (Fig. 1.2), and this approximation is then substituted into the weak form i.e., Eq. (1.8) to obtain a set of ordinary differential equations (ODEs). Finally using any time integration methods of ODEs to advance in time. At this stage, the PDE has been completely transformed into a discrete form—a system of algebraic equations—which can be easily solved by digital computers. This is known as the method of lines. There exists methods which involve full discretization in both space and time, but they are less popular and not further discussed in this book.

The approximation of the unknown field $u(x, t)$ is written as

$$u(x,t) \approx u^h(x,t) = \sum_I^n N_I(x) u_I(t) \tag{1.9}$$

where $N_I(x)$ are the approximation functions or shape functions in the FEM context and $u_I(t)$ denotes the value of u at point I at time instant t and constitutes the unknowns to be solved. The support of node I is defined as the set of points where $N_I(x) \neq 0$. Usually, only a few points are within the support of a given node and thus the shape function is said to have a compact support. And this compact support is crucial to the computational efficiency of the method: the resulting matrices are sparse not full. If the support of $N_I(x)$ is the whole domain, the method is called a spectral method. After having obtained u_I, Eq. (1.9) is used to compute the function at any other points.

Even though there are many choices for the weight functions w, in the Bubnov-Galerkin method, which is the most commonly used method at least for solid mechanics applications, the weight function is approximated using the same shape functions as u. That is

$$w(x,t) = \sum_I^n N_I(x) w_I \tag{1.10}$$

where w_I are the nodal values of the weight function; they are not functions of time.

Now, the numerical solution of the weak form of the wave equation i.e., Eq. (1.8) is thus given by: find u_J such that

$$\int_0^L \left(N_I(x)\ddot{u}_I\right)\left(N_J(x)w_J\right)\mathrm{d}x + c^2\int_0^L \left(\frac{\partial N_I}{\partial x}u_I\right)\left(\frac{\partial N_J}{\partial x}w_J\right)\mathrm{d}x = 0 \tag{1.11}$$

for all w_J. Note that we have used the Einstein summation rule: indices which are repeated twice in a term are summed, see Sect. 1.9 for detail.

The arbitrariness of w_J results in the following system of ordinary differential equations

$$\begin{bmatrix} \int_0^L N_1N_1\mathrm{d}x & \int_0^L N_1N_2\mathrm{d}x & \dots & \int_0^L N_1N_n\mathrm{d}x \\ \int_0^L N_2N_1\mathrm{d}x & \int_0^L N_2N_2\mathrm{d}x & \dots & \int_0^L N_2N_n\mathrm{d}x \\ \vdots & \vdots & \vdots & \vdots \\ \int_0^L N_nN_1\mathrm{d}x & \int_0^L N_nN_2\mathrm{d}x & \dots & \int_0^L N_nN_n\mathrm{d}x \end{bmatrix} \begin{bmatrix} \ddot{u}_1 \\ \ddot{u}_2 \\ \vdots \\ \ddot{u}_n \end{bmatrix}$$

$$+ c^2 \begin{bmatrix} \int_0^L dN_1dN_1\mathrm{d}x & \int_0^L dN_1dN_2\mathrm{d}x & \dots & \int_0^L dN_1dN_n\mathrm{d}x \\ \int_0^L dN_2dN_1\mathrm{d}x & \int_0^L dN_2dN_2\mathrm{d}x & \dots & \int_0^L dN_2dN_n\mathrm{d}x \\ \vdots & \vdots & \vdots & \vdots \\ \int_0^L dN_ndN_1\mathrm{d}x & \int_0^L dN_ndN_2\mathrm{d}x & \dots & \int_0^L dN_ndN_n\mathrm{d}x \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{1.12}$$

where $dN_I = dN_I/dx$ is the first spatial derivative of the shape function N_I. The integrals in the above equation are called weak form integrals. For this simple 1D problem, they can be exactly computed, but generally, numerical integration is used to evaluate these integrals. We refer to Sect. 2.4 for a discussion on numerical integration.

And Eq. (1.12) can be cast in the following compact equation using a matrix notation

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{0} \tag{1.13}$$

where $\mathbf{u}$ and $\ddot{\mathbf{u}}$ are the vector of displacements and accelerations of the whole problem, respectively. They are one dimensional array of length n; $\mathbf{M}$ and $\mathbf{K}$ are the mass and stiffness matrix—matrices of dimension $n \times n$.

Equation (1.13) is referred to as the *semi-discrete equation* as the time has not been yet discretized. Any time integration methods for ODEs can be used to discretize Eq. (1.13) in time. A time integration scheme is called *implicit* if one has to solve a system of algebraic equations to obtain $\mathbf{u}$ at a time instant t. This system of algebraic equations is very large for practical problems: it is not uncommon to encounter a system of millions of unknowns. On the other hand, it is called *explicit* if one can get $\mathbf{u}$ without solving any system of algebraic equations. Implicit time integration schemes allow large time increments i.e., there are fewer time steps to resolve whereas explicit schemes require small time increments. Generally, explicit schemes are preferred for fast transient problems such as impact simulations.

For time discretization, the time interval $[0, T]$ is partitioned into a number of time steps Δt i.e., the semi-discrete equation is evaluated at discrete time instants $t^m = (m-1)\Delta t$. Assuming that we are at time t and need to advance to time $t + \Delta t$. By using the central difference scheme, which is the most commonly used explicit time integration method, we have

$$\mathbf{u}^{t+\Delta t} = \Delta t^2 \ddot{\mathbf{u}}^t + 2\mathbf{u}^t - \mathbf{u}^{t-\Delta t} \tag{1.14}$$

which allows us to obtain the displacements at $t + \Delta t$ upon substitution of Eq. (1.13) for $\ddot{\mathbf{u}}^t$

$$\mathbf{u}^{t+\Delta t} = -\Delta t^2 \mathbf{M}^{-1}\mathbf{K}\mathbf{u}^t + 2\mathbf{u}^t - \mathbf{u}^{t-\Delta t} \tag{1.15}$$

To avoid inversion of the mass matrix, a technique known as *mass lumping* is often adopted to make $\mathbf{M}$ diagonal.

The final step is to impose Dirichlet boundary conditions e.g. $u(0, t) = a$. If the shape functions N_I have been constructed such that they satisfy the Kronecker delta property then it is pretty straightforward to impose Dirichlet conditions: one simply override the displacements computed in Eq. (1.15) by the prescribed values. In this specific case, simply setting $u_1 = a$ and $u_n = b$ does the job. Shape functions are said to satisfy the Kronecker delta property when they fulfill the following equation

$$N_I(x_J) = \delta_{IJ}, \quad \delta_{IJ} = \begin{cases} 1 & \text{if } I = J \\ 0 & \text{otherwise} \end{cases} \tag{1.16}$$

Therefore, condition $u(0) = a$ becomes $u(0) = \sum_I N_I(0)u_I = u_1 = a$.

What value should be assigned for n or in other words, how many nodes/points should we use? That is the eternal question of computational engineer. There is no theorem which says n should be such and such. A rule of thump is n should be big to have accuracy and not so big to reduce the cost. Practically, one pick an n, do the simulation and evaluate a certain quantity of interest (e.g. the maximum displacement or maximum stress) against analytical solutions (if any) or experimental

data. If a large difference exists, then double n and repeat until a convergence has been obtained. If no solution is available, then one needs to compare the numerical solutions of at least two resolutions (two different n) and they should be close to each other.

Up to this point, how the shape functions N_I are constructed is not yet discussed. In the next section, we discuss this construction of shape functions.

1.4 Mesh-Based and Meshfree Methods

Spatial discretization methods can generally be divided into groups: mesh-based methods (Sect. 1.4.1) and meshless or meshfree methods (Sect. 1.4.2). The three most common mesh-based methods are finite element method (FEM), finite volume method (FVM) and finite difference method (FDM). Herein we focus on FEM since it is the most widely used and commercially available method to date for solid mechanics. Furthermore, the material point method can be considered a variant of FEM.

1.4.1 Mesh-Based Methods

Since its inception about 70 years ago (Courant 1943; Clough 1960),[1] the finite element method has been used with great success in many fields with both academic and industrial applications. They have been the primary computational methodologies in engineering computations for more than half a century. The basic idea is to divide the domain of interest (generally with a complex shape) into a (finite) number of sub-domains called elements (Fig. 1.3). These elements have simple geometry e.g. triangles or quadrilaterals in two dimensions and tetrahedra in three dimensions. The elements are connected at nodes. A field quantity, such as the displacement field, is interpolated by a polynomial defined over the elements. Integrals in the weak form e.g. the stiffness matrix or force vectors are evaluated over individual elements using a quadrature rule (e.g. Gauss quadrature). For deformable solids of which behavior is history dependent (*inelastic solids*) it is the quadrature points where stresses, strains, history variables (such as equivalent plastic strain, damage variables) are stored.

As a simplest description of the FE shape functions we plot in Fig. 1.4 one dimensional linear and quadratic shape functions. The spatial domain is $[0, 4]$ which is divided into four equidistant elements. In the first case of linear elements, each element has two nodes. There are thus five nodes. For the case of quadratic elements, each element has three nodes–two nodes at the extremities and one mid-side node.

[1] For an interesting discount on the history of FEM, we refer to Clough (1980), Gander and Wanner (2012).

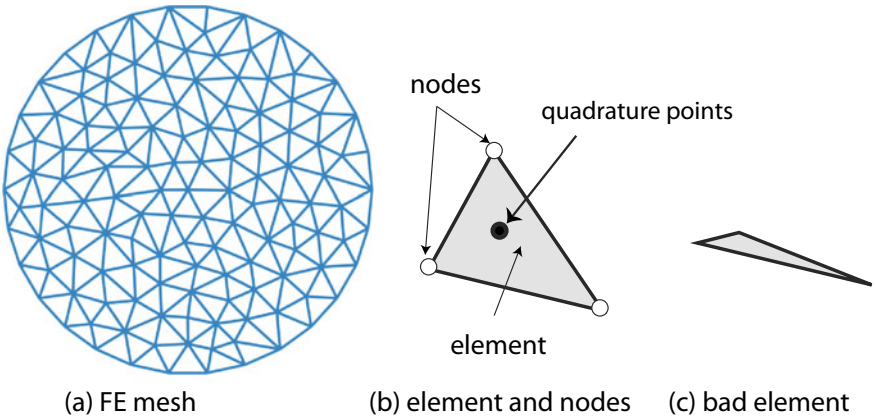


Fig. 1.3 The finite element method: domain is discretized into a number of elements

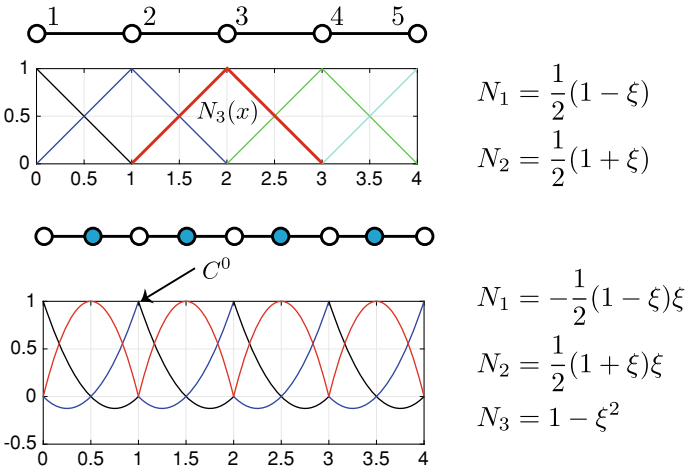


Fig. 1.4 Finite element shape functions in one dimension: linear elements (top) and quadratic elements (bottom). FE shape functions are polynomials defined in the so-called parent domain which is $[-1, 1]$ in one dimension. These shape functions satisfy the Kronecker delta property

Note that even though quadratic functions are smooth they are only C^0 across the element boundaries.

Remark 4 C^0 means that only the functions are continuous across element boundaries but their derivatives are not. This property poses a great challenge in solving PDEs with high order spatial derivatives which occur frequently in structural mechanics (i.e., PDEs that govern the behaviour of beams/plates/shells often involve fourth order derivatives of the primary unknown field).

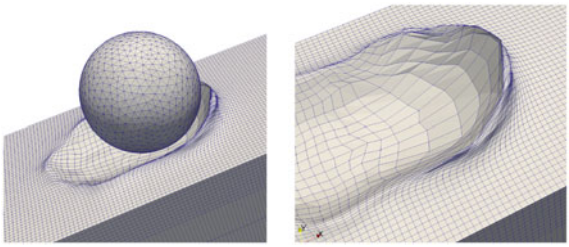


Fig. 1.5 Element distortion in the FEM: simulation of material scratch

The availability of the elements provides a natural way to evaluate the weak form integrals. For example, consider a function f in two dimensions, one can write

$$\int_{\Omega} f(x, y)\mathrm{d}\Omega = \sum_{e} \int_{\Omega_e} f(x, y)\mathrm{d}\Omega = \sum_{e} \sum_{g} f(x_g, y_g) w_g \tag{1.17}$$

where subscript e denotes the elements and subscript g denotes the integration points. The fact that FE shape functions are polynomials and the element domains Ω_e align with the support of the shape functions leads to a very accurate computation of the weak form integrals. This is in sharp contrast to meshfree methods (and also to the material point method) to be discussed shortly.

For problems exhibiting large deformation (precisely large strain) e.g. simulation of manufacturing processes such as extrusion and molding operations, the elements inevitably become distorted which leads to higher errors and even premature termination of the program (Fig. 1.5). This issue of element distortion is often solved using a technique called *remeshing* in which a new mesh with quality elements replaces the old mesh with distorted elements. There are two major problems with remeshing. First, it is a time and human labour consuming task, which is not guaranteed to be feasible in finite time for complex three-dimensional geometries. Second, remeshing requires mapping or projection of information from the old to the new mesh, a step that inevitably introduces error, particularly for inelastic materials involving history variables. The more history variables a material model has the more error this remeshing step will induce.

Another difficulty of FEM is the conversion of a continuum (Fig. 1.6a) into a finite element mesh of good quality (Fig. 1.6b), in a reasonable amount of time and involves least user intervention. This is because solid geometries are created in a CAD (Computer Aided Design) software, and FE simulations are carried out in a FE software that accepts only FE meshes. Meshfree methods were born to remove the remeshing burden of FEM. But it can alleviate the mesh burden as well; even though isogeometric analysis pioneered by Hughes et al. (2005) is probably better for this issue.

Fig. 1.6 Conversion from a CAD (**a**) to a FE mesh (**b**)

(a) A solid in a CAD system (b) Corresponding FE mesh

1.4.2 Meshless Methods

Meshless methods are so named as the space is discretized into a number of points, or particles, in which each point interacts with its neighboring points in a flexible manner: not via a rigid mesh as in the FEM. It should be noted that up to now, there is no unified framework for meshfree methods. This is reflected by the plethora of existing methods[2] in the literature. The oldest developed method is Smoothed Particle Hydrodynamics (SPH) introduced by Gingold and Monaghan (1977), Lucy (1977) which was used for modeling astrophysical phenomena without boundaries such as exploding stars and dust clouds. Nowadays, the SPH is a popular simulation technique in various engineering and science fields. Next, the Generalized Finite Difference Method of Liszka and Orkisz (1980) was proposed followed by the Diffuse Element Method (DEM) by Nayroles et al. (1992), the Element Free Galerkin (EFG) by Belytschko et al. (1994); the Material Point Method (Sulsky et al. 1994), the Reproducing Kernel Particle Method (RKPM) by Liu et al. (1995); the $h-p$ cloud method (Duarte and Oden 1996); the Natural Element Method (Sukumar et al. 1998); the Meshless Local Petrov Galerkin (MLPG) by Atluri and Zhu (1998); the Maximum entropy (Arroyo and Ortiz 2006); the Particle Finite Element Method (PFEM) of Idelsohn et al. (2006), Sabel et al. (2014), the optimal transport meshfree method (OTM) of Li et al. (2010), just to name but the most popular meshfree methods.

In general, all meshless methods share the same characteristic: the domain of interest is completely discretized by nodes (or points or particles) as illustrated by Fig. 1.7. The concept of connectivity in mesh-based methods is replaced by domain of influence which indicates nodes fall within the support of a given node. A meshfree method is characterized by the following items

Collocation or Galerkin formulation DEM, EFG, RKPM, MLPG, NEM are Galerkin meshfree methods i.e., weak form based methods. Galerkin MMs are stable, accurate but computationally expensive. Collocation MMs are ones that approximate the strong form of a PDE (i.e., the PDE itself). One notable collocation MM is the SPH.[3] SPH is classified as a meshfree particle method or more

[2] The wikipedia page on meshfree methods lists about 30 methods and new methods are being created, https://en.wikipedia.org/wiki/Meshfree_methods.

[3] Note that since there are different SPH approximation rules, there thus exists different forms of discrete SPH equations. All of them are used in practice.

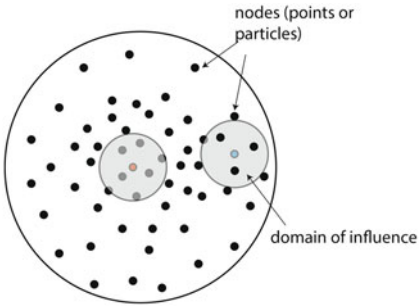


Fig. 1.7 Meshless discretization by a cloud of points

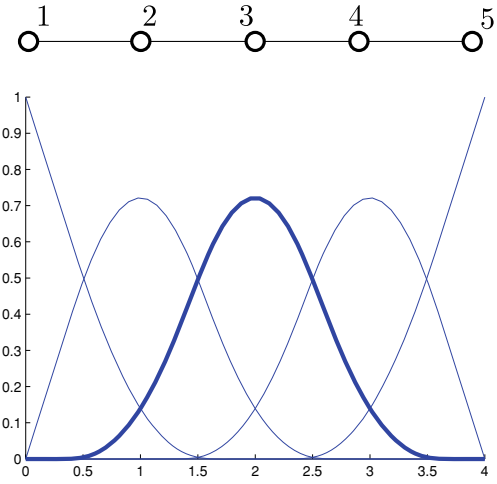


Fig. 1.8 Typical smooth meshfree basis functions: one dimensional MLS functions defined for a set of 5 nodes equally spaced. The highlighted function is the one of the middle node

precisely meshfree Lagrangian particle method because the particles are endowed with physical quantities such as density and volumes. In other words, interpolation points and particles coincide;

Interpolation method Different MMs adopt different high order interpolation techniques: EFG/DEM/MLPG utilizes the MLS (Moving Least Square), in RKPM and SPH kernel estimates are the interpolations. Generally meshfree functions/derivatives are smooth, cf. Fig. 1.8, but computationally expensive than FE functions;

Numerical integration For Galerkin meshfree methods a numerical integration scheme is needed of which one can mention background element/cell technique, nodal integration etc.;

Imposition of essential boundary conditions Most meshfree basis functions do not satisfy the Kronecker delta property thus making enforcement of essential boundary conditions a daunting task.

For a comprehensive consideration of MMs, we refer to various review articles, for instance Belytschko et al. (1996), Babuška et al. (2002), and Nguyen et al. (2008)

which provide computer implementation details including enrichment for weak and strong discontinuities, Hsieh and Pan (2014) for an essential software framework for MMs, Doblaré et al. (2005) (focused on the applications of meshfree methods in biomechanics) and the textbook of Liu and Liu (2003), Liu (2002), Li and Liu (2007), Fasshauer (2007). The last textbook gave a historical account of meshfree approximation theories such as moving least square, radial basis functions etc.

Some MMs have been incorporated into commercial FEA packages such as Abaqus (SPH), LS-Dyna (SPH and EFG), ANSYS (SPH). There exists also purely meshless packages such as NoGrid that implements the finite pointset method for applications in fluid dynamics.

1.5 A Brief Introduction to the MPM

The Material Point Method is one of the latest developments in particle-in-cell (PIC) methods. The first PIC technique was developed in the early 1950s by Harlow (1964), Harlow (2004) at Los Alamos National Laboratory and was used primarily in fluid mechanics. The first PICs suffered from excessive energy dissipation which was overcome in 1986, by Brackbill and Ruppel with the introduction of FLIP-the Fluid Implicit Particle method (Brackbill and Ruppel 1986; Brackbill et al. 1988). In computer graphics, PIC/FLIP has become the de facto standard method for fluid simulations (Zhu and Bridson 2005). The FLIP was later modified and tailored for applications in solid mechanics by Sulsky and her co-workers (Sulsky et al. 1994, 1995b) at University of New Mexico and has since been referred to as the Material Point Method (Sulsky and Schreyer 1996).

In FLIP, the strain and stresses are stored at the cell centers. Yet, in the MPM, they are carried by the particles themselves. Thus, the MPM particles carry the full physical state of the material including position, mass, velocity, volume, stress, temperature etc.. Note that in PIC, the particles carry only position and mass.

The MPM is built on the two main concepts already used in PIC that are the use of Lagrangian material points that carry physical information, and a background Eulerian grid used for the discretization of continuous fields (i.e., displacement field). For a short description of the Lagrangian and Eulerian descriptions, see Fig. 1.9.

1.5.1 Lagrangian Particles and Eulerian Grid

In the MPM, a continuum body is discretized by a finite set of n_p Lagrangian material points (or particles) that are tracked throughout the deformation process. The terms *particle* and *material point* will be used interchangeably throughout this book. In the original MPM, the subregions represented by the particles are not explicitly defined. Only their mass and volume are tracked. In advanced MPM formulations such as GIMP or CPDI, the shape of these subregions is tracked though. Each material point

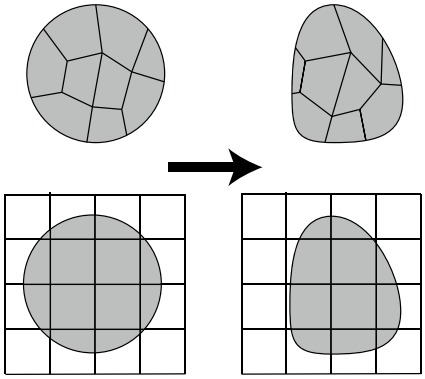


Fig. 1.9 Lagrangian description (top) versus Eulerian description (bottom). In a Lagrangian description, the grid is attached to the solid and thus it deforms during the deformation process of the solid. Each point in the grid is always associated to just one single material point, thus making modeling history-dependent materials easy. The solid boundary is also well defined. However, the grid can become distorted. On the other hand, the Eulerian grid is fixed in space and material flows through the mesh. Mesh distortion never happens

has an associated position $\mathbf{x}_p^t$ $(p = 1, 2, \ldots, n_p)$, mass m_p, density ρ_p, velocity $\mathbf{v}_p$, deformation gradient $\mathbf{F}_p$, Cauchy stress tensor $\boldsymbol{\sigma}_p$, temperature T_p, and any other internal state variables necessary for the constitutive model. Collectively, these material points provide a Lagrangian description of the continuum body. As each material point contains a fixed amount of mass at all time, mass conservation is automatically satisfied.

The original MPM developed by Sulsky is effectively an updated Lagrangian scheme. For this MPM, the space that the simulated body occupies and will occupy during deformation is discretized by a grid, called background grid where the equation of balance of momentum is solved. On the other hand, in the Total Lagrangian MPM (de Vaucorbeil et al. 2020), the background grid covers only the space occupied by the body in its reference configuration. We refer to Fig. 1.10 for a graphical illustration of material points overlaying on a Cartesian grid for both ULMPM and TLMPM. The grid is fixed and the particles are moving over it (Fig. 1.11).

The use of a grid has the following benefits. First, it allows the method to be quite scalable by eliminating the need for directly computing particle-particle interactions. Second, collision is treated easily through this background Eulerian grid (in fact, a non-slip, non-penetration contact is inherent in the method). Third, the momentum equation is solved on the grid, and as there are many fewer grid points than particles, this is a very efficient substitution. Most often, a fixed regular Cartesian grid is used throughout the simulation for efficiency reasons.

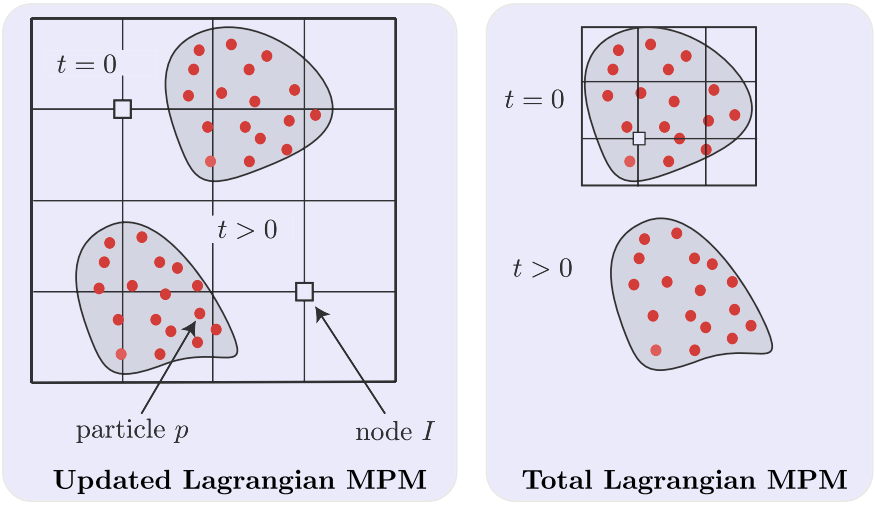


Fig. 1.10 The MPM discretization: the space is discretized by a background grid which can be either a Cartesian grid or an unstructured grid (not shown), while a solid is discretized using particles. The updated Lagrangian MPM grid covers the entire deformation space whereas the total Lagrangian MPM grid only covers the initial configuration

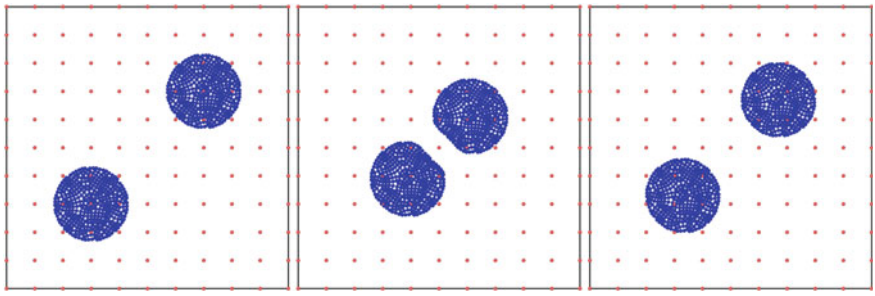


Fig. 1.11 The MPM discretization: the grid is fixed and over which the particles are moving

1.5.2 The Basic MPM Algorithm

The MPM was originally developed to solve fast transient impact solid mechanics problems (Sulsky et al. 1994). Therefore, the MPM has been developed using an explicit solver which is more efficient than an implicit solver for such problems. The method was then applied to other applications in which the loading rates are low. For these problems, implicit solvers are more suitable. As the explicit MPM algorithm is simpler than the implicit, in what follows, the updated Lagrangian MPM algorithm is presented using an explicit solver. Implicit MPM formulations are discussed in Remark 22. From the updated Lagrangian MPM, the total Lagrangian MPM is obtained by making only slight modifications.

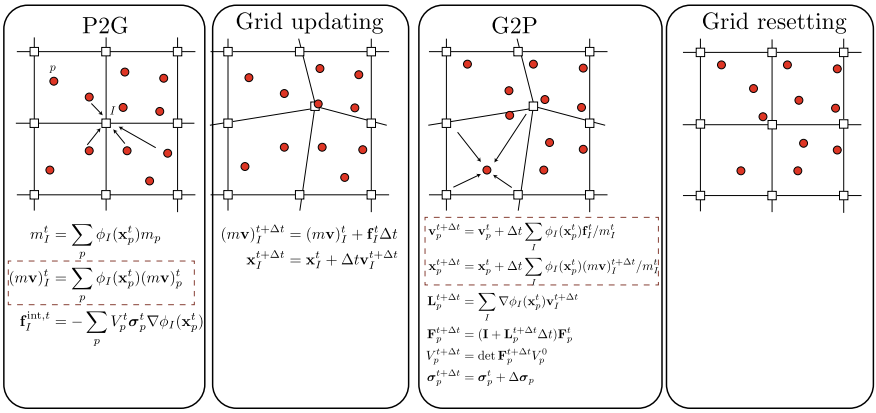


Fig. 1.12 Material point method: a computational step consists of four steps: (1) P2G (Particle to Grid) in which information is mapped from particles to nodes, (2) Grid Updating in which momentum equations are solved for the nodes, (3) G2P (Grid to Particles) where the updated nodes are then mapped back to the particles to update their positions and velocities and (4) Grid resetting where the grid is reset. The operations in dashed boxes are not present in the ULFEM

Table 1.1 Overall characteristics of common MPM variants

MPM variant	Efficiency	Quad. error	Cell cross.	Num. frac.	Grid type	Contacts
MPM	☺ ☺ ☺	☹ ☹ ☹	Yes	Yes	Cart./unstr.	☺
GIMP	☺ ☺	☹ ☹	No	Yes	Cartesian	☺
CPDI	☺ ☺	☹	No	No	Cartesian	☺
TLMPM	☺ ☺ ☺ ☺	☹	No	No	Cart./unstr.	☹
iMPM	☺	☹	No	n/a	n/a	n/a
GPIC	☺ ☺ ☺ ☺	☹	No	No	Cart./unstr.	☺

A typical explicit ULMPM computational cycle is given in Fig. 1.12. We refer to Table 1.1 for a list (not exhaustive) of notations and Table 1.3 for abbreviations. This algorithm is somewhat premature in the sense that some terms have not yet been precisely defined, but it is presented at this stage to give some perspective. The first step is mapping information from the particles to the grid (P2G) as the grid is reset at every cycle. Next, the discrete equations of momentum are solved on the grid nodes (Grid updating). Then, the particles' position, velocity, volume, density, deformation gradient, stresses and all relevant internal variables are updated (G2P). These last two steps are equivalent to updated Lagrangian FEM. Therefore, it is incorrect to state that the MPM uses a Eulerian kernel as in Gupta et al. (2011). Finally, the grid is reset to its original state. Due to this grid resetting mesh distortion never occurs making MPM a good method for large deformation problems. Note that the grid needs not to be reset at every time step. For instance, reset can be done every N time steps. N could be as large as one wants. The grid can also never be reset (Guilkey

et al. 2006). Moreover, a completely new grid can be used. But to the best of our knowledge, this is not yet implemented in any code.

It is quite difficult to precisely categorize the MPM due to the combination of Lagrangian particles and a background grid. In our view, as the MPM solves the momentum equations in their weak form, it can be seen as a Galerkin meshfree method, similar to EFG, the RKPM, the OTM. What differentiates the MPM from other Galerkin MMs is the ease with which the shape functions are constructed. Indeed, they are simple and efficient polynomials defined on a fixed Eulerian grid. Note that most of meshfree shape functions are computationally expensive rational functions defined on a cloud of nodes. When the grid is not fixed, the MPM is very similar to the OTM (or vice versa). The difference being that the OTM adopts the max-ent approximation (Iaconeta et al. 2017). Contrary to OTM, the MPM uses a background grid which if not fixed would generate mesh entanglement problems, similarly to updated Lagrangian FEM.

1.5.3 Advantages and Disadvantages of the MPM

Advantages of the MPM include:

- the absence of mesh-entanglement problems;
- error-free advection of material properties via the motion of the material points;
- a no-slip, no-penetration contact algorithm is automatic to the method. That is, it comes at no additional computational expense;
- a straightforward and efficient treatment of frictional contacts of multi-bodies thanks to the background Eulerian grid;
- suitability for image-based simulations, as it is easy to convert images into an MPM model;
- an easy computer implementation of the MPM compared to existing meshfree methods. The MPM algorithm is easily programmed for parallel, distributed-memory computers through decomposition of the computational domain;
- the leverage of existing well-studied FEM algorithms due to the similarity of the MPM with the FEM.
- suitability for problems with very complex geometries which are difficult to be converted into good quality FE meshes. In this regard, the MPM bears similarities with immersed boundary methods (Mittal and Iaccarino 2005; Schillinger et al. 2012) and its recent variants such as finite cell method (Parvizian et al. 2007), and cut FEM (Burman et al. 2015). All these methods, regardless the names, embed a solid of any shape into a cube (3D) which is larger than the solid. The cube is meshed by a regular structured grid equipped with smooth C^k basis functions be it B-splines or T-splines.

In addition to these advantages that the MPM offers, as with any numerical method, it has its own set of shortcomings:

- large memory footprint as the grid has to cover the entire region that the bodies occupy;
- formal analysis (convergence, error and stability) of the MPM is extremely difficult;
- enforcement of boundary conditions is difficult compared with the FEM;
- lower accuracy than the FEM as the material points do not generally lie at the optimal positions for numerical integration.

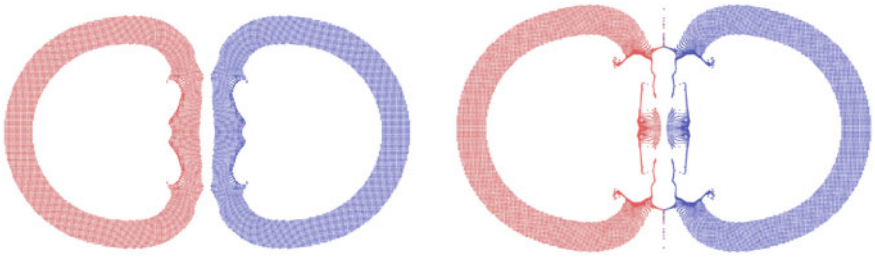
The first item applies only to the ULMPM not the TLMPM as in the later the grid covers only the initial undeformed configuration. The formal analysis of the MPM is extremely difficult due to the irregular distribution of the particles but also due to their relative motion with respect to the grid. If such an analysis is to be carried out, many assumptions are required to make the analysis manageable i.e., 1D, linear elastic materials, particles do not move from one cell to another (York et al. 1999). The difficult enforcement of boundary conditions is due to the lack of an explicit representation of the boundaries. But, it is still easier than some other methods (e.g. SPH) (Raymond et al. 2018). The generalized particle in cell (GPIC) is, however, free from this issue (Nguyen et al. 2021). Finally, the low accuracy only applies to small and moderately large deformation problems.

1.5.4 Existing MPM Formulations

The main differences between different MPM variants available in the literature emerge from the use of (i) different grid basis functions, (ii) different time integration schemes, (iii) different types of the grid (Cartesian or unstructured) and (iv) an updated Lagrangian or a total Lagrangian formulation. The discussion herein limits to the ULMPM.

Use of different grid basis functions. In the original MPM, dubbed herein as the standard/conventional MPM, the grid basis functions $\phi_I(\mathbf{x})$ are linear hat functions. Because these functions are only C^0, the standard MPM suffers from the so-called cell-crossing instability when particles cross the cell boundaries. In an attempt to remedy this issue, Bardenhagen and Kober (2004) introduced the generalized interpolation material point (GIMP) method. In GIMP, contrary to the original MPM, particles are not point like, but rather have finite extent, see Fig. 1.15. The resulting grid basis functions of GIMP are C^1 smooth functions which resemble B-splines functions used by Steffen et al. (2008b). The MPM with B-splines, named BSMPM, is now quite popular (Stomakhin et al. 2014a; Tielen et al. 2017; Gan et al. 2018).

Another issue related to particles moving from one cell to another is numerical fracture—unphysical separation of the solid (Fig. 1.13). This happens when two originally adjacent particles are separated by a distance high enough such that they



(a) before fracture (b) fracture

Fig. 1.13 Numerical fracture due to instability in the MPM. This is simulation of a collision of two rubber rings with the standard MPM (linear basis). No fracture model is included in the model

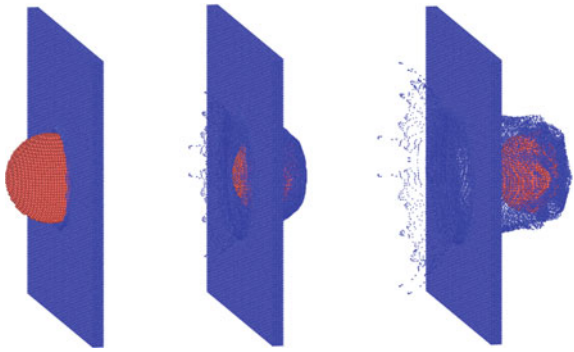
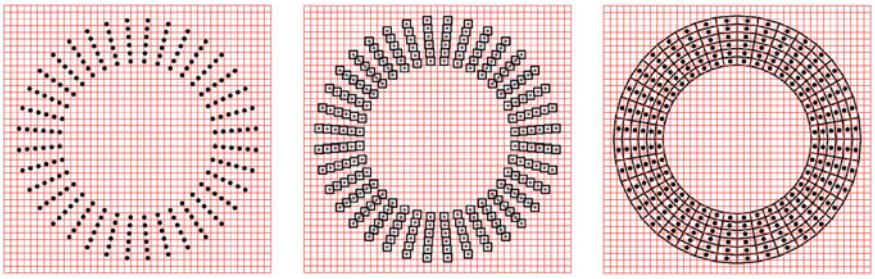


Fig. 1.14 Spherical projectile penetrates a steel plate: local Johnson-Cook damage model. ULMPM with linear weighting functions

no longer interact with each other. This distance depends on the type of the grid function used. Because of this numerical fracture, it is difficult to make sure the predicted fracture is physical or numerical, see Fig. 1.14.

Even though in GIMP, particle domains are tracked to some extent, gaps between them remain leading to low accuracy under arbitrary deformations. The Convected Particle Domain Interpolation (CPDI), the latest development in GIMP per our understanding, proposed in Sadeghirad et al. (2011), Sadeghirad et al. (2013) solves this problem. In CPDI, particles are modeled as quadrilaterals and tetrahedrons, in 2D and 3D, respectively. Therefore, the deformed solid is tiled without gap for arbitrary loadings, see Fig. 1.15c. CPDI is very good at handling extreme tensile deformations without exhibiting numerical fracture and is able to faithfully represent complex geometries. However, it is not exempt from issues. First, CPDI suffers from mesh-distortion problems that go against the spirit of the conventional MPM, see Wang et al. (2019). Second, parallelization is more complicated in CPDI (Homel et al. 2016) as particles might have a domain of influence large enough to span over many different CPU domains.

One of the latest developments in the MPM is the Generalized Particle in Cell of Nguyen et al. (2021). It extends the CPDI to any element types and background grid and it does not suffer from the mesh distortion of CPDI as it adopts a Total



(a) MPM (b) GIMP (c) CPDI

Fig. 1.15 Particles in the standard MPM (MPM with hat and B-splines functions), GIMP and CPDI

Lagrangian formulation for the calculation of forces. We present some applications of GPIC in Fig. 1.16.

Use of different time integration schemes. The most common implementation of the MPM uses an explicit backward Euler integration scheme. Other explicit integrations schemes have also been used with varying success. Sulsky et al. (2007) adopted a staggered central difference integration scheme for the simulation of sea ice dynamics which was later compared with the most common implementation by Wallstedt and Guilkey (2008). Their conclusions are that there is only small differences between these two schemes.

Implicit MPM also exists. Explicit time integration schemes are easy to implement, efficient and stable for short-duration dynamic problems. However, they are required to adhere to the Courant stability condition $dt < dx/c$ where dx is the grid spacing and c is the speed of sound in the material. For low strain rate and quasi-static problems e.g. manufacturing problems like metal rolling, upsetting, and machining, this condition is severe and it is more efficient to use an implicit time integration scheme then. In an implicit MPM, one can either form explicitly the Jacobian matrix or adopt a matrix-free solver. In the following, we review the implicit MPM for both dynamics and quasi-static problems.

The first to use an implicit scheme for the MPM were Cummins and Brackbill (2002). They applied it to the simulation of quasi-static loading of granular materials involving frictional contacts. Implicit integration schemes are known to be computationally expensive. Therefore, to reduce computational time (by avoiding the construction of the tangent matrix), they adopted the matrix-free Newton-Krylov algorithm. Sulsky and Kaul (2004) reported a similar method and Love and Sulsky (2006b), Love and Sulsky (2006a) extended it for hyperelastic-plastic materials. These papers adopted a three-field formulation to avoid volumetric locking (Sect. 9.6 provides a discussion on volumetric locking). Conservation of linear and angular momentum of the MPM was discussed in detail in these references. Nair and Roy (2012) implemented this matrix-free implicit dynamics for GIMP. This matrix-free approach was extended to quasi-static problems in Sanchez et al. (2015) where it

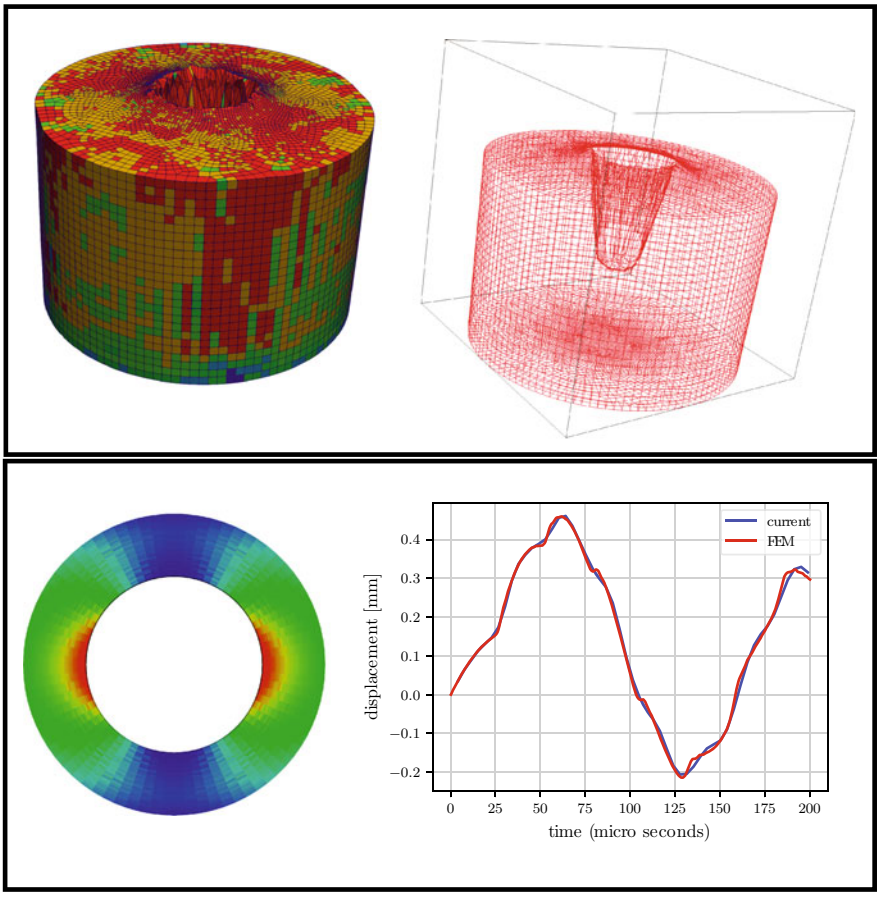


Fig. 1.16 Generalized particle in cell (GPIC) method: TL finite element meshes embedded in an Eulerian grid. GPIC solves many problems of a particle-based MPM: (i) better representation of solid boundaries, (ii) seamless enforcement of Dirichlet and Neumann boundary conditions, (iii) seamless treatment of material interfaces, (iii) higher efficiency and (iv) better stress fields (Nguyen et al. 2021)

showcased the advantage that a consistent material tangent, which is usually hard to obtain for complex constitutive models, is not required.

In contrast, Guilkey and Weiss (2003), Wang et al. (2016) explicitly formed the tangent stiffness matrix and used the Newton-Raphson method together with the well known Newmark integration scheme to solve the equilibrium equations in time. They reported that time steps hundreds of times larger than those used in explicit MPMs. Moreover, the use of a consistent tangent allows time steps to be much larger than those for the matrix-free formulation of Cummins and Brackbill (2002), Sulsky and Kaul (2004). We refer to Iaconeta et al. (2017) for a detailed presentation of the algorithm of Guilkey and Weiss (2003).

Cartesian grid versus unstructured grid. In the MPM a uniform Cartesian grid is usually used. This eliminates the need for computationally expensive neighborhood searches during particle-mesh interaction (i.e., to which nodes a particle maps its data and vice versa). Furthermore, it is easier to develop smooth C^k ($k \geq 1$) basis functions with a Cartesian than with an unstructured grid.

On the other hand, in the geo-technical engineering community, unstructured grids are usually adopted. Wieçkowski et al. (1999), Wieçkowski (2004) were the first to use an unstructured grid for silo discharging applications. Since then, unstructured meshes have been used in later works of related research groups in University of Stuttgart, Germany and Delft University of Technology, The Netherlands e.g. Beuth et al. (2011), Jassim et al. (2013). In contrast to a Cartesian grid, the search for which element contains a given particle in an unstructured mesh is not trivial and is very time-consuming. However, the use of an unstructured grid facilitates the enforcement of complex boundary conditions (i.e., boundary conditions on curved surfaces).

In a line of research parallel to that of GIMP and CPDI, Zhang et al. (2011) proposed the dual domain MPM (DDMPM) for unstructured grids. DDMPM was motivated by the difficulty to develop C^1 functions for an unstructured grid to mitigate the cell-crossing issue. The basic idea is to map the particle stresses to the grid nodes and then interpolate them to obtain a continuous stress field at any point of the domain. This dual mapping process makes a smoother gradient emerge. More recently de Koster et al. (2019) developed C^1 basis functions over unstructured grids using Powell-Sabin functions (Powell and Sabin 1977). We refer to Sect. 5.4 for a presentation of the MPM using unstructured grids.

High order MPMs. Even though impressive simulations have been done with the MPM and its previously presented variants, rigorous analyses of the method for simple problems show that the convergence rate is poor (rarely of second order) as explained further in Chap. 9. Aiming to improve the order of convergence, Wallstedt and Guilkey (2011) presented a weighted least square MPM for solids and Edwards and Bridson (2012) proposed a moving least square (MLS) MPM for fluids. An improved MPM (iMPM) was presented by Sulsky and Gong (2016). In the iMPM, the hat functions are used for all the mapping except the velocity mapping from particle to node for which MLS is used. Additionally, one point quadrature is used i.e., the quadrature points are the cell centers. This removes cell-crossing instability. They demonstrated second-order convergence for iMPM, but only for moderately fined meshes. No convergence was observed for very fine meshes. Similar ideas can be found in Wobbes et al. (2019). This high order of convergence were only demonstrated for 1D problems. Very recently, Liang et al. (2019) also use one-point quadrature but the cell center data are reconstructed, not via MLS, but with an extra staggered grid. A presentation of the iMPM is given in Chap. 9.

Another stability issue of MPM is the so-called null space issue—the mapping of non-zero particle values can result in zero nodal values. This problem arises from the difference between the number of particles and the number of the nodal grid points. Null space issue might be the culprit to the non convergence of iMPMs when

used with very fine meshes. Methods to solve this issue are presented in Gritton and Berzins (2017), Tran and Sołowski (2019).

A table presenting the characteristics of common MPM variants is given in Table 1.1. Note that even though the iMPM was applied to the standard MPM only, it can be used with other MPM formulations.

1.5.5 Multiphysics MPM

Although the MPM was originally developed for mechanical problems, it also has been extended to the simulation of multi-physics problems. Such simulations involve solving a coupled system of more than one partial differential equations. For example, for thermo-mechanical processes, the equation of motion is coupled with the energy balance equation (or heat diffusion equation) as presented in Chen et al. (2008), Nairn and Guilkey (2015), Fagan et al. (2016), Tao et al. (2016), Gritton et al. (2017), Tao et al. (2018), Leroch et al. (2018). They show that when the explicit MPM solver is used, incorporating the heat diffusion is straightforward using an staggered solver. Among these works, Fagan et al. (2016) demonstrated that the MPM is able to simulate friction stir welding (FSW). FSW is a recent and complicated thermo-mechanical process used to join different materials which is still not well understood. And Gritton et al. (2017) reported first coupled chemical/mechanical MPM simulations of the deformation of a silicon anode. As these are important developments in the MPM, the algorithms used to solve thermo-mechanical coupling is presented in Sect. 10.3.

1.5.6 Contacts

A contact happens when two deformable solids touch each other. It is a key element to understand many engineering problems such as pile-soil interaction, wear, metal forming processes, etc.. The simulation of contacts remains challenging and various algorithms (e.g. note-to-segment contact, segment-to-segment contact, penalty and Lagrange multiplier methods) have been developed (Benson 1992). However, it is much easier in the MPM due to the use of a background grid. Indeed, it was developed mainly to handle contact problems.

A no-slip no-penetration contact is inherent in the MPM i.e., contact of solids is automatically handled without any extra numerical treatment. This is due to the use of a single-valued velocity field for updating the positions of the material points. This automatic no-slip contact ability of the MPM has been used in many works that involve complex contacts. For example, Bardenhagen et al. (2005), Brydon et al. (2005) used GIMP to simulate the compression of foam microstructures to full densification. This is a challenging problem which involves the combination of discretizing complex microstructures, simulating large deformations and multiple contacts. This work demonstrated that particle methods are well suited for the

simulations of solids with complex geometries because body-fitted meshes are not needed. Nairn (2006)[4] applied the MPM to study the transverse compression and densification of wood. Liu et al. (2015a) studies honeycomb sandwich panel subjected to high-velocity impact. And more recently Sinaie et al. (2019) simulated the large deformation response of cellular structures which involve many contacts.

In contrast, frictional contact between different bodies or self contact within a single solid requires a modification of the standard MPM algorithm.

Multi-body frictional contacts. The first Coulomb frictional contact algorithm in the MPM was probably presented by Wieçkowski et al. (1999) to model contact between a deformable body and a rigid wall in silo discharging simulations. At the same time, York et al. (1999), York et al. (2000) observed that, in MPM simulations, two bodies sometimes "stick" to one another unphysically when they should separate. To alleviate this problem, they proposed the first contact algorithm for the MPM that allows the contacting bodies to release from one another. But the contact is still no-slip. Bardenhagen et al. (2000), Bardenhagen et al. (2001) extended York's contact algorithm to frictional contact and used it to model interaction between grains in granular materials. Bardenhagen's algorithm, known as *multi-material* or *multi-body* contact, is very efficient as it is linear in the number of grains and allows separation, sliding and rolling. The algorithm's basic idea is to modify the velocities of contact nodes (i.e., those receive contributions from more than one material/body) to account for the collision. This algorithm is very popular. For instance, Nairn (2013) used it with modifications to model imperfect (debonding) interfaces. An improved version of Bardenhagen's algorithm was given in Lemiale et al. (2010) to model a metal forming process called the equal channel angular pressing technique. This work emphasized on the treatment of contact at interfaces between rigid and deformable bodies. All prior applications have been limited to Coulomb friction. Most recently, Nairn et al. (2018) generalizes the MPM approach for contact to handle any friction law with examples given for friction with adhesion or with a velocity-dependent coefficient of friction. The Bardenhagen's algorithm will be presented in Sect. 8.1.

In the field of geotechnical engineering, a first contact algorithm for the quasi-static MPM was presented in Beuth (2012). They adapted zero-thickness interface elements, commonly used in FEM, to model contact between soil and rigid surfaces. This technique applies only to the case where the contact surface is known prior the start of the simulation. Bardenhagen's algorithm was modified in Al-Kafaji (2013) for adhesive contact using an explicit dynamics MPM code. Yet another modification of Bardenhagen's contact model suitable for geotechnical engineering (that involves contact between stiff structural elements and soil) was presented in Ma et al. (2014).

As most of MPM variants provide a noisy stress field, de Vaucorbeil and Nguyen (2021b) presented a contact model for the TLMPM. The method produces smoother stress fields which are crucial for e.g. damage/fracture problems. We illustrate this in

[4] This work received the George Marra Award for paper of the year from the Society of Wood Science and Technology.

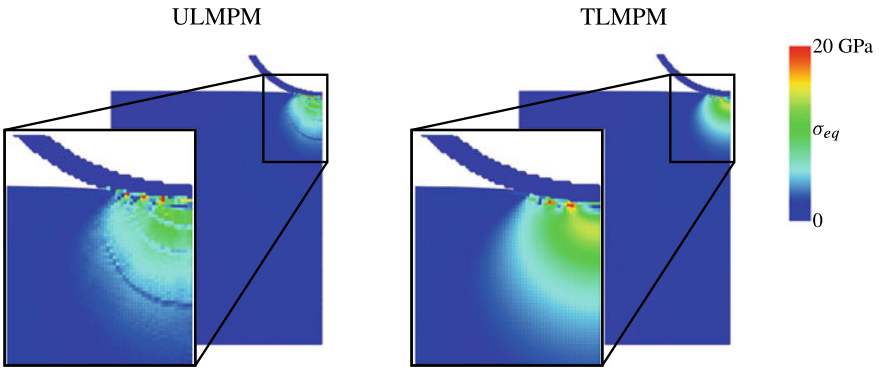


Fig. 1.17 Comparison of the equivalent stress field due to the contact between a rigid indenter and an elastic half plane using ULMPM and TLMPM. These results were obtained using linear shape functions and one particle per cell with a cellsize of 0.0025 mm mm when the indentation force equals 120 N. One can clearly see that with ULMPM, the stress is not smooth (de Vaucorbeil and Nguyen 2021b)

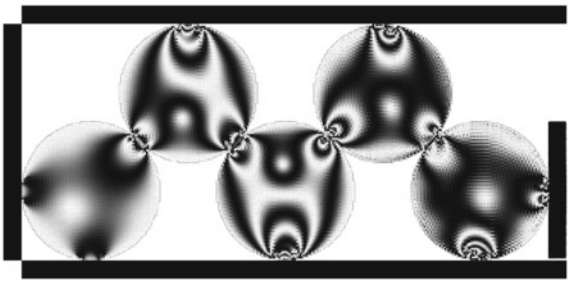


Fig. 1.18 Stress wave in a granular material using the TLMPM (de Vaucorbeil and Nguyen 2021b)

Fig. 1.17 for a contact between a sphere and a rigid plane. And Fig. 1.18 for multiple contacts. Liu and Sun (2020) presented an implicit MPM that adopts the level set method to represent the solid boundary and the iMPM to have better quadrature accuracy.

Frictional self contact. Homel and Herbold (2017) was the first to propose an algorithm for frictional self contact. Its basic idea is to automatically and dynamically detect contact nodes. This is achieved with a scalar field (actually its gradient) that defines potential contact surfaces. The same idea (named DFG) was applied to fracture and resulted in a first model for fracture and post-fracture frictional contacts i.e., contacts between crack faces.

1.5.7 Fracture

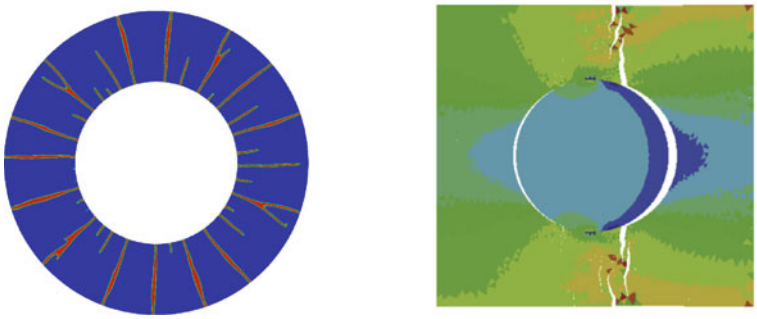
Fracture is defined as the separation of a solid into two or many pieces. This phenomenon has been extensively studied since the pioneering works of Griffith (1920), Irwin (1957). They developed the field of fracture mechanics which has found tremendous applications in many engineering disciplines notably aerospace engineering.

Modeling the initiation and propagation of cracks in a solid is a challenging problem as one needs to track a set of evolving internal surfaces across which the displacement field is discontinuous. Basically, there are three approaches to modeling fracture: discontinuous, continuous and mixed continuous-discontinuous approaches (Rabczuk 2013). We also refer to the interesting article of Boyce et al. (2016) that presents an analysis of the predictability of a number of different techniques in simulating ductile fracture.

In the discontinuous approach, the cracks are explicitly represented (following a fracture mechanics theory). In the continuous approach, a crack is, however, not explicitly modeled (adopting a damage mechanics theory (Kachanov 1958; Lemaitre and Chaboche 1994)). In the later approach, the presence of cracks is treated via damage variables used to degrade the stresses. Figure 1.19 presents fracture simulations using these two methods.

There exists yet another approach–the combined continuous-discontinuous approach. The combined continuous-discontinuous approach to fracture combines the two approaches, as its name implies. It was developed mainly to overcome the shortcoming of continuous fracture models that a true material separation cannot be captured.

Discontinuous approach. In the MPM, explicit cracks have been modeled as strong discontinuities by allowing *multiple velocity fields* at nodes whose supports are cut



(a) Continuous approach (b) Discontinuous approach

Fig. 1.19 Continuous versus discontinuous approaches to fracture modeling. In **a**, fracture of a cylinder under internal pressure impulse is given. The red color denotes the damage variable close to one which corresponds to failed elements (Mandal et al. 2020a). In **b**, cracking of a fiber-reinforced composite material is shown (Nguyen 2014)

by the cracks. This is similar to duplicated nodes in the FEM. The crack can be modeled using either the LEFM (Linear Elastic Fracture Mechanics) (Nairn 2003; Tan and Nairn 2002; Nairn 2007a; Guo and Nairn 2004; Gilabert et al. 2011; Wang et al. 2005) or cohesive zone models (Daphalapurkar et al. 2007; Nairn 2007b) or a combination of both (Nairn 2009; Bardenhagen et al. 2011). Explicit fracture simulation is computationally expensive (as one needs to track evolving surfaces) and prohibitive for large scale simulations. Furthermore, the implementation is intricate particularly for complex crack patterns. The latest development in this direction is the work of Moutsanidis et al. (2019a) that introduces a single velocity field for cracks by modifying the grid basis functions.

Remark 5 Although a majority of these works involve simulations involving small deformation, we don't see any limitations for the discontinuous approach to be used for large deformations. We do not think that the MPM is better than the FEM for small (or moderately large) deformation fracture mechanics problems. This observation is backed up by the outcome of the first Sandia fracture challenge (Boyce et al. 2014) where the MPM was used by one team (Yang et al. 2014) and the MPM was not giving the best solution.

Continuous approach. In the MPM, fracture has been modeled using various continuous methods. They all share one common feature: no need to explicitly represent the crack surfaces as in the discontinuous approach. For penetration problems, fracture is usually treated using a strain-based failure criterion and particle erosion. That is, a particle is set to be failed when it satisfies a certain strain-based fracture condition. When that happens, its deviatoric stress is set to zero, but it remains part of the simulation, and its mass contributes to the overall inertia of the material. Thus mass conservation is enforced. Strain-based failure combined with particle erosion was also used in Ionescu et al. (2006) to study failure of soft tissues penetrated by a low-velocity projectile. They reported convergent results for fine grids, but no quantitative evidence was provided. An elastic-plastic material with particle erosion was employed in Huang et al. (2011) to model penetration of thin steel plates and perforation of thick aluminum plates. Sensitivity of the results with respect to grid size and number of particles were studied, and some quantities showed convergence. A constitutive model belongs to this category is given in Sect. 4.3.

For low strain rate and quasi-static problems, standard constitutive models are often used. For example, a softening Mohr-Coulomb plasticity model was used in Alonso and Zabala (2011) to model the failure of the Aznalcóllar dam. More recently, a softening Drucker-Prager plasticity coupled to the Grady Kipp damage model was presented in Raymond et al. (2019) to model failure of aggregate materials. They reported mesh-convergent results even though no special treatment was done to deal with softening. Homel and Herbold (2017) adopted a Rankine damage model with linear strain-softening.

Also for static and low strain rate dynamic problems, Schreyer et al. (2002), Sulsky and Schreyer (2004) used a smeared crack model (they referred it as a decohesion

model), see e.g. Rots et al. (1985), Rots (1991). Chen et al. (2002) improved this model by using a strain-based damage diffusion equation combined with a tensile damage model. Similar works include Shen and Chen (2005), Shen (2009), Yang et al. (2012), Yang et al. (2014). Implementation details of this decohesion model in the MPM are given only recently in Sanchez (2011) and he demonstrated the results are mesh biased. That is, accurate solutions are achieved only for cases in which the crack orientation coincides with the orientation of the grid cell lines. Sanchez went even further to comment that research on smeared crack models in the MPM would be a waste of time.

In the original damage mechanics framework, damage is determined using local variables such as stress and strains. This has proven to create mesh-dependent softening. To alleviate this issue, non-local models can be used. By non-local, we mean that damage is determined using non-local variables such as averaged stress and strains in the vicinity. The idea is to introduce a length scale in the equation. Burghardt et al. (2012) were the first ones to implement a non-local plastic model in the MPM.

Non-local damage can also be simulated using a phase field variational damage/fracture (PFF) theory see e.g. Bourdin et al. (2008), Miehe et al. (2010), Wu (2017), Wu and Nguyen (2018) and the most recent review of Wu et al. (2019). PFF was implemented in an MPM code for the first time by Kakouris and Triantafyllou (2017a, b). Unfortunately only problems involving small deformation without contact were demonstrated then. Recently, Cheon and Kim (2019) also presents a similar method. They use adaptive grid refinement and particle splitting to capture the gradients of the phase field. In the computer graphics community, a similar idea has been recently proposed in Wolper et al. (2019) with impressive fracture simulations typically seen in this community. Another variational fracture model, the eigen-erosion model of Schmidt et al. (2009) was used by Zhang et al. (2020) to model small deformation brittle fracture.

Figure 1.20 shows an application of the TLMPM coupled with a nonlocal Johnson-Cook damage model to simulate ductile fracture due to bullet penetration into a Weldox steel plate.

Continuous-discontinuous approach. The first continuous-discontinuous brittle fracture model in the MPM (precisely CPDI) is presented in Homel and Herbold (2017). In this model, fracture is first treated with a continuum damage model and then a self-contact algorithm is applied to the cracked nodes to separate the materials on the two sides of the crack. The damage model is enhanced with random material properties to mitigate mesh bias and the fracture energy is scaled with the grid size. Post-failure contact between the faces of a crack is allowed using the DFG algorithm. Recently, Homel and Herbold (2018) applied their model for mesoscale simulations of porous materials.

Remark 6 With the development of variational approaches to fracture (or phase-field fracture models), the modeling of the initiation and propagation of complex crack networks in solids on a fixed mesh with conventional continuum finite elements can be done. Therefore, the statement that meshfree methods are better suitable for

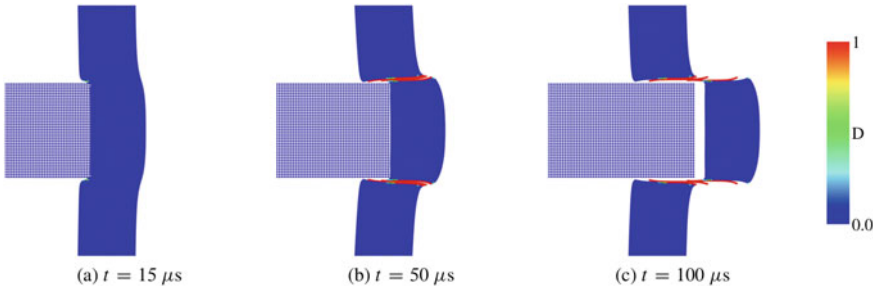


Fig. 1.20 Time evolution of impact between the projectile and the target for an initial velocity of 250 m/s (with particle erosion). The colors show the amount of damage present in each particles. For ease of visualisation, only a small part of the simulated domain focused around the impact area is shown (de Vaucorbeil et al. 2022b)

fracture seems no longer convincing. Nevertheless, it is very attractive to have a method that can handle large deformation, contact, fracture and post-failure contacts between fragments. The MPM might be such a method even though more works must be done before this goal is achieved.

1.5.8 Fluids and Gases

Although the MPM was developed for applications in solid mechanics and its main application area is still solid mechanics, it has also been applied to fluid mechanics problems. This probably arises from the need to simulate fluid-structure interactions (FSI) using just the MPM—to avoid coupling the MPM solid solver with a fluid solver which is not an easy job. Herein, we only discuss MPM algorithms for fluids. For works where the MPM used for solids is coupled to a fluid solver, we refer to Sect. 1.6.2.

There are basically two ways to handle fluids (and gases) using the MPM. The first is the so-called weakly compressible MPM. This way is almost identical to the solid MPM solver except that a constitutive model for fluids is used. This is done by describing the relation between the fluid pressure and density using an artificial equation of state (EOS). This fluid MPM solver was first presented in York et al. (1999), and it has been used for gas dynamics problems in York et al. (2000), Tran et al. (2010), Ma et al. (2009b). It has also been used for fluid flow problems (Li et al. 2014; Mast et al. 2012). Using this fluid MPM solver, comparative studies of SPH and the MPM for fluid mechanics are presented in Zhao et al. (2017), Vargas et al. (2018), Sun et al. (2018). Overall, it has been shown that both MPM and SPH predictions are quite similar but the MPM computational time is smaller. This weakly compressible fluid/gas MPM solver is presented in Sect. 10.1.

There are two main issues with the weakly compressible fluid MPM. First, if using an explicit time solver, very small time step is needed owing to the need to use a very

large bulk modulus. Second, pressure oscillation occurs. To solve these problems, the second way is the truly incompressible MPM. Stomakhin et al. (2014b) in the computer graphics community were the first to present this formulation. They used the Chorin's operator splitting method (Chorin 1968). In the engineering community, Zhang et al. (2017), Kularathna and Soga (2017) presented similar formulations.

1.5.9 The MPM Versus Other Methods

Which meshfree method should I use? This is the constant question facing every person new to MMs every time they need to solve a large deformation problem, if they are not forced to use any particular method. It is rather certain that no one is able to answer this question and neither are we. Nevertheless, to these beginners, herein we discuss the comparison between the FEM, SPH, the discrete element method (DEM), Galerkin MMs, immersed boundarymethods (Mittal and Iaccarino 2005; Schillinger et al. 2012), and the MPM based on the literature. By based on the literature, we meant that this section is merely based on results reported by other researchers. We synthesize their result and give a picture of common MMs.

SPH is probably the most popular meshfree method as it has been used in engineering and computer graphics and incorporated into many commercial softwares such as AUTODYN, PAM-CRASH, LS-DYNA and ABAQUS. Even though not a continuum-based numerical method, the ability to deal with large deformation and fracture makes the DEM a very popular technique particularly in geo-technical engineering (Cundall and Strack 1979; Scholtès and Donzé 2012; Sinaie et al. 2018a). One can attach the popularity of these two methods to their simplicity in implementation and they work in 2D and 3D.

The first comparative study of SPH and the MPM for hypervelocity impact problems was conducted by Ma et al. (2009a), Ma et al. (2009b). For the simulations considered, they showed that the MPM is faster and more accurate than SPH (precisely SPH in LS-DYNA). This can be explained by the fact that, the critical time step size in the MPM depends on the cell size of the background grid, rather than the particle space as in SPH, so that the time step used in the MPM is much larger than that of SPH. Furthermore, there is no neighboring particles search which is very time consuming. Note, however, that this conclusion is rather superficial as it was based on only a few numerical tests. It is certain that a working MPM simulation requires less *tricky ad hoc* parameters than SPH. A typical MPM simulation requires only data such as mesh spacing and time steps in the same manner with the FEM.

Interestingly, explicit MPM is faster than LS-DYNA explicit FEM for the well known Taylor impact problem, see Ma et al. (2009b). This superior efficiency was also reported in Leroch et al. (2018) for micro-milling simulations. The reason is the same: the critical time step size in the MPM is very much larger than the one in the FEM as the FEM mesh deforms and the element sizes become very small.

Within the context of implicit dynamics, Iaconeta et al. (2017) carried out a comparative study of the standard MPM and the so-called Galerkin meshfree method

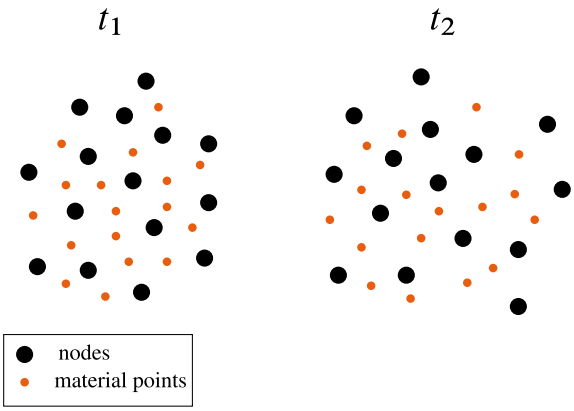


Fig. 1.21 The optimal transport meshfree (OTM) method: beside the optimal transport theory which is new to the computational solid mechanics community, the OTM method employs classic concepts for the discretization: nodes and material points. The material points play the role of integration points and move in the velocity field of the nodes. As there is no grid, nodal shape functions are more involved. There is no mapping from material points to the grid as in the MPM

(GMM) proposed in Espluga (2014). This GMM is very similar to the OTM of Li et al. (2010), except that a forward Euler is used for time integration (Fig. 1.21). The authors find that the GMM using the max-ent and MLS lack robustness for very large deformation problems. This lack of robustness is due to the meshfree nature of these shape functions i.e., one has to ensure that there are always sufficient nodes surrounding a given material point. A stabilized OTM was presented in Weißenfels and Wriggers (2018). Wobbes et al. (2020) presented a comparative study of the MPM and the OTM where they found that despite being quite different, these two methods bear similarities.

Remark 7 Recent works combine the MPM and SPH. For example, Raymond et al. (2018) coupled SPH to an MPM. They used SPH in the bulk and MPM around the surfaces for an easy enforcement of boundary conditions. He et al. (2019) developed the so-called SMPM where the particle velocities and stresses are smoothed using the SPH functions. The SMPM is shown, for impact simulations, to perform better than the traditional MPM and SPH and more efficient than the latter. See also He and Chen (2019) for an application of the SMPM to strain localization.

For granular flow, some studies on the performance of the DEM and MPM have been reported in Coetzee (2003), Coetzee et al. (2007), Ceccato et al. (2018), Gracia et al. (2019). Basically, the MPM can capture the DEM accuracy if a proper constitutive model is adopted. One might argue that MPM should be used for large scale simulations using constitutive models derived from DEM simulations. In Dunatunga and Kamrin (2015), a constitutive framework for granular media was presented that is able to simulate in one setting a wide range of granular behaviors spanning several phases: solid-like static behavior, plastic flow (up to very large strains), as well as

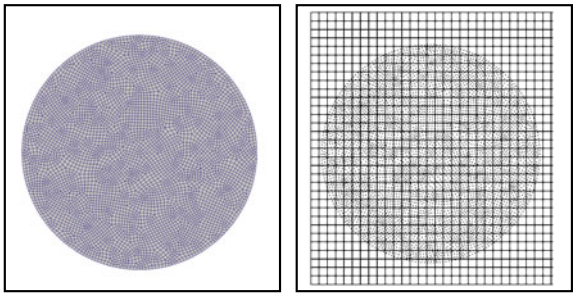


Fig. 1.22 Boundary conforming methods (left) versus immersed boundary methods (right)

separation and re-consolidation of the material. A recent work of Jiang et al. (2019) presented a combined MPM-DEM method.

Although the background Eulerian grid originally used in the MPM is to handle collision, recent trends of using the finite element quadrature, see Liu and Sun (2020), for evaluating the weak form integrals (this trend started from the work of Sulsky and Gong (2016) with the iMPM as discussed previously) make the MPM similar to immersed boundary methods or vice versa. In immersed boundary methods, the solid under consideration is immersed in a grid which is larger than it and does not conform to the solid boundary, see Fig. 1.22. Some popular immersed boundary methods are the finite cell method (Parvizian et al. 2007), and cut FEM (Burman et al. 2015). The tricky part of these methods is the quadrature for the cells cut by the solid boundary. Recently, Liu and Sun (2020) used material points to represent the solid domain, level sets to capture the boundary and information of material points are mapped to the standard quadrature points for numerical integration.

1.5.10 Coupling the MPM with Other Methods

As no method is without shortcomings, it is natural to want to couple two (or even more than two) methods together to take advantages of their best features. Herein, we review works that present the coupling of the MPM with other numerical methods be it the FEM or molecular dynamics. Section 1.6.2 discusses the coupling of the MPM with other methods for fluid-structure interaction problems.

It is reasonable to employ the efficient and accurate FEM in regions where the deformation is moderate and a particle method such as SPH or the MPM in domains featuring high deformation. Zhang et al. (2006) developed an explicit material point finite element method (MPFEM) to take advantages of both FEM and the MPM. The basic ideas is that the solid is discretized by a mesh of finite elements, and a computational grid is additionally predefined in the potential large deformation zone. The nodes covered by the grid are treated as MPM particles, and the remaining nodes are treated as FE nodes. The MPFEM was improved later by Lian et al. (2011a), Lian et al. (2012), Chen et al. (2015). These algorithms are not presented here, and we

refer to the textbook of Zhang et al. (2016b) for details. Results reported in these papers show that the coupled MPFEM is indeed faster than the pure MPM.

The MPM has been coupled with molecular dynamics to achieve a multiscale atomistic-to-continuum method (Ayton et al. 2001; Lu et al. 2006; Chen et al. 2012; Liu et al. 2013). It is the particle nature of MPM that makes the coupling straightforward. As we lack experiences in this area, our discussion is merely for completeness. We refer the reader to the textbook of Zhang et al. (2016b) for details.

1.6 Applications of the MPM

The MPM has been applied to a wide range of problems involving solids, fluids and gases undergoing very large deformation. This section attempts to present an extensive overview of MPM applications. The idea is to give an outline of the types of simulation that can be done with the MPM. By no means it is exhaustive. Therefore, we apologize to those authors whose works were not mentioned here due to our negligence.

This section is organized as follows. Sections 1.6.1 presents works done in the geo-technical engineering field. Fluid-structure interactions are given in Sects. 1.6.2. Image-based simulations are discussed in Sects. 1.6.3. Sections 1.6.4 covers the works done by computer graphics researchers. Finally, Sects. 1.6.5 summarizes important works from other fields. As contact and fracture problems were detailed earlier (see Sects. 1.5.6 and 1.5.7), they are not covered specifically in this section.

1.6.1 Large Strain Geo-Technical Engineering

The MPM has been adopted in large strain geo-technical engineering problems including landslide (Andersen and Andersen 2010b; Llano-Serna et al. 2016; Soga et al. 2015; Yerro et al. 2018), silo discharging (Wieçkowski et al. 1999; Wieçkowski 2004; Mühlhaus et al. 2001), anchor pull-out (Coetzee et al. 2005), excavator bucket filling (Coetzee et al. 2007), pile driving (Lim et al. 2013; Nguyen et al. 2016; Galavi et al. 2017), problem of subsidence of landfill covers (Zhou et al. 1999), shaped-charge jet penetration in wellbore completion (Burghardt et al. 2010; Homel et al. 2015) and installation of geosynthetics materials (Hamad et al. 2015). The MPM was also used to model large deformation of saturated porous media, see for instance Zhang et al. (2009), Beuth et al. (2011), Jassim et al. (2013), Zheng et al. (2013), Abe et al. (2014), Ma et al. (2014), Yerro et al. (2015), Pinyol et al. (2017). More recently, the MPM has been successfully used in the field of terramechanics (Agarwal et al. 2019) to simulate the interaction of rigid wheels with dry granular media. We refer to the book of Fern et al. (2019) for a more complete coverage of MPM applications in geo-engineering.

All the above works adopted the conventional theory of continuum mechanics to the exception of Mühlhaus et al. (2001) who used the Cosserat continuum framework in the MPM. The Cosserat theory is different as it also involves rotational velocities and the balance of angular momentum results in a non-symmetrical Cauchy stress tensor.

Remark 8 People who are familiar with FLAC (Fast Lagrangian Analysis of Continua)[5] developed their own MPM formulations. For example Konagai and Johansson (2001) made the Lagrangian Particle finite difference method (LPFDM) for geomechanics. Differences with the standard MPM lie in the fact that stresses are smeared over the background grid i.e., internal forces are computed using the averaged stress tensor[6] stored at the element center.

1.6.2 Fluid-Structure Interaction

Fluid-structure interaction (FSI) is the interaction of a rigid or deformable structure with an internal or surrounding fluid flow. Fluid-structure interactions are a crucial consideration in the design of many engineering systems, e.g. aircraft, spacecraft, engines and bridges. Two main approaches exist for the simulation of fluid-structure interaction problems: (1) the monolithic approach where the equations governing the flow and the displacement of the structure are solved simultaneously—with a single solver—and (2) the partitioned approach where the equations governing the flow and the displacement of the structure are solved separately—with two distinct solvers. We refer to Bazilevs and Takizawa (2017) for a comprehensive account on this very challenging topic. The following of this section reviews works on FSI using the MPM.

Monolithic FSI MPM. The first FSI using the MPM was presented in York et al. (1999), York et al. (2000). The structures studied in this work are two dimensional membranes. In this formulation, a membrane is represented by a set of material points and the fluids/gases by a another set of particles, see Fig. 1.23a. Both types of particles interact with each other via the background grid. The constitutive model for the membranes is such that they only sustains axial stress. The algorithm was applied to airbag impact simulations and the results were found to be in good agreement with experiments.

This model was improved later by other researchers e.g. Gan et al. (2011), Lian et al. (2011c), Lian et al. (2014), Nguyen et al. (2017). Particularly, Lian et al. (2011c) treat the membrane (actually reinforcement bars) as 1D two-node bar elements. They now connect the membrane particles together, see Fig. 1.23b. By doing this, the number of particles necessary to discretize the membrane is significantly reduced

[5] FLAC is a two-dimensional explicit finite difference program for engineering mechanics computation, and a product of Itasca: http://www.itascacg.com/software/flac.

[6] Averaged means a volume weighted sum of particle stresses.

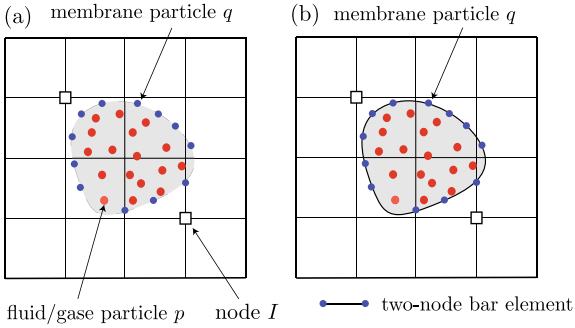


Fig. 1.23 Monolithic FSI solver using the MPM. If the solid part is a structure, it can be discretized by just a set of particles as in **a** or it is modeled as a set of finite elements (bar elements in 2D and membrane elements in 3D)

(Nguyen et al. 2017). In this model, the membrane is discretized by a set of two-node bar finite elements. For these elements, their mass and their internal forces are calculated the FEM way. Then, instead of solving the movement of these elements' nodes the FEM way, the mass and internal forces are projected onto the background grid. Then, the motion of the membrane particles and the fluid particles are updated the standard MPM way.

Based on the work of Lian et al. (2011c), Hamad et al. (2015) developed a new 3D solid-membrane coupling method. It is essentially a coupling of an MPM for the solid and a FEM (three-node triangular elements) for the membrane. The method was applied to simulate the installation process and the behavior of geosynthetics systems for geotechnical applications. In the computer graphics community, similar work has been done. For example Guo et al. (2018) presented an MPM for thin shells with frictional contact where the shells are represented by Catmull-Clark subdivision surfaces of which control points are treated as particles in an MPM method. In a related work, but for bird strike simulations, Wu et al. (2018) presented a coupling of shell finite elements with the MPM (to model the large deformation of the poor birds).

The algorithm of York et al. (1999), York et al. (2000) was applied to fluid-structure interaction problems in Hu et al. (2009), Mao (2013), Yang et al. (2018), Su et al. (2019), Sun et al. (2019). In Su et al. (2019), temperature effects were considered. Sun et al. (2019) presented a set of benchmark tests for FSI problems and verified MPM results against experiments and other numerical methods. All these works only consider no-slip condition on the fluid-structure interface, to the exception of Hu et al. (2009, 2011) who considered slip boundary conditions. Hu and coworkers' works introduced many techniques for a robust MPM-based fluid-structure interaction simulator: (i) interface material points to track the fluid- structure interface, (ii) fluid particle regularization (or redistribution) to alleviate large particle distortion which is typical of fluid motion, (iii) adaptive mesh refinement, using GIMP, to reduce computational cost that is inherent in traditional uniform grids.

Partitioned FSI MPM. There also exist hybrid approaches or partitioned approaches where a fluid flow solver is coupled with an MPM solid solver for FSI problems e.g.

Guilkey et al. (2007), Gilmanov and Acharya (2008b), Sun et al. (2010). This was motivated by the fact that the MPM is not the best solver for fluids.

In Gilmanov and Acharya (2008b), Gilmanov and Acharya (2008a) the hybrid immersed boundary method (HIBM) for fluids is combined with the MPM for solids and is presented as an effective strategy for solving 3D fluid-structure interaction problems. The idea is based on the immersed boundary method of Peskin et al. (2002) where the fluid is treated using a Cartesian grid (with finite difference solver) and the fluid-structure interface is immersed in this grid. In Gilmanov and Acharya (2008b), the structure is a 3D soft membrane (for example a capsule moving in a blood) which is discretized by an unstructured mesh made of three-node triangular finite elements. The deformation of the membrane was, however, not treated by using the FEM but the MPM. That is, the membrane mass and internal forces (calculated the FEM way) are projected to an MPM grid. Therefore, there are the grids in this method: one Cartesian grid for the fluid, one unstructured grid for the membrane, and one Cartesian background grid to solve for the membrane deformation. As one can see, the work of Lian et al. (2011c), Hamad et al. (2015) reinvented this algorithm (without knowing its existence).

1.6.3 Image-Based Simulations

The MPM (or any meshfree methods) is better suited for image-based simulations involving large deformations than the FEM. This is because the MPM requires only a set of particles, not a body-fitted mesh. It is easier to transform an image into a set of points (particles in the MPM) rather than a finite element mesh. The algorithm used for this is as simple as reading the image voxel by voxel, converting each voxel into a material point. This idea has been exploited in a couple of works such as Bardenhagen et al. (2005), Brydon et al. (2005), Lelong and Rochais (2019) for the simulation of foam microstructures, Nairn (2006) for woods, Lee and Huang (2010) for low-density snow, and Xue et al. (2006b), Xue et al. (2006a) for highly filled composites and nanoparticle-polymer composite membranes.

Image-based simulations find natural applications in biomechanics, see for instance Guilkey et al. (2006), Liu et al. (2015b). A recent work in this direction is that of Liu and Sun (2019) in which a shift boundary method to apply boundary conditions on surfaces not aligned with the grid nodes is introduced.

There are two points warrant further discussions on this. First, all meshless methods, not just the MPM, are suitable for image-based numerical simulations. Second, when images of very high resolution are involved, the resulting numerical model can be very large. Therefore converting each voxel to a particle might not be the most efficient way to do image-based simulations. In spite of that, as of today and to the best of our knowledge, nobody has published more efficient algorithms.

Remark 9 It should be noted that there are many contact events in the simulations of foam densification reported in Bardenhagen et al. (2005), Brydon et al. (2005) but

they are just no-slip contacts. If needed, frictional contacts of these complex foam materials can be treated using the self-contact method of Homel and Herbold (2017). But this has yet to be done.

1.6.4 Computer Graphics

Why discussing works done by the computer graphics community? Because innovative MPM algorithms have been developed by this community. Their computer science knowledge helped them to write efficient MPM codes that can run in real-time (Hu et al. 2019). We, engineers, can definitely benefit from their contributions. Being aware of the advances made by the computer graphics community would prevent people from reinventing the wheel.

The MPM is now popular in computer graphics. It is integrated into the production framework of Walt Disney Animation Studios and has been used in featured animations including Frozen, Big Hero 6 and Zootopia (Jiang et al. 2016). The power of the MPM has been demonstrated in a number of papers for simulating various materials including elastic objects, sand, cloth, hair, snow, lava, and viscoelastic fluids (Daviet and Bertails-Descoubes 2016; Fu et al. 2017; Jiang et al. 2017; Hu et al. 2018; Wolper et al. 2019; Han et al. 2019). It all started with the pioneering work of Stomakhin et al. (2014a) who developed an semi-implicit MPM technique and constitutive model to animate the unique behavior of snow. The snow is treated as a continuum avoiding the need to model every snow grain. Jiang et al. (2015a) subsequently extended this MPM to incorporate phase changes due to heat flow. Yue et al. (2015) presented a method to simulate dense foams that exhibit nonlinear, viscoplastic behaviors using the highly flexible Herschel-Bulkley constitutive model. To robustly treat large shearing effects characteristic of dense foams, the authors developed a particle resampling technique, based on the Poisson disk sampling, for the MPM to prevent the formation of nonphysical voids. Topology optimization Li et al. (2020b).

The computer graphics community focuses on the efficiency and visual effects of simulators, but not so much on the physics. So, if engineers want to use advances made by this community, they need to make sure that the physics relevant for the considered problems are well modeled.

1.6.5 Other Applications

In the context of geophysical simulation, Moresi et al. (2003), Moresi et al. (2007) reinterpreted the MPM as a type of FEM with the particles as integration points. They coined the method FEM with Lagrange integration points (FEMLIP). They computed quadrature weights such that affine functions can be exactly re-constructed, giving a second-order accurate reconstruction. Their entire algorithm does, however,

not achieve second-order accuracy, due to other low-order approximations as in the standard MPM. Later, they proposed a multiscale MPM in the sense of FE^2 methods (reviews of which are given in Geers et al. (2010), Nguyen et al. (2012)) using computational homogenization to obtain on-the-fly micromechanically derived constitutive behaviors.

The MPM was used in sea ice models for climate simulation (Sulsky et al. 2007), snow avalanche (Gaume et al. 2018, 2019; Li et al. 2020a, 2021) as well as more traditional explosive-related simulations e.g. explosive welding (Wang et al. 2011), high explosive explosion (Ma et al. 2009b), explosively driven metals (Lian et al. 2011b), cutting processes (Ambati et al. 2012), high melting explosive with cavities (Pan et al. 2008), blast and fragmentation (Banerjee 2004; Hu and Chen 2006), and wear (Mishra et al. 2019). Different low and high velocity impact problems were studied in Li et al. (2011), Zhou et al. (2013), Zhiping Ye et al. (2018).

1.7 Open Source and Commercial MPM Codes

Open source codes play an enormous role in the development of any numerical method. In the context of the MPM, there are various open source codes written in high-level and low-level programming languages. We discuss these codes herein. The aim is to help the readers to find the MPM implementation that suits their needs best.

MPM codes written in a high level, dynamic language such as Matlab or Julia can be found in Sinaie et al. (2017), Coombs and Augarde (2020). Julia is a high-level programming language designed for high-performance numerical analysis and computational science, similar to Matlab but much faster. These codes are best suited for education and implementation of new ideas. However, they are quite slow for computationally intensive simulations.

On the other hand, to solve large scale problems, one would need a more efficient MPM code, usually implemented in a low-level language such as Fortran or C++. This need can probably be covered by existing efficient open source codes:

- `Uintah`, http://www.uintah.utah.edu
- `Mechsys`, http://mechsys.nongnu.org/index.shtml
- `MPM-GIMP`, http://sourceforge.net/p/mpmgimp/home/Home/
- `NairnMPM`, http://osupdocs.forestry.oregonstate.edu/index.php/Main_Page
- `CB-Geo`, https://www.cb-geo.com/research/mpm/
- `MPM3D`, http://comdyn.hy.tsinghua.edu.cn/english/mpm3d.

All are written in C++. `CB-Geo` is an MPM code for geomechanics problems. `Uintah` implements probably the most efficient MPM to date: it uses blocked structured adaptive mesh refinement (BSAMR) grids running on hundreds to thousands of processors. It is not easy for people new to this code to modify it though.

For engineers and scientists who just want to use existing MPM packages for their research, open source MPM codes are as useful as commercial FE packages such as `Abaqus`. To researchers whose work involves developing new MPM techniques, these codes are also useful as it is not an easy task to develop an efficient, scalable and extensible MPM code from scratch.

A commercial MPM package is `MPMsim`, found at http://www.mpmsim.com. The core is written in C++ and it has a user friendly interface written in Matlab. It can convert CAD files to material points.

1.8 Layout

The previous discussion has painted an overall picture about the MPM. The remaining of the book is to fill in the details. Eleven ready-to-code algorithms are provided such that everyone can implement the MPM and carry out the simulations given in Fig. 1.24. The implementation can be done either in easy-to-use programming languages such as `Matlab` and `Julia` or in the static language C++.

The remaining of the book is organized as follows.

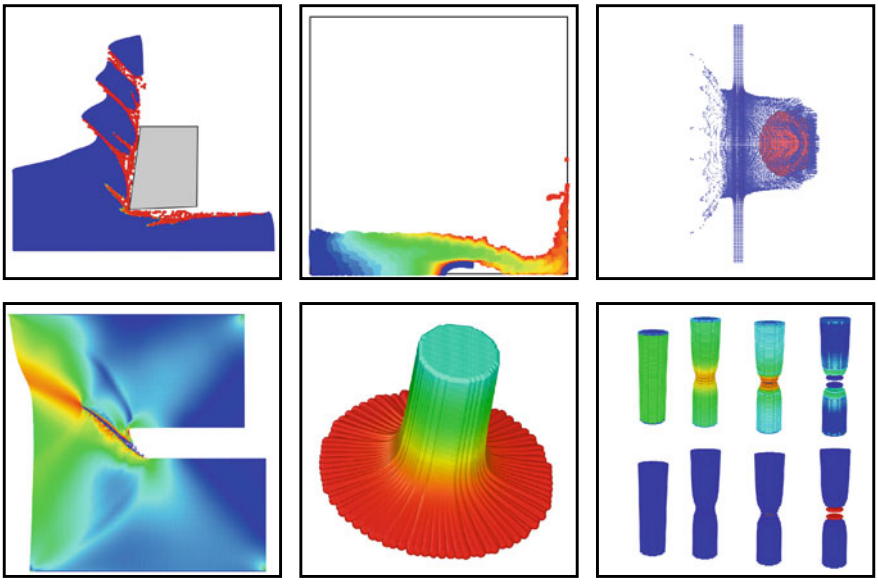


Fig. 1.24 Representative simulations that can be done using MPM algorithms presented in this book. These simulations involve very large deformation, contacts and fracture. And one simulation is a fluid-structure interaction problem

Chapter 2 describes the general basic MPM for solids. By general, we mean that no specification of the shape functions is given. And by basic, we mean that collisions/contacts are not discussed. Both the updated Lagrangian and total Lagrangian MPM are covered. Plane stress/strain, axi-symmetric and 3D formulations are all discussed.

Chapter 3 presents different MPM versions that basically adopt different shape functions e.g. hat functions, B-splines, Bernstein, GIMP and CPDI. They provide specific formulae for $\phi_I(\mathbf{x})$ in the MPM algorithm, cf. Fig. 1.12. Also treated is a new formulation called generalized particle in cell (GPIC) which combines the FEM and MPM that takes advantages of both methods.

Chapter 4 provides some common constitutive models such as linear elastic, hyperelastic and elasto-viscoplastic models. These models furnish a way to compute the stress $\boldsymbol{\sigma}$ in Fig. 1.12. All the components of a material point method for solid mechanics are now ready. We move to implementation details.

Chapter 5 discusses some implementation details such as particle generation for simple geometries and for images, initial and boundary conditions, MPM with unstructured grids and visualization of MPM results. These implementation details will be used in our different codes discussed in the next two chapters.

Chapter 6 presents a tutorial MPM code written in Matlab. This code serves to illustrate the MPM algorithms discussed in the book and it can be used to solve one, two and three dimensional problems. However, only small problems should be solved using this code. The chapter prodives some MPM simulations involving a collision of elastic solids, high velocity impact of a steel disk into a plastic solid and lateral compression of thin-walled steel tubes. This last simulation is validated against available experiments and the involved validation process is given.

Chapter 7 describes `Karamelo`—an open source parallel C++ package for the material point method. This code can be used to solve large-scale problems as it can be run on multiple processors using MPI. Simulations shown in Fig. 1.24 were solved using this code. With such an efficient code, we present three dimensional simulations to demonstrate the capability of the MPM in solving large deformation solid mechanics problems.

Chapter 8 presents some advanced topics such as contacts and fracture. For modeling contacts, we present updated Lagrangian MPM, total Lagrangian MPM and GPIC contact formulations. Both frictionless and frictional contact are discussed. One notable application of these contact algorithms is the simulation of scratch test–a popular mechanical test to measure a solid's hardness. We then discuss fracture modeling in the framework of the MPM. We present different nonlocal fracture/-damage theories including the variational phase-field fracture and nonlocal gradient enhanced Johnson-Cook damage model. Application of the MPM to model large strain ductile fracture of metals is then provided.

Chapter 9 discusses the mathematical analysis of the MPM regarding its stability and accuracy. We discuss the conservation of energy and momenta in various MPM variants. We then present the method of manufactured solutions—a method suitable for testing convergence of nonlinear codes. We describe the improved MPM (iMPM) that uses MLS (moving least square) data construction in a MPM framework to

enhance the convergence rate of the MPM. We study the convergence behavior of all MPM variants discussed in this book for a problem involving only compression/tension deformation and another problem involving simple shear with superimposed rotation. Convergence tests reveal shortcomings (if any) of any method and thus provide guidelines for improvements.

Chapter 10 discusses fluid/gases modeling, membrane modeling and heat conduction. With the information provided in this chapter, one can carry out fluid-structure interaction simulations, air-bag simulations and thermo-mechanical simulations.

The book contains some appendices. Appendix A discusses the strong and weak form of the momentum equation and their equivalence. Appendix B presents derivation of various CPDI basis functions. Some useful utilities such as how to use an open source computer algebra system (SageMath and SymPy) to derive CPDI functions, how to use remote machines to run large-scale simulations and consistent units are given in Appendix C. Appendix D gives a short but practical presentation of updated and total Lagrangian, explicit dynamics FEM for nonlinear solid mechanics. This is useful to see the difference and similarities of the MPM and the FEM. Appendix E treats implicit dynamics FEM so that it is easier to understand implicit MPM (even though this is not discussed in this book). Finally, we describe yet another MPM code in Appendix F, now written in Julia—a new high level dynamic programming language which is easy to use as Python and as fast as C.

1.9 Notations

Three notations are adopted in this work namely indicial notation, tensor notation and matrix notation (also known as engineering notation). The squared magnitude of a three-dimensional vector expressed in these three notations is given in the following equation

$$r^2 = \underbrace{x_i x_i}_{\text{indicial notation}} = \underbrace{\mathbf{x}\cdot\mathbf{x}}_{\text{tensor notation}} = \underbrace{\mathbf{x}^{\mathrm{T}}\mathbf{x}}_{\text{matrix notation}} \tag{1.18}$$

Any vectors and tensors in this book is defined in a rectangular Cartesian coordinate system. Therefore a vector $\mathbf{x}$ can be written as

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad \text{or} \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{1.19}$$

In indicial notation, the components of tensors are explicitly specified e.g. a vector in indicial notation is hence given by x_i in which the index i ranges from one to the number of spatial dimensions. Indices which are repeated twice in a term are

Table 1.2 Important notations used in this contribution. Subscript I denotes the value of grid nodes, and subscript p denotes the value of particles

Variable	Type	Meaning
$\mathbf{x}_p$	Vector	Particle position (time-dependent)
$\mathbf{X}_p$	Vector	Particle initial position
$\mathbf{v}_p$	Vector	Particle velocity
m_p	Scalar	Particle mass
V_p	Scalar	Particle volume
ρ_p	Scalar	Particle density
T_p	Scalar	Particle temperature
$\boldsymbol{\sigma}_p$	Tensor/Matrix	Particle Cauchy stress
$\mathbf{P}_p$	Tensor/Matrix	Particle 1st Piola-Kirchoff stress
$\mathbf{F}_p$	Tensor/Matrix	Particle deformation gradient
$\mathbf{L}_p$	Tensor/Matrix	Particle velocity gradient
$\mathbf{D}_p$ or $\dot{\boldsymbol{\epsilon}}_p$	Tensor/Matrix	Particle rate of deformation
$\mathbf{v}_I$	Vector	Node velocity
$\tilde{\mathbf{v}}_I^{t+\Delta t}$	Vector	Temporary updated node velocity
$\mathbf{v}_I^{t+\Delta t}$	Vector	Final updated node velocity
m_I	Scalar	Node mass
$\phi_I(\mathbf{x}_p)$ or ϕ_{Ip}	Scalar	Weighting function of node I evaluated at particle p
$N_I(\mathbf{x}_p)$	Scalar	Hat functions of node I evaluated at p
$\nabla\phi_I(\mathbf{x}_p)$ or $\nabla\phi_{Ip}$	Vector	Gradient of weighting function of node I evaluated at particle p
$\nabla_0\phi_I(\mathbf{x}_p)$ or $\nabla_0\phi_{Ip}$	Vector	Gradient (w.r.t $\mathbf{X}$) of weighting function of node I evaluated at p

summed, a rule known as the Einstein summation, an example of which is the squared magnitude of a vector:

$$r^2 = x_i x_i = x_1^2 + x_2^2 + x_3^2 \tag{1.20}$$

Scalars are written using a normal font whereas tensors of order one or greater are expressed in boldface. Table 1.2 provides a list of important notations used in this contribution. And Table 1.3 presents abbreviations.

Table 1.3 A list of abbreviations used in this book

Abbreviation	Full name
MPM	Material Point Method
ULMPM	Updated Lagrangian Material Point Method
TLMPM	Total Lagrangian Material Point Method
PIC	Particle in Cell
FLIP	Fluid Implicit Particle method
FEM	Finite Element Method
ULFEM	Updated Lagrangian Finite Element Method
TLFEM	Total Lagrangian Finite Element Method
CPDI	Convected Particle Domain Integrator
GIMP	Generalized Interpolation Material Point
BSMPM	B-splines Material Point Method
DDMPM	Dual Domain Material Point Method
SPH	Smoothed Particle Hydrodynamics
iMPM	improved Material Point Method
GPIC	Generalized Particle in Cell Method

References

Abe, K., Soga, K., Bandara, S.: Material point method for coupled hydromechanical problems. J. Geotech. Geoenviron. Eng. **140**(3), 04013033 (2014)

Agarwal, S., Senatore, C., Zhang, T., Kingsbury, M., Iagnemma, K., Goldman, D.I., Kamrin, K.: Modeling of the interaction of rigid wheels with dry granular media. J. Terramech. **85**, 1–14 (2019)

Al-Kafaji, I.K.J.: Formulation of a Dynamic Material Point Method (MPM) for Geomechanical Problems. Ph.D. thesis, University of Stuttgart (2013)

Alonso, E.E., Zabala, F.: Progressive failure of Aznalcóllar dam using the material point method. Géotechnique **61**(9), 795–808 (2011)

Ambati, R., Pan, X., Yuan, H., Zhang, X.: Application of material point methods for cutting process simulations. Comput. Mater. Sci. **57**, 102–110 (2012)

Andersen, S., Andersen, L.: Modelling of landslides with the material-point method. Comput. Geosci. **14**(1), 137–147 (2010)

Arroyo, M., Ortiz, M.: Local maximum-entropy approximation schemes: a seamless bridge between finite elements and meshfree methods. Int. J. Numer. Meth. Eng. **65**(13), 2167–2202 (2006)

Atluri, S.N., Zhu, T.: A new meshless local Petrov-Galerkin (MLPG) approach in computational mechanics. Comput. Mech. **22**, 117–127 (1998)

Ayton, G., Bardenhagen, S.G., McMurtry, P., Sulsky, D., Voth, G.A.: Interfacing continuum and molecular dynamics: an application to lipid bilayers. J. Chem. Phys. **114**(15), 6913–6924 (2001)

Babuška, I., Banerjee, U., Osborn, J.E.: Meshless and generalized finite element methods: a survey of some major results. In: Griebel, M., Schweitzer, M.A. (eds.) Meshfree Methods for Partial Differential Equations. Lecture Notes in Computational Science and Engineering, vol. 26, pp. 1–20. Springer, Berlin (2002)

Banerjee, B.: Material point method simulations of fragmenting cylinders. In: 17th ASCE Engineering Mechanics Conference, University of Delaware, Newark, DE (2004)
Bardenhagen, S.G., Kober, E.M.: The generalized interpolation material point method. Comput. Model. Eng. Sci. **5**(6), 477–495 (2004)
Bardenhagen, S.G., Brackbill, J.U., Sulsky, D.: The material-point method for granular materials. Comput. Methods Appl. Mech. Eng. **187**(3–4), 529–541 (2000)
Bardenhagen, S.G., Guilkey, J.E., Roessig, K.M., Brackbill, J.U., Witzel, W.M., Foster, J.C.: An improved contact algorithm for the material point method and application to stress propagation in granular material. Comput. Model. Eng. Sci. **2**(4), 509–522 (2001)
Bardenhagen, S.G., Brydon, A.D., Guilkey, J.E.: Insight into the physics of foam densification via numerical simulation. J. Mech. Phys. Solids **53**(3), 597–617 (2005)
Bardenhagen, S.G., Nairn, J.A., Lu, H.: Simulation of dynamic fracture with the Material Point Method using a mixed J-integral and cohesive law approach. Int. J. Fract. **170**(1), 49–66 (2011)
Bazilevs, Y., Takizawa, K.: Advances in Computational Fluid-Structure Interaction and Flow Simulation. Springer (2017)
Belytschko, T., Lu, Y.Y., Gu, L.: Element-free Galerkin methods. Int. J. Numer. Methods Eng. **37**, 229–256 (1994)
Belytschko, T., Krongauz, Y., Organ, D., Fleming, M., Krysl, P.: Meshless methods: an overview and recent developments. Comput. Methods Appl. Mech. Eng. **139**, 3–47 (1996)
Benson, D.J.: Computational methods in Lagrangian and Eulerian hydrocodes. Comput. Methods Appl. Mech. Eng. **99**(2–3), 235–394 (1992)
Beuth, L.: Formulation and Application of a Quasi-Static Material Point Method. Ph.D. thesis, University of Stuttgart (2012)
Beuth, L., Wieckowski, Z., Vermeer, P.A.: Solution of quasi-static large-strain problems by the material point method. Int. J. Numer. Anal. Meth. Geomech. **35**(13), 1451–1465 (2011)
Bo, W.,: Chen, Z., Zhang, X., Liu, Y., Lian, Y.: Coupled shell-material point method for bird strike simulation. Acta Mech. Solida Sin. **31**(1), 1–18 (2018)
Bourdin, B., Francfort, G.A., Marigo, J.J.: The variational approach to fracture. J. Elast. **91**(1–3), 5–148 (2008)
Boyce, B.L., Kramer, S.L.B., Eliot Fang, H., Cordova, T.E., Neilsen, M.K., Dion, K., Kaczmarowski, A.K., Karasz, E., Xue, L., Gross, A.J., et al.: The sandia fracture challenge: blind round robin predictions of ductile tearing. Int. J. Fract. **186**(1–2), 5–68 (2014)
Boyce, B.L., Kramer, S.L.B., Bosiljevac, T.R., Corona, E., Moore, J.A., Elkhodary, K., Simha, C.H.M., Williams, B.W., Cerrone, A.R., Nonn, A., et al.: The second sandia fracture challenge: predictions of ductile failure under quasi-static and moderate-rate dynamic loading. Int. J. Fract. **198**(1–2), 5–100 (2016)
Brackbill, J.U., Ruppel, H.M.: FLIP: a method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. J. Comput. Phys. **65**(2), 314–343 (1986)
Brackbill, J.U., Kothe, D.B., Ruppel, H.M.: Flip: a low-dissipation, particle-in-cell method for fluid flow. Comput. Phys. Commun. **48**(1), 25–38 (1988)
Brydon, A.D., Bardenhagen, S.G., Miller, E.A., Seidler, G.T.: Simulation of the densification of real open-celled foam microstructures. J. Mech. Phys. Solids **53**(12), 2638–2660 (2005)
Burghardt, J., Leavy, B., Guilkey, J., Xue, Z., Brannon, R.: Application of Uintah-MPM to shaped charge jet penetration of aluminum. IOP Conf. Ser.: Mater. Sci. Eng. **10**(1), 012223 (2010)
Burghardt, J., Brannon, R., Guilkey, J.: A nonlocal plasticity formulation for the material point method. Comput. Methods Appl. Mech. Eng. **225–228**, 55–64 (2012)
Burman, E., Claus, S., Hansbo, P., Larson, M.G., Massing, A.: Cutfem: discretizing geometry and partial differential equations. Int. J. Numer. Methods Eng. **104**(7), 472–501 (2015)
Ceccato, F., Redaelli, I., di Prisco, C., Simonini, P.: Impact forces of granular flows on rigid structures: comparison between discontinuous (DEM) and continuous (MPM) numerical approaches. Comput. Geotech. **103**, 201–217 (2018)
Chen, Z., Hu, W., Shen, L., Xin, X., Brannon, R.: An evaluation of the MPM for simulating dynamic failure with damage diffusion. Eng. Fract. Mech. **69**(17), 1873–1890 (2002)

Chen, Z., Gan, Y., Chen, J.K.: A coupled thermo-mechanical model for simulating the material failure evolution due to localized heating. Comput. Model. Eng. Sci. **26**(2), 123 (2008)
Chen, Z., Han, Y., Jiang, S., Gan, Y., Sewell, T.D.: A multiscale material point method for impact simulation. Theor. Appl. Mech. Lett. **2**(5), 051003 (2012)
Chen, Z.P., Qiu, X.M., Zhang, X., Lian, Y.P.: Improved coupling of finite element method with material point method based on a particle-to-surface contact algorithm. Comput. Methods Appl. Mech. Eng. **293**, 1–19 (2015)
Cheon, Y-J., Kim, H.G.: An adaptive material point method coupled with a phase-field fracture model for brittle materials. Int. J. Numer. Methods Eng. (2019)
Chorin, A.J.: Numerical solution of the Navier-Stokes equations. Math. Comput. **22**(104), 745–762 (1968)
Clough, R.W.: The finite element method in plane stress analysis. In: Proceedings of 2nd ASCE Conference on Electronic Computation, Pittsburgh Pa., Sept. 8 and 9, 1960 (1960)
Clough, R.W.: The finite element method after twenty-five years: a personal view. Comput. Struct. **12**(4), 361–370 (1980)
Coetzee, C.J.: The Modelling of Granular Flow Using the Particle-in-Cell Method. Ph.D. thesis, University of Stellenbosch (2003)
Coetzee, C.J., Vermeer, P.A., Basson, A.H.: The modelling of anchors using the material point method. Int. J. Numer. Anal. Meth. Geomech. **29**(9), 879–895 (2005)
Coetzee, C.J., Basson, A.H., Vermeer, P.A.: Discrete and continuum modelling of excavator bucket filling. J. Terrramech. **44**(2), 177–186 (2007)
Coombs, W.M., Augarde, C.E.: Ample: a material point learning environment. Adv. Eng. Softw. **139**, 102748 (2020)
Courant, R.: Variational methods for the solution of problems of equilibrium and vibrations. Bull. Am. Math. Soc. **49**(1), 1–23 (1943)
Cummins, S.J., Brackbill, J.U.: An implicit particle-in-cell method for granular materials. J. Comput. Phys. **180**(2), 506–548 (2002)
Cundall, P.A., Strack, O.D.L.: A discrete numerical model for granular assemblies. Geotechnique **29**(1), 47–65 (1979)
Daphalapurkar, N.P., Lu, H., Coker, D., Komanduri, R.: Simulation of dynamic crack growth using the generalized interpolation material point (GIMP) method. Int. J. Fract. **143**(1), 79–102 (2007)
Daviet, G., Bertails-Descoubes, F.: A semi-implicit material point method for the continuum simulation of granular materials. ACM Trans. Graph. (TOG) **35**(4), 102 (2016)
de Vaucorbeil, A., Nguyen:, V.P.: Modeling contacts with a total lagrangian matertial point method. Comput. Methods Appl. Mech. Eng. **360**, 112783 (2021). https://doi.org/10.1016/j.cma.2019.112783
de Vaucorbeil, A., Nguyen, V.P., Mandal, T.K.: Mesh objective simulations of large strain ductile fracture: a new nonlocal Johnson-cook damage formulation for the total lagrangian material point method. Comput. Methods Appl. Mech. Eng. **389**, 114388 (2022b)
de Koster, P., Tielen, R., Wobbes, E., Moller, M.: Extension of B-spline material point method for unstructured triangular grids using powell-sabin splines. Comput. Mech. (2019)
de Vaucorbeil, A., Nguyen, V.P., Hutchinson, C.R.: A Total-Lagrangian Material Point Method for solid mechanics problems involving large deformations. Comput. Methods Appl. Mech. Eng. **360**, 112783 (2020). https://doi.org/10.1016/j.cma.2019.11278
Doblaré, M., Cueto, E., Calvo, B., Martínez, M.A., Garcia, J.M., Cegonino, J.: On the employ of meshless methods in biomechanics. Comput. Methods Appl. Mech. Eng. **194**(6–8), 801–821 (2005)
Duarte, C.A.M., Oden, J.T.: H-p clouds- an h-p meshless method. Numer, Methods Partial Differential Equations (1996)
Dunatunga, S., Kamrin, K.: Continuum modeling and simulation of granular flows through their many phases. J. Fluids Mech. (2015)
Edwards, E., Bridson, R.: A high-order accurate particle-in-cell method. Int. J. Numer. Meth. Eng. **90**(9), 1073–1088 (2012)

Espluga, M.U.: Analysis of Meshfree Methods for Lagrangian Fluid-Structure Interaction. Ph.D. thesis, Universidad Politécnica de Madrid (2014)
Fagan, T., Lemiale, V., Nairn, J., Ahuja, Y., Ibrahim, R., Estrin, Y.: Detailed thermal and material flow analyses of friction stir forming using a three-dimensional particle based model. J. Mater. Process. Technol. **231**, 422–430 (2016)
Fasshauer, G.E.: Meshfree Approximation Methods with MATLAB. Interdisciplinary Mathematical Sciences (Book 6). World Scientific Publishing Company (2007)
Fern, J., Rohe, A., Soga, K., Alonso, E.: The Material Point Method for Geotechnical Engineering: A Practical Guide. CRC Press (2019)
Fu, C., Guo, Q., Gast, T., Jiang, C., Teran, J.: A polynomial particle-in-cell method. ACM Trans. Graph. **36**(6), 222:1-222:12 (2017)
Galavi, V., Beuth, L., Coelho, B.Z., Tehrani, F.S., Hölscher, P., Van Tol, F.: Numerical simulation of pile installation in saturated sand using material point method. Proc. Eng. **175**, 72–79 (2017)
Gan, Y., Chen, Z., Montgomery-Smith, S.: Improved material point method for simulating the zona failure response in piezo-assisted intracytoplasmic sperm injection. Comput. Model. Eng. Sci. 1–24 (2011)
Gan, Y., Sun, Z., Chen, Z., Zhang, X., Liu, Y..: Enhancement of the material point method using B-spline basis functions. Int. J. Numer. Methods Eng. **113**(3), 411–431 (2018)
Gander, M.J., Wanner, G.: From Euler, Ritz, and Galerkin to modern computing. Siam Rev. **54**(4), 627–666 (2012)
Gaume, J., van Herwijnen, A., Gast, T., Teran, J., Jiang, C.: Investigating the release and flow of snow avalanches at the slope-scale using a unified model based on the material point method. Cold Reg. Sci. Technol. **168**, 102847 (2019)
Gaume, J., Gast, T., Teran, J., van Herwijnen, A., Jiang, C.: Dynamic anticrack propagation in snow. Nat. Commun. **9**(1), 3047 (2018)
Geers, M.G.D., Kouznetsova, V.G., Brekelmans, W.A.M.: Multi-scale computational homogenization: trends and challenges. J. Comput. Appl. Math. **234**(7), 2175–2182 (2010)
Gilabert, F.A., Cantavella, V., Sánchez, E., Mallol, G.: Modelling fracture process in ceramic materials using the material point method. EPL (Europhys. Lett.) **96**(2), 24002 (2011)
Gilmanov, A., Acharya, S.: A computational strategy for simulating heat transfer and flow past deformable objects. Int. J. Heat Mass Transf. **51**(17–18), 4415–4426 (2008)
Gilmanov, A., Acharya, S.: A hybrid immersed boundary and material point method for simulating 3D fluid-structure interaction problems. Int. J. Numer. Meth. Fluids **56**(12), 2151–2177 (2008)
Gingold, R.A., Monaghan, J.J.: Smoothed particle hydrodynamics: theory and application to non-spherical stars. Mon. Notices Royal Astro. Soc. **181**, 375–389 (1977)
Gracia, F., Villard, P., Richefeu, V.: Comparison of two numerical approaches (DEM and MPM) applied to unsteady flow. Comput. Part. Mech. 1–19 (2019)
Griffith, A.A.: The phenomena of rupture and flow in solids. Phil. Trans. Roy. Soc. Lond. **221**, 163–198 (1920)
Gritton, C., Berzins, M.: Improving accuracy in the MPM method using a null space filter. Comput. Part. Mech. **4**(1), 131–142 (2017)
Gritton, C., Guilkey, J., Hooper, J., Bedrov, D., Kirby, R.M., Berzins, M.: Using the material point method to model chemical/mechanical coupling in the deformation of a silicon anode. Model. Simul. Mater. Sci. Eng. **25**(4), 045005 (2017)
Guilkey, J.E., Weiss, J.A.: Implicit time integration for the material point method: quantitative and algorithmic comparisons with the finite element method. Int. J. Numer. Meth. Eng. **57**(9), 1323–1338 (2003)
Guilkey, James E., Hoying, James B., Weiss, Jeffrey A.: Computational modeling of multicellular constructs with the material point method. J. Biomech. **39**(11), 2074–2086 (2006)
Guilkey, J.E., Harman, T.B., Banerjee, B.: An Eulerian-Lagrangian approach for simulating explosions of energetic devices. Comput. Struct. **85**(11–14), 660–674 (2007)
Guo, Q., Han, X., Chuyuan, F., Gast, T., Tamstorf, R., Teran, J.: A material point method for thin shells with frictional contact. ACM Trans. Graph. (TOG) **37**(4), 147 (2018)

Guo, Y., Nairn, J.A.: Calculation of j-integral and stress intensity factors using the material point method. Comput. Model. Eng. Sci. **6**, 295–308 (2004)
Gupta, V., Rajagopal, S., Gupta, N.: A comparative study of meshfree methods for fracture. Int. J. Damage Mech **20**(5), 729–751 (2011)
Hamad, F., Stolle, D., Vermeer, P.: Modelling of membranes in the material point method with applications. Int. J. Numer. Anal. Meth. Geomech. **39**(8), 833–853 (2015)
Han, X., Gast, T.F., Guo, Q., Wang, S., Jiang, C., Teran, J.: A hybrid material point method for frictional contact with diverse materials. Proc. ACM Comput. Graphics Interact. Tech. **2**(2), 17 (2019)
Harlow, F.H.: The particle-in-cell computing method for fluid dynamics. Methods Comput. Phys. **3**, 319–343 (1964)
Harlow, F.H.: Fluid dynamics in Group T-3 Los Alamos National Laboratory: (LA-UR-03-3852). J. Comput. Phys. **195**(2), 414–433 (2004)
He, L., Chen, Z.: Study on one-dimensional softening with localization via integrated MPM and SPH. Comput. Part. Mech. 1–8 (2019)
He, L., Gan, Y., Chen, Z.: Preliminary effort in developing the smoothed material point method for impact. Comput. Part. Mech. **6**(1), 45–53 (2019)
Homel, M.A., Guilkey, J.E., Brannon, R.M.: Continuum effective-stress approach for high-rate plastic deformation of fluid-saturated geomaterials with application to shaped-charge jet penetration. Acta Mech. 1–32 (2015)
Homel, M.A., Herbold, E.B.: Mesoscale modeling of porous materials using new methodology for fracture and frictional contact in the material point method. In: Dynamic Behavior of Materials, vol. 1, pp. 97–102. Springer (2018)
Homel, M.A., Herbold, E.B.: Field-gradient partitioning for fracture and frictional contact in the material point method. Int. J. Numer. Meth. Eng. **109**(7), 1013–1044 (2017)
Homel, M.A., Brannon, R.M., Guilkey, J.: Controlling the onset of numerical fracture in parallelized implementations of the material point method (MPM) with convective particle domain interpolation (CPDI) domain scaling. Int. J. Numer. Meth. Eng. **107**(1), 31–48 (2016)
Hsieh, Y-M., Pan, M-S.: Esfm: an essential software framework for meshfree methods. Adv. Eng. Softw. **76**, 133–147 (2014)
Hu, Y., Liu, J., Spielberg, A., Tenenbaum, J.B., Freeman, W.T., Wu, J., Rus, D., Matusik, W.: Chainqueen: A real-time differentiable physical simulator for soft robotics. In: 2019 International Conference on Robotics and Automation (ICRA), p. 6265–6271. IEEE (2019)
Hu, P., Xue, L., Kamakoti, R., Li, Q., Wang, Z., Brenner, M.: Particle-based methods with least squares technique for nonlinear aeroelasticity and fluid-structure interactions in aste-p toolset. In: 52nd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference 19th AIAA/ASME/AHS Adaptive Structures Conference 13t, p. 2062 (2011)
Hu, P., Xue, L., Qu, K., Ni, K., Brenner, M.: Unified Solver for Modeling and Simulation of Nonlinear Aeroelasticity and Fluid-Structure Interactions. American Institute of Aeronautics and Astronautics (2009)
Hu, W., Chen, Z.: Model-based simulation of the synergistic effects of blast and fragmentation on a concrete wall using the MPM. Int. J. Impact Eng **32**(12), 2066–2096 (2006)
Huang, P., Zhang, X., Ma, S., Huang, X.: Contact algorithms for the material point method in impact and penetration simulation. Int. J. Numer. Meth. Eng. **85**(4), 498–517 (2011)
Hughes, T.J.R., Cottrell, J.A., Bazilevs, Y.: Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. Comput. Methods Appl. Mech. Eng. **194**(39–41), 4135–4195 (2005)
Iaconeta, I., Larese, A., Rossi, R., Guo, Z.: Comparison of a material point method and a Galerkin meshfree method for the simulation of cohesive-frictional materials. Materials **10**(10), 1150 (2017)
Idelsohn, S.R., Onate, E., Del Pin, F., Calvo, N.: Fluid-structure interaction using the particle finite element method. Comput. Methods Appl. Mech. Eng. **195**(17–18), 2100–2123 (2006)

Ionescu, I., Guilkey, J.E., Berzins, M., Kirby, R.M., Weiss, J.A.: Simulation of soft tissue failure using the Material Point Method. J. Biomech. Eng. **128**(6), 917–924 (2006)
Irwin, G.R.: Analysis of stresses and strains near the end of a crack traversing a plate. J. Appl. Mech. **24**, 361–364 (1957)
Jassim, I., Stolle, D., Vermeer, P.: Two-phase dynamic analysis by material point method. Int. J. Numer. Anal. Meth. Geomech. **37**(15), 2502–2522 (2013)
Jiang, C., Gast, T., Teran, J.: Anisotropic elastoplasticity for cloth, knit and hair frictional contact. ACM Trans. Graph. (TOG) **36**(4), 152 (2017)
Jiang, Y., Li, M., Jiang, C., Alonso-marroquin, F.: A hybrid material-point spheropolygon-element method for solid and granular material interaction (2019). arXiv:1909.13655
Jiang, C., Schroeder, C., Selle, A., Teran, J., Stomakhin, A.: The affine particle-in-cell method. ACM Trans. Graph. 34(4): 51:1–51:10 (2015a)
Jiang, C., Schroeder, C., Teran, J., Stomakhin, A., Selle, A.: The material point method for simulating continuum materials. In: ACM SIGGRAPH 2016 Courses, p. 24. ACM (2016)
Joshuah Wolper, Y., Fang, M.L., Jiecong, L., Gao, M., Jiang, C.: Cd-MPM: continuum damage material point methods for dynamic fracture animation. ACM Trans. Graph. (TOG) **38**(4), 119 (2019)
Kachanov, L.M.: Time rupture process under creep conditions. Izv. A Rad. Nauk. SSSR otd Tekh. Nauk **8**, 26–31 (1958)
Kakouris, E.G., Triantafyllou, S.P.: Material point method for crack propagation in anisotropic media: a phase field approach. Arch. Appl. Mech. (2017a)
Kakouris, E.G., Triantafyllou, S.P.: Phase-field material point method for brittle fracture. Int. J. Numer. Methods Eng. **112**(12), 1750–1776 (2017b)
Konagai, K., Johansson, J.: Two dimensional lagrangian particle finite-difference method for modeling large soil deformations. Struct. Eng./Earthq. Eng. JSCE **18**(2), 105s–110s (2001)
Kularathna, S., Soga, K.: Comparison of two projection methods for modeling incompressible flows in MPM. J. Hydrodyn. Ser. B **29**(3), 405–412 (2017)
Lee, J.H., Huang, D.: Material point method modeling of porous semi-brittle materials. IOP Conf. Seri.: Mater. Sci. Eng. **10**(1), 012093 (2010)
Lelong, N., Rochais, D.: Influence of microstructure on the dynamic behavior of a polyurethane foam with the material point method. Materialia **5**, 100199 (2019)
Lemaitre. J., Chaboche, J.-L.: Mechanics of Solid Materials. Cambridge University Press (1994)
Lemiale, V., Nairn, J., Hurmane, A.: Material point method simulation of equal channel angular pressing involving large plastic strain and contact through sharp corners. Comput. Model. Eng. Sci. **70**(1), 41–66 (2010)
Leroch, S., Eder, S.J., Ganzenmüller, G., Murillo, L.J.S., Rodríguez Ripoll, M.: Development and validation of a meshless 3D material point method for simulating the micro-milling process. J. Mater. Process. Technol. **262**, 449–458 (2018)
Li, B., Habbal, F., Ortiz, M.: Optimal transportation meshfree approximation schemes for fluid and plastic flows. Int. J. Numer. Methods Eng. **83**(12), 1541–1579 (2010)
Li, Y., Li, X., Li, M., Zhu, Y., Zhu, B., Jiang, C.: A hybrid lagrangian-eulerian method for topology optimization (2020b). arXiv:2003.01215
Li, S., Liu, W.K.: Meshfree Particle Methods [Hardcover]. Springer (2007). ISBN 3540222561
Li, X., Sovilla, B., Jiang, C., Gaume, J.: Three-dimensional and real-scale modeling of flow regimes in dense snow avalanches. Landslides **18**(10), 3393–3406 (2021)
Li, X., Sovilla, B., Wang, S., Jiang, C., Gaume, J.: Numerical modeling of snow avalanche dynamics based on the material point method. In: tEGU General Assembly Conference Abstracts, p. 2153 (2020a)
Li, F., Pan, J., Sinka, C.: Modelling brittle impact failure of disc particles using material point method. Int. J. Impact Eng. **38**(7), 653–660 (2011)
Li, J.G., Hamamoto, Y., Liu, Y., Zhang, X.: Sloshing impact simulation with material point method and its experimental validations. Comput. Fluids **103**, 86–99 (2014)

Lian, Y.-P., Liu, Y., Zhang, X.: Coupling of membrane element with material point method for fluid-membrane interaction problems. Int. J. Mech. Mater. Des. 1–13 (2014)
Lian, Y.P., Zhang, X., Liu, Y.: Coupling of finite element method with material point method by local multi-mesh contact method. Comput. Methods Appl. Mech. Eng. **200**(47–48), 3482–3494 (2011)
Lian, Y.P., Zhang, X., Zhou, X., Ma, Z.T.: A FEMP method and its application in modeling dynamic response of reinforced concrete subjected to impact loading. Comput. Methods Appl. Mech. Eng. **200**(17–20), 1659–1670 (2011)
Lian, Y.P., Zhang, X., Zhou, X., Ma, S., Zhao, Y.L.: Numerical simulation of explosively driven metal by material point method. Int. J. Impact Eng. **38**(4), 238–246 (2011)
Lian, Y., Zhang, X., Liu, Y.: Coupling between finite element method and material point method for problems with extreme deformation. Theor. Appl. Mech. Lett. **021003**, 2–5 (2012)
Liang, Y., Zhang, X., Liu, Yan.: An efficient staggered grid material point method. Comput. Methods Appl. Mech. Eng. **352**, 85–109 (2019)
Lim, L.J., Andreykiv, A., Brinkgreve, R.B.J.: Pile penetration simulation with Material Point Method. In: Installation Effects in Geotechnical Engineering, pp. 24–30. CRC Press (2013)
Liszka, T., Orkisz, J.: The finite difference method at arbitrary irregular grids and its application in applied mechanics. Comput. Struct. **11**, 83–95 (1980)
Liu, G.R., Liu, M.B.: Smoothed Particle Hydrodynamics: A Meshfree Particle Method. World Scientific (2003)
Liu, C., Sun, W.: ILS-MPM: an implicit level-set-based material point method for frictional particulate contact mechanics of deformable particles. Comput. Methods Appl. Mech. Eng. **369**(113168) (2020)
Liu, C., Sun, W.: Shift boundary material point method: an image-to-simulation workflow for solids of complex geometries undergoing large deformation. Computational Particle Mechanics, pp. 1–18 (2019)
Liu, Y., Wang, H-K., Zhang, X.: A multiscale framework for high-velocity impact process with combined material point method and molecular dynamics. Int. J. Mech. Mater. Des. **9**(2), 127–139 (2013)
Liu, G.R.: Mesh Free Methods: Moving Beyond the Finite Element Method. CRC Press (2002)
Liu, W.K., Jun, S., Zhang, Y.F.: Reproducing kernel particle methods. Int. J. Numer. Methods Eng. **20**, 1081–1106 (1995)
Liu, P., Liu, Y., Zhang, X.: Internal-structure-model based simulation research of shielding properties of honeycomb sandwich panel subjected to high-velocity impact. Int. J. Impact Eng **77**, 120–133 (2015)
Liu, Y., Qiu, X., Zhang, X., Yu, T.X.: Response of woodpecker's head during pecking process simulated by material point method. PloS One **10**(4), e0122677 (2015b)
Llano-Serna, M.A., Farias, M.M., Pedroso, D.M.: An assessment of the material point method for modelling large scale run-out processes in landslides. Landslides **13**(5), 1057–1066 (2016)
Love, E., Sulsky, D.L.: An unconditionally stable, energy-momentum consistent implementation of the material-point method. Comput. Methods Appl. Mech. Eng. **195**(33–36), 3903–3925 (2006)
Love, E., Sulsky, D.L.: An energy-consistent material-point method for dynamic finite deformation plasticity. Int. J. Numer. Meth. Eng. **65**(10), 1608–1638 (2006)
Lu, H., Daphalapurkar, N.P., Wang, B., Roy, S., Komanduri, R.: Multiscale simulation from atomistic to continuum-coupling molecular dynamics (MD) with the material point method (MPM). Phil. Mag. **86**(20), 2971–2994 (2006)
Lucy, L.: A numerical approach to the testing of the fission hypothesis. Astron. J. **82**, 1013–1024 (1977)
Ma, S., Zhang, X., Lian, Y., Zhou, X.: Simulation of high explosive explosion using adaptive material point method. Comput. Model. Eng. Sci. (CMES) **39**(2), 101 (2009)
Ma, S., Zhang, X., Qiu, X.M.: Comparison study of MPM and SPH in modeling hypervelocity impact problems. Int. J. Impact Eng **36**(2), 272–282 (2009)

Ma, J., Wang, D., Randolph, M.F.: A new contact algorithm in the material point method for geotechnical simulations. Int. J. Numer. Anal. Meth. Geomech. **38**(11), 1197–1210 (2014)
Malvern, L.E.: Introduction to the Mechanics of a Continuous Medium. Prentice-Hall International, Englewood Cliffs, New Jersey (1969)
Mandal, T.K., Nguyen, V.P., Wu, J.-Y.: Evaluation of variational phase-field models for dynamic brittle fracture. Eng. Fract. Mech. **345**, 618–643 (2020a)
Mao, S.: Material point method and adaptive meshing applied to fluid-structure interaction (FSI) problems. In: ASME 2013 Fluids Engineering Division Summer Meeting, pp. V01BT13A004–V01BT13A004. American Society of Mechanical Engineers (2013)
Mast, C.M., Mackenzie-Helnwein, P., Arduino, P., Miller, G.R., Shin, W.: Mitigating kinematic locking in the material point method. J. Comput. Phys. **231**(16), 5351–5373 (2012)
Miehe, C., Welschinger, F., Hofacker, M.: Thermodynamically consistent phase-field models of fracture: variational principles and multi-field fe implementations. Int. J. Numer. Meth. Engng. **83**, 1273–1311 (2010)
Mishra, T., Ganzenmüller, G.C., de Rooij, M., Shisode, M., Hazrati, J., Schipper, D.J.: Modelling of ploughing in a single-asperity sliding contact using material point method. Wear **418**, 180–190 (2019)
Mittal, R., Iaccarino, G.: Immersed boundary methods. Annu. Rev. Fluid Mech. **37**, 239–261 (2005)
Moresi, L., Dufour, F., Mühlhaus, H.-B.: A Lagrangian integration point finite element method for large deformation modeling of viscoelastic geomaterials. J. Comput. Phys. **184**(2), 476–497 (2003)
Moresi, L., Quenette, S., Lemiale, V., Mériaux, C., Appelbe, B., Mühlhaus, H.-B.: Computational approaches to studying nonlinear dynamics of the crust and mantle. Phys. Earth Planet. Inter. **163**(1–4), 69–82 (2007)
Moutsanidis, G., Kamensky, D., Zhang, D.Z., Bazilevs, Y., Long, C.C.: Modeling strong discontinuities in the material point method using a single velocity field. Comput. Methods Appl. Mech. Eng. **345**, 584–601 (2019a)
Mühlhaus, H.-B., Sakaguchi, H., Moresi, L., Fahey, M.: Discrete and continuum modelling of granular materials. In: Vermeer, P.A., Herrmann, H.J., Luding, S., Ehlers, W., Diebels, S., Ramm, E. (eds.) Continuous and Discontinuous Modelling of Cohesive-Frictional Materials. Lecture Notes in Physics, vol. 568, pp. 185–204. Springer, Berlin, Heidelberg (2001)
Nair, A., Roy, S.: Implicit time integration in the generalized interpolation material point method for finite deformation hyperelasticity. Mech. Adv. Mater. Struct. **19**(6), 465–473 (2012)
Nairn, J.A.: Material point method calculations with explicit cracks. Comput. Model. Eng. Sci. **4**(6), 649–663 (2003)
Nairn, J.A.: Numerical simulations of transverse compression and densification in wood. Wood Fiber Sci. **38**(4), 576–591 (2006)
Nairn, J.A.: Material point method simulations of transverse fracture in wood with realistic morphologies. Holzforschung **61**(4), 375–381 (2007)
Nairn, John A.: Numerical implementation of imperfect interfaces. Comput. Mater. Sci. **40**(4), 525–536 (2007)
Nairn, J.A.: Analytical and numerical modeling of R curves for cracks with bridging zones. Int. J. Fract. **155**(2), 167–181 (2009)
Nairn, J.A.: Modeling imperfect interfaces in the material point method using multimaterial methods. Comput. Methods Appl. Mech. Eng. **1**(1), 1–15 (2013)
Nairn, J.A., Guilkey, J.E.: Axisymmetric form of the generalized interpolation material point method. Int. J. Numer. Meth. Eng. **101**(2), 127–147 (2015)
Nairn, J.A., Bardenhagen, S.G., Smith, G.D.: Generalized contact and improved frictional heating in the material point method. Comput. Part. Mech. **5**(3), 285–296 (2018)
Nayroles, B., Touzot, G., Villon, P.: Generalizing the finite element method: diffuse approximation and diffuse elements. Comput. Mech. **10**, 307–318 (1992)
Nguyen, V.P.: Discontinuous Galerkin/Extrinsic cohesive zone modeling: implementation caveats and applications in computational fracture mechanics. Eng. Fract. Mech. **128**, 37–68 (2014)

Nguyen, V.P., Rabczuk, T., Bordas, S., Duflot, M.: Meshless methods: a review and computer implementation aspects. Math. Comput. Simul. **79**(3), 763–813 (2008). ISSN 0378-4754
Nguyen, V.P., Stroeven, M., Sluys, L.J.: Multiscale continuous and discontinuous modelling of heterogeneous materials: a review on recent developments. J. Multiscale Model. **3**(4), 1–42 (2012)
Nguyen, T.V.P., Van Tol, A.F., Elkadi, A.S.K., Rohe, A.: Numerical investigation of pile installation effects in sand using material point method. Comput. Geotech. **73**, 58–71 (2016)
Nguyen, V.P., Nguyen, C.T., Rabczuk, T., Natarajan, S.: On a family of convicted particle domain interpolations in the material point method. Finite Elements Anal. Des. **126**, 50–64 (2017)
Nguyen, V.P., de Vaucorbeil, A., Nguyen-Thanh, C., Mandal, T.K., Kindal, T.: A generalized particle in cell method for explicit solid dynamics. Comput. Methods Appl. Mech. Eng. **360**, 112783 (2021). https://doi.org/10.1016/j.cma.2019.112783
Pan, X.F., Xu, A., Zhang, G., Zhu, J.: Generalized interpolation material point approach to high melting explosive with cavities under shock. J. Phys. D Appl. Phys. **41**(1), 015401 (2008)
Parvizian, Jamshid, Düster, Alexander, Rank, Ernst: Finite cell method. Comput. Mech. **41**(1), 121–133 (2007)
Peskin, C.S.: The immersed boundary method. Acta Numer. 11: 479–517 (2002)
Pinyol, N.M., Alvarado, M., Alonso, E.E., Zabala, F.: Thermal effects in landslide mobility. Géotechnique **68**(6), 528–545 (2017)
Powell, M.J.D., Sabin, M.A.: Piecewise quadratic approximations on triangles. ACM Trans. Math. Softw. (TOMS) **3**(4), 316–325 (1977)
Rabczuk, T.: Computational methods for fracture in brittle and quasi-brittle solids: state-of-the-art review and future perspectives. ISRN Applied Mathematics (2013). https://doi.org/10.1155/2013/849231. Article ID 849231, 38 pages
Raymond, S.J., Jones, B.D., Williams, J.R.: Modeling damage and plasticity in aggregates with the material point method (MPM). Comput. Part. Mech. **6**(3), 371–382 (2019)
Raymond, S.J., Jones, B., Williams, J.R.: A strategy to couple the material point method (MPM) and smoothed particle hydrodynamics (SPH) computational techniques. Comput. Particle Mech. **5**(1), 49–58 (2018)
Rots, Jan G.: Smeared and discrete representations of localized fracture. Int. J. Fract. **51**(1), 45–59 (1991)
Rots, J.G., Nauta, P., Kusters, G.M.A., Blaauwendraad, J.: Smeared crack approach and fracture localization in concrete. Heron **30**, 1–47 (1985)
Sabel, M., Sator, C., Müller, R.: A particle finite element method for machining simulations. Comput. Mech. **54**(1), 123–131 (2014)
Sadeghirad, A., Brannon, R.M., Burghardt, J.: A convicted particle domain interpolation technique to extend applicability of the material point method for problems involving massive deformations. Int. J. Numer. Meth. Eng. **86**(12), 1435–1456 (2011)
Sadeghirad, A., Brannon, R.M., Guilkey, J.E.: Second-order convicted particle domain interpolation (CPDI2) with enrichment for weak discontinuities at material interfaces. Int. J. Numer. Meth. Eng. **95**(11), 928–952 (2013)
Sanchez, J.: A Critical Evaluation of Computational Fracture Using a Smeared Crack Approach in MPM. Ph.D. thesis, University of New Mexico (2011)
Sanchez, J., Schreyer, H., Sulsky, D., Wallstedt, P.: Solving quasi-static equations with the material-point method. Int. J. Numer. Meth. Eng. **103**(1), 60–78 (2015)
Schillinger, D., Dede, L., Scott, M.A., Evans, J.A., Borden, M.J., Rank, E., Hughes, T.J.R.: An isogeometric design-through-analysis methodology based on adaptive hierarchical refinement of nurbs, immersed boundary methods, and t-spline cad surfaces. Comput. Methods Appl. Mech. Eng. **249**, 116–150 (2012)
Schmidt, B., Fraternali, F., Ortiz, M.: Eigenfracture: an eigendeformation approach to variational fracture. Multiscale Model. Simul. **7**(3), 1237–1266 (2009)
Scholtès, L., Donzé, F.V.: Modelling progressive failure in fractured rock masses using a 3D discrete element method. Int. J. Rock Mech. Min. Sci. **52**, 18–30 (2012)

Schreyer, H.L., Sulsky, D.L., Zhou, S.-J.: Modeling delamination as a strong discontinuity with the material point method. Comput. Methods Appl. Mech. Eng. **191**(23–24), 2483–2507 (2002)
Shen, L.: A rate-dependent damage/decohesion model for simulating glass fragmentation under impact using the material point method. Comput. Model. Eng. Sci. **49**(1), 23–45 (2009)
Shen, L., Chen, Z.: A multi-scale simulation of tungsten film delamination from silicon substrate. Int. J. Solids Struct. **42**(18–19), 5036–5056 (2005)
Sinaie, S., Nguyen, V.P., Nguyen, C.T., Bordas, S.: Programming the material point method in julia. Adv. Eng. Softw. **105**, 17–29 (2017)
Sinaie, S., Ngo, T.D., Nguyen, V.P.: A discrete element model of concrete for cyclic loading. Comput. Struct. **196**, 173–185 (2018)
Sinaie, S., Ngo, T.D., Kashani, A., Whittaker, A.S.: Simulation of cellular structures under large deformations using the material point method. Int. J. Impact Eng. **134**, 103385 (2019)
Soga, K., Alonso, E., Yerro, A., Kumar, K., Bandara, S.: Trends in large-deformation analysis of landslide mass movements with particular emphasis on the material point method. Géotechnique **66**(3), 248–273 (2015)
Steffen, M., Wallstedt, P.C., Guilkey, J.E., Kirby, R.M., Berzins, M.: Examination and analysis of implementation choices within the material point method (MPM). Comput. Model. Eng. Sci. **31**(2), 107–127 (2008)
Stomakhin, A., Schroeder, C., Jiang, C., Chai, L., Teran, J., Selle, A.: Augmented MPM for phase-change and varied materials. ACM Trans. Graph. **33**(4), 138:1-138:11 (2014a)
Stomakhin, A., Schroeder, C., Jiang, C., Chai, L., Teran, J., Selle, A.: Augmented MPM for phase-change and varied materials. ACM Trans. Graph. **33**(4), 138:1-138:11 (2014b)
Su, Y-C., Tao, J., Jiang, S., Chen, Z., Lu, J-M.: Study on the fully coupled thermodynamic fluid–structure interaction with the material point method. Computational Particle Mechanics, pp. 1–16 (2019)
Sukumar, N., Moran, B., Belytschko, T.: The natural element method in solid mechanics. Int. J. Numer. Methods Eng. **43**, 839–887 (1998)
Sulsky, D., Gong, M.: Improving the material-point method. In: Innovative Numerical Approaches for Multi-Field and Multi-Scale Problems, pp. 217–240. Springer (2016)
Sulsky, D., Kaul, A.: Implicit dynamics in the material-point method. Comput. Methods Appl. Mech. Eng. **193**(12–14), 1137–1170 (2004)
Sulsky, D., Schreyer, H.L.: Axisymmetric form of the material point method with applications to upsetting and Taylor impact problems. Comput. Methods Appl. Mech. Eng. **139**, 409–429 (1996)
Sulsky, D., Schreyer, L.: MPM simulation of dynamic material failure with a decohesion constitutive model. Eur. J. Mech. A. Solids **23**(3), 423–445 (2004)
Sulsky, D., Chen, Z., Schreyer, H.L.: A particle method for history-dependent materials. Comput. Methods Appl. Mech. Eng. **5**, 179–196 (1994)
Sulsky, D., Zhou, S.J., Schreyer, H.L.: Application of a particle-in-cell method to solid mechanics. Comput. Phys. Commun. **87**(1–2), 236–252 (1995)
Sulsky, D., Schreyer, H.L., Peterson, K., Kwok, R., Coon, M.: Using the material-point method to model sea ice dynamics. J. Geophys. Res. **112**(C2), C02S90 (2007)
Sun, Z., Huang, Z., Zhou, X.: Benchmarking the material point method for interaction problems between the free surface flow and elastic structure. Prog. Comput. Fluid Dyn. Int. J. **19**(1), 1–11 (2019)
Sun, Z., Li, H., Gan, H., Liu, H., Huang, Z., He, L.: Material point method and smoothed particle hydrodynamics simulations of fluid flow problems: a comparative study. Prog. Comput. Fluid Dyn. An Int. J. (PCFD) **18**(1), 1–18 (2018)
Sun, L., Mathur, S.R., Murthy, J.Y.: An unstructured finite-volume method for incompressible flows with complex immersed boundaries. Numer. Heat Transf. Part B: Fundam. **58**(4), 217–241 (2010)
Tan, H., Nairn, J.A.: Hierarchical, adaptive, material point method for dynamic energy release rate calculations. Comput. Methods Appl. Mech. Eng. **191**(19–20), 2123–2137 (2002)

Tao, J., Zhang, H., Zheng, Y., Chen, .Development of generalized interpolation material point method for simulating fully coupled thermomechanical failure evolution. Comput. Methods Appl. Mech. Eng. **332**, 325–342 (2018)

Tao, J. Zheng, Y., Chen, Z., Zhang, H.: Generalized interpolation material point method for coupled thermo-mechanical processes. Int. J. Mech. Mater. Des. **12**(4), 577–595 (2016)

Tielen, R., Wobbes, E., Möller, M., Beuth, L.: A high order material point method. Proc. Eng. **175**, 265–272 (2017)

Tran, Q-A., Sołowski, W.: Temporal and null-space filter for the material point method. Int. J. Numer. Methods Eng. (2019)

Tran, L.T., Kim, J., Berzins, M.: Solving time-dependent PDEs using the material point method, a case study from gas dynamics. Int. J. Numer. Meth. Fluids **62**(7), 709–732 (2010)

Vargas, M., Nascimento, E., Nascimento, G., Hotta, M., Almeida, M.: Comparative study of the material point method and smoothed particle hydrodynamics applied to the numerical simulation of a dam-break flow in the presence of geometric obstacles. Curr. J. Appl. Sci. Technol. (2018)

Wallstedt, P.C., Guilkey, J.E.: An evaluation of explicit time integration schemes for use with the generalized interpolation material point method. J. Comput. Phys. **227**(22), 9628–9642 (2008)

Wallstedt, P.C., Guilkey, J.E.: A weighted least squares particle-in-cell method for solid mechanics. Int. J. Numer. Meth. Eng. **85**(13), 1687–1704 (2011)

Wang, B., Karuppiah, V., Lu, H., Komanduri, R., Roy, S.: Two-dimensional mixed mode crack simulation using the material point method. Mech. Adv. Mater. Struct. **12**(6), 471–484 (2005)

Wang, Y., Beom, H.G., Sun, M., Lin, S.: Numerical simulation of explosive welding using the material point method. Int. J. Impact Eng. **38**(1), 51–60 (2011)

Wang, B., Vardon, P.J., Hicks, M.A., Chen, Z.: Development of an implicit material point method for geotechnical applications. Comput. Geotech. **71**, 159–167 (2016)

Wang, L., Coombs, W.M., Augarde, C.E., Cortis, M., Charlton, T.J., Brown, M.J., Knappett, J., Brennan, A., Davidson, C., Richards, D., et al.: On the use of domain-based material point methods for problems involving large distortion. Comput. Methods Appl. Mech. Eng. **355**, 1003–1025 (2019)

Weißenfels, C., Wriggers, P.: Stabilization algorithm for the optimal transportation meshfree approximation scheme. Comput. Methods Appl. Mech. Eng. **329**, 421–443 (2018)

Wie, Z., çkowski, Z., Sung-kie, Y., Jeoung-heum, Y.: A Particle-in-cell solution to the silo discharging problem. Int. J. Numer. Methods Eng. **45**, 1203–1225 (1999)

Wieçkowski, Z.: The material point method in large strain engineering problems. Comput. Methods Appl. Mech. Eng. **193**(39–41), 4417–4438 (2004)

Wobbes, E., Möller, M., Galavi, V., Vuik, C.: Conservative taylor least squares reconstruction with application to material point methods. Int. J. Numer. Meth. Eng. **117**(3), 271–290 (2019)

Wobbes, E., Tielen, R., Möller, M., Vuik, C.: Comparison and unification of material-point and optimal transportation meshfree methods. Comput. Part. Mech. 1–21 (2020)

Wu, J.Y., Nguyen, V.P., Nguyen, C.T., Sutula, D., Sinaie, S., Bordas, S.: Phase field modeling of fracture. Adv. Appl. Mech.: Fract. Mech.: Recent Dev. Trends **53**, submitted (2019)

Wu, J.Y.: A unified phase-field theory for the mechanics of damage and quasi-brittle failure in solids. J. Mech. Phys. Solids **103**, 72–99 (2017)

Wu, J.Y., Nguyen, V.P.: A length scale insensitive phase-field damage model for brittle fracture. J. Mech. Phys. Solids **119**, 20–42 (2018)

Xue, L., Borodin, O., Smith, G.D., Nairn, J.: Micromechanics simulations of the viscoelastic properties of highly filled composites by the material point method (MPM). Modell. Simul. Mater. Sci. Eng. **14**(4), 703 (2006)

Xue, L., Borodin, O., Smith, G.D.: Modeling of enhanced penetrant diffusion in nanoparticle-polymer composite membranes. J. Membr. Sci. **286**(1–2), 293–300 (2006)

Yang, P., Liu, Y., Zhang, X., Zhou, X., Zhao, Y.: Simulation of fragmentation with material point method based on Gurson model and random failure. Comput. Model. Eng. Sci. **85**(3), 207–236 (2012)

Yang, P., Gan, Y., Zhang, X., Chen, Z., Qi, W., Liu, P.: Improved decohesion modeling with the material point method for simulating crack evolution. Int. J. Fract. **186**(1–2), 177–184 (2014)
Yang, W.C., Arduino, P., Miller, G.R., Mackenzie-Helnwein, P.: Smoothing algorithm for stabilization of the material point method for fluid–solid interaction problems. Comput. Methods Appl. Mech. Eng. **342**, 177–199 (2018)
Ye, Z., Zhang, X., Zheng, G., Jia, G.: A material point method model and ballistic limit equation for hyper velocity impact of multi-layer fabric coated aluminum plate. Int. J. Mech. Mater. Des. **14**(4), 511–526 (2018)
Yerro, A., Soga, K., Bray, J.: Runout evaluation of oso landslide with the material point method. Can. Geotech. J. **999**, 1–14 (2018)
Yerro, A., Alonso, E.E., Pinyol, N.M.: The material point method for unsaturated soils. Géotechnique **65**(16), 201–217 (2015)
York, A.R., Sulsky, D., Schreyer, H.L.: Fluid-membrane interaction based on the material point method. Int. J. Numer. Methods Eng. 901–924 (2000)
York, A.R., Sulsky, D., Schreyer, H.L.: The material point method for simulation of thin membranes. Int. J. Numer. Meth. Eng. **44**(10), 1429–1456 (1999)
Yuanming, H., Fang, Y., Ge, Z., Ziyin, Q., Zhu, Y., Pradhana, A., Jiang, C.: A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. ACM Trans. Graph. (TOG) **37**(4), 150 (2018)
Yue, Y., Smith, B., Batty, C., Zheng, C., Grinspun, E.: A material point method for shear-dependent flows. ACM Trans. Graph. Contin. foam (2015)
Zhang, K., Shen, S-L., Zhou, A.: Dynamic brittle fracture with eigenerosion enhanced material point method. Int. J. Numer. Methods Eng. (2020)
Zhang, F., Zhang, X., Sze, K.Y., Lian, Y., Liu, Y.: Incompressible material point method for free surface flow. J. Comput. Phys. **330**, 92–110 (2017)
Zhang, X., Sze, K.Y., Ma, S.: An explicit material point finite element method for hyper-velocity impact. Int. J. Numer. Meth. Eng. **66**(4), 689–706 (2006)
Zhang, H.W., Wang, K.P., Chen, Z.: Material point method for dynamic analysis of saturated porous media under external contact/impact of solid bodies. Comput. Methods Appl. Mech. Eng. **198**(17–20), 1456–1472 (2009)
Zhang, D.Z., Ma, X., Giguere, P.T.: Material point method enhanced by modified gradient of shape function. J. Comput. Phys. **230**(16), 6379–6398 (2011)
Zhang, X., Chen, Z., Liu, Y.: The Material Point Method: A Continuum-Based Particle Method for Extreme Loading Cases. Academic (2016b)
Zhao, X., Liang, D., Martinelli, M.: MPM simulations of dam-break floods. J. Hydrodyn. **29**(3), 397–404 (2017)
Zheng, Y., Gao, F., Zhang, H., Lu, M.: Improved convicted particle domain interpolation method for coupled dynamic analysis of fully saturated porous media involving large deformation. Comput. Methods Appl. Mech. Eng. **257**, 150–163 (2013)
Zhou, S., Stormont, J., Chen, Z.: Simulation of geomembrane response to settlement in landfills by using the material point method. Int. J. Numer. Anal. Meth. Geomech. **23**(15), 1977–1994 (1999)
Zhou, S., Zhang, X., Ma, H.: Numerical simulation of human head impact using the material point method. Int. J. Comput. Methods **10**(04), 1350014 (2013)
Zhu, Y., Bridson, R.: Animating sand as a fluid. ACM Trans. Graph. **24**(3), 965–972 (2005)

Chapter 2
A General MPM for Solid Mechanics

This chapter presents a general formulation of the MPM for solid mechanics. This formulation applies to all existing MPM variants such as GIMP, CPDI and BSMPM. Here, even though only explicit dynamics MPM is presented in great details, we also touch briefly on implicit dynamics and quasi-static MPM. Moreover, both updated and total Lagrangian MPM are treated.

We start with a short review of continuum mechanics in Sect. 2.1. Next, governing equations using the updated Lagrangian description written in a strong form are stated in Sect. 2.2. The weak form corresponding to the equation of motion is given in Sect. 2.3. Also, the MPM spatial discretization procedure of this weak form is treated. The obtained semi-discrete equations can also be derived from the updated Lagrangian finite element weak form discretization considering the finite elements' quadrature points as material points as shown in Sect. 2.4. The semi-discrete equations are ordinary differential equations (ODEs) in which time is still a continuous variable. One needs to discretize time to get algebraic equations. Time discretization of the semi-discrete equations is presented in Sect. 2.5. In this section, various MPM algorithms such as Updated Stress Last (USL) and Modified Updated Stress Last (MUSL) are also discussed.

In Sect. 2.6, we discuss the algorithm of the total Lagrangian MPM. Axisymmetric formulations of ULMPM and TLMPM are given in Sect. 2.7 and adaptive stable time steps are presented in Sect. 2.8. Regarding the stability of the MPM, we often encounter the negative Jacobian issue due to element inversion and we examine this in Sect. 2.9. Finally, grid adaptivity and particle adaptivity are briefly discussed in Sect. 2.10.

V. P. Nguyen et al., *The Material Point Method*, Scientific Computation,
https://doi.org/10.1007/978-3-031-24070-6_2

2.1 Basic Concepts of Continuum Mechanics

Continuum mechanics is a theory that models solids and fluids at a macroscopic scale which ignores inhomogeneities such as molecular, granular, or crystal structures. Therefore, within this theory, the behavior of solids and fluids can be characterized by smooth functions of spatial variables. The subject of continuum mechanics comprises the following basic topics: (1) the study of motion and deformation without considering the causes (kinematics), (2) the study of internal forces (kinetics), (3) the conservation equations or balance principles that state that there are certain important physical properties e.g. mass, momentum and energies that must be conserved, and (4) constitutive models that furnish the relationship between kinematics and kinetics variables. It is via a constitutive model that, in continuum mechanics, one differentiate a solid from a fluid, a rubber from a rock, etc.

This section reviews the key concepts and equations of continuum mechanics. Good knowledge of continuum mechanics is essential for the understanding of the MPM as it is a continuum-based numerical method. Derivations are not presented, but relevant literature is given. For more details, we refer to standard textbooks such as Malvern (1969), Gurtin (1981), Marsden and Hughes (1983), Ogden (1984), Holzapfel (2000).

2.1.1 Motion and Deformation

In continuum mechanics, a body $\mathcal{B}$ is considered as being formed by an infinite set of material points, which are endowed with certain mechanical properties. The position vector of a material point in the initial, undeformed configuration of the body is denoted $\mathbf{X}$ relative to some coordinate basis. $\mathbf{X}$ is named the *material or Lagrangian coordinate*. The position of the same material point, in the deformed configuration, is designated by $\mathbf{x}$ called *spatial or Eulerian coordinate*.

The motion (deformation) of a solid is described by a function $\boldsymbol{\phi}(\mathbf{X}, t)$. A relation between spatial coordinates and material coordinates can be established as follows

$$\mathbf{x} = \boldsymbol{\phi}(\mathbf{X}, t) \tag{2.1}$$

The displacement, velocity and acceleration fields of a body are the primary kinematical fields in describing the motion of the body. The displacement of a material point $\mathbf{X}$, denoted by $\mathbf{u}(\mathbf{X}, t)$, is the difference between its current position $\phi(\mathbf{X}, t)$ and its initial position $\phi(\mathbf{X}, 0)$. So,

$$\mathbf{u}(\mathbf{X}, t) := \boldsymbol{\phi}(\mathbf{X}, t) - \boldsymbol{\phi}(\mathbf{X}, 0) = \mathbf{x} - \mathbf{X} \tag{2.2}$$

The velocity of a material point $\mathbf{X}$, denoted by $\mathbf{v}(\mathbf{X}, t)$, is defined as the rate of change of position of this material point, that is

$$\mathbf{v}(\mathbf{X}, t) := \frac{\partial \boldsymbol{\phi}(\mathbf{X}, t)}{\partial t} \tag{2.3}$$

This is the Lagrangian velocity field. There exists a Eulerian form for this velocity field but as the MPM adopts a Lagrangian description, it is not discussed here.

The acceleration of a material point $\mathbf{X}$ is the rate of change of its velocity, i.e., the material time derivative of the velocity,

$$\mathbf{a}(\mathbf{X}, t) := \frac{\partial \mathbf{v}(\mathbf{X}, t)}{\partial t} = \frac{\partial^2 \boldsymbol{\phi}(\mathbf{X}, t)}{\partial t^2} \tag{2.4}$$

The deformation gradient tensor $\mathbf{F}$ is a key quantity in finite deformation continuum mechanics as all deformation quantities are derived from it. It is a linear mapping operator which maps each infinitesimal linear element $d\mathbf{X}$ in the reference configuration into an infinitesimal linear element $d\mathbf{x}$ in the current configuration. It is defined as:

$$\mathbf{F} := \frac{\partial \boldsymbol{\phi}}{\partial \mathbf{X}} = \frac{\partial \mathbf{x}}{\partial \mathbf{X}} \quad \text{or} \quad F_{ij} = \frac{\partial x_i}{\partial X_j} \tag{2.5}$$

Next, the concept of *material time derivative* is introduced. To understand this important concept, consider the following situation. Assume we have a certain field φ (scalar, vectorial or tensorial) defined over the body for which we want to know the rate of change, at a given material point $\mathbf{X}$. This is known as the material time derivative of φ. There are two definition of this concept, corresponding to material and spatial descriptions, respectively:

1. Lagrangian description. In the Lagrangian description, the independent variables are the material coordinates $\mathbf{X}$ and time t. So all we have to do is taking the partial derivative of the given field φ with respect to time. For a material field $\varphi(\mathbf{X}, t)$, its material time derivative is

$$\frac{D\varphi(\mathbf{X}, t)}{Dt} \equiv \dot{\varphi} = \frac{\partial \varphi(\mathbf{X}, t)}{\partial t} \tag{2.6}$$

   where the first two equations indicate standard notation for the material time derivative. As the MPM adopts a Lagrangian description, the material time derivative is very simple.
2. Eulerian description. The considered field is $\varphi(\mathbf{x}, t)$. This case is much more complicated since not only time changes but also the spatial position $\mathbf{x}$ of the considered particle. We must calculate the partial derivative with respect to time of the material description of $\varphi(\mathbf{x}, t)$, keeping $\mathbf{X}$ fixed.

$$\frac{D\varphi(\mathbf{x}, t)}{Dt} \equiv \dot{\varphi} := \lim_{\Delta t \to 0} \frac{\varphi(\boldsymbol{\phi}(\mathbf{X}, t + \Delta t), t + \Delta t) - \varphi(\boldsymbol{\phi}(\mathbf{X}, t), t)}{\Delta t} \tag{2.7}$$

   Using the chain rule, we obtain the important formula of the material time derivative for a Eulerian scalar field:

$$\frac{D\varphi(\mathbf{x},t)}{Dt} = \frac{\partial\varphi(\mathbf{x},t)}{\partial t} + \nabla\varphi(\mathbf{x},t)\cdot\mathbf{v}(\mathbf{x},t) \tag{2.8}$$

The term $\partial\varphi/\partial t$ is called the *spatial time derivative*, and the term $\varphi_{,j}v_j$ is the convective term, which is also called the transport term.

2.1.2 Strain Measures

There are many measures of strain and strain rate in nonlinear continuum mechanics. A strain measure must vanish for any rigid body motion, and in particular for rigid body rotation. Herein, we review some strain measures commonly adopted in nonlinear continuum mechanics. They are the right Cauchy-Green deformation tensor $\mathbf{C}$, the Green strain tensor $\mathbf{E}$, and the rate of deformation tensor $\mathbf{D}$.

The right Cauchy-Green deformation tensor is written as

$$\mathbf{C} := \mathbf{F}^{\mathrm{T}}\cdot\mathbf{F}; \quad C_{ij} := F_{ki}F_{kj} \tag{2.9}$$

where the superscript T denotes the transpose operator. The Green strain tensor is given by

$$\mathbf{E} := \frac{1}{2}(\mathbf{C}-\mathbf{I}) = \frac{1}{2}(\mathbf{F}^{\mathrm{T}}\cdot\mathbf{F}-\mathbf{I}), \quad E_{ij} = \frac{1}{2}(F_{ki}F_{kj}-\delta_{ij}) \tag{2.10}$$

This strain tensor measures the difference of the square of the length of $d\mathbf{x}$ and $d\mathbf{X}$.

The spatial gradient of velocity or *velocity gradient tensor* $\mathbf{L}$ is defined as the spatial gradient of the velocity, that is

$$\mathbf{L}(\mathbf{x},t) := \frac{\partial\mathbf{v}}{\partial\mathbf{x}}, \quad \text{or} \quad L_{ij} = \frac{\partial v_i}{\partial x_j} \tag{2.11}$$

The velocity gradient $\mathbf{L}$ allows the material time derivative of the deformation gradient $\mathbf{F}$ to be written as

$$\dot{\mathbf{F}} = \frac{\partial}{\partial t}\left(\frac{\partial\phi(\mathbf{X},t)}{\partial\mathbf{X}}\right) = \frac{\partial\mathbf{v}}{\partial\mathbf{X}} = \frac{\partial\mathbf{v}}{\partial\mathbf{x}}\cdot\frac{\partial\mathbf{x}}{\partial\mathbf{X}} = \mathbf{L}\cdot\mathbf{F}, \quad \Rightarrow \mathbf{L} = \dot{\mathbf{F}}\cdot\mathbf{F}^{-1} \tag{2.12}$$

where in the second equality, we have used the fact that material time derivative of Lagrangian fields commute with material gradient. Noting that, this fact does not hold generally for Eulerian fields.

The velocity gradient tensor $\mathbf{L}$ can be decomposed into symmetric and skew-symmetric parts by

$$\mathbf{L} = \frac{1}{2}(\mathbf{L} + \mathbf{L}^{\mathrm{T}}) + \frac{1}{2}(\mathbf{L} - \mathbf{L}^{\mathrm{T}}) \tag{2.13}$$

which is a standard decomposition of a second-order tensor. The rate of deformation tensor $\mathbf{D}$ is defined as the symmetric part of $\mathbf{L}$, and the spin tensor as the skew-symmetric part. Using these definitions, one can write

$$\mathbf{D} := \frac{1}{2}(\mathbf{L} + \mathbf{L}^{\mathrm{T}}), \quad \mathbf{W} := \frac{1}{2}(\mathbf{L} - \mathbf{L}^{\mathrm{T}}) \tag{2.14}$$

where $\mathbf{D}$ describes the rate of stretching and shearing.

2.1.3 Stress Measures

As there are different strain measures there also exist different stress measures which are work conjugated with them. The most commonly used stress tensors are (1) Cauchy stress, (2) Kirchhoff stress, (3) first Piola-Kirchhoff stress (1st PK) and (4) second Piola-Kirchhoff stress (2nd PK). Relation between these stress tensors is given in Table 2.1. The Cauchy stress is the true stress and work conjugate with the rate of deformation $\mathbf{D}$ with respect to the deformed volume, cf. Eq. (2.19). The Kirchhoff stress—also referred to as the weighted Cauchy stress—is work conjugate with the rate of deformation tensor with respect to the initial volume. The 1st PK stress, which is not symmetric, is work conjugate to the rate of deformation gradient. The 2nd Piola-Kirchhoff stress, a totally material symmetric stress tensor, is work conjugate to the Green strain rate tensor. Note that, some authors e.g. Belytschko et al. (2000), prefer to work with the nominal stress, the transpose of which is the 1st PK stress.

Table 2.1 Relation between different stress measures

	Cauchy stress $\boldsymbol{\sigma}$	Kirchhoff stress $\boldsymbol{\tau}$	1st PK $\mathbf{P}$	2nd PK $\mathbf{S}$
$\boldsymbol{\sigma}$	–	$\boldsymbol{\tau} J^{-1}$	$J^{-1}\mathbf{P}\mathbf{F}^{\mathrm{T}}$	$J^{-1}\mathbf{F}\mathbf{S}\mathbf{F}^{\mathrm{T}}$
$\boldsymbol{\tau}$	$J\boldsymbol{\sigma}$	–	$\mathbf{P}\mathbf{F}^{\mathrm{T}}$	$\mathbf{F}\mathbf{S}\mathbf{F}^{\mathrm{T}}$
$\mathbf{P}$	$J\boldsymbol{\sigma}\mathbf{F}^{-\mathrm{T}}$	$\boldsymbol{\tau}\mathbf{F}^{-\mathrm{T}}$	–	$\mathbf{F}\mathbf{S}$
$\mathbf{S}$	$J\mathbf{F}^{-1}\boldsymbol{\sigma}\mathbf{F}^{-\mathrm{T}}$	$\mathbf{F}^{-1}\boldsymbol{\tau}\mathbf{F}^{-\mathrm{T}}$	$\mathbf{F}^{-1}\mathbf{P}$	–

2.1.4 Objective Stress Rates

Constitutive equations are often written in a rate form i.e., a relation between a stress rate and a deformation rate. Under large rotations, simply using the material derivatives of the stress tensors e.g. the rate of Cauchy stress $D\boldsymbol{\sigma}/Dt$ is wrong as it does not transform properly as a tensor under a superposed rigid body motion. We discuss three objective stress rates: the Jaumman rate, the Truesdell rate and the Green-Naghdi rate which are frequently used in practice. A constitutive model can be formulated in terms of any one of these objective stress rates, and changing from one rate to another requires that the constitutive model be reformulated.

They are collectively given in Eq. (2.15).

$$\begin{aligned}
\boldsymbol{\sigma}^{\nabla T} &= \frac{D\boldsymbol{\sigma}}{Dt} + \operatorname{div}(\mathbf{v})\boldsymbol{\sigma} - \mathbf{L}\cdot\boldsymbol{\sigma} - \boldsymbol{\sigma}\cdot\mathbf{L}^{\mathrm{T}} && \text{Truesdel rate} \\
\boldsymbol{\sigma}^{\nabla J} &= \frac{D\boldsymbol{\sigma}}{Dt} - \mathbf{W}\cdot\boldsymbol{\sigma} - \boldsymbol{\sigma}\cdot\mathbf{W}^{\mathrm{T}} && \text{Jaumman rate} \\
\boldsymbol{\sigma}^{\nabla G} &= \frac{D\boldsymbol{\sigma}}{Dt} - \boldsymbol{\Omega}\cdot\boldsymbol{\sigma} - \boldsymbol{\sigma}\cdot\boldsymbol{\Omega}^{\mathrm{T}} && \text{Green-Naghdi rate}
\end{aligned} \tag{2.15}$$

A discussion on which objective stress rate to be used was given in Benson (1992).

2.1.5 Conservation Equations

An important set of equations in continuum mechanics are the conservation equations or balance equations . For thermomechanical systems, the conservation laws include

1. Conservation of mass
2. Conservation of linear momentum
3. Conservation of angular momentum, and
4. Conservation of energy

Conservation of mass. The law of conservation of mass is described by the *equation of mass conservation*, or often called *equation of continuity* written as:

$$\frac{D\rho}{Dt} + \rho\nabla\cdot\mathbf{v} \quad \text{or} \quad \dot{\rho} + \rho v_{i,i} = 0 \tag{2.16}$$

If the density does not change, i.e., the material is incompressible, hence the material time derivative of the density vanishes, and the continuity equation becomes $v_{i,i} = 0$ which is the well known incompressibility condition. For Lagrangian description, a simpler algebraic equation for the mass conservation is given by

$$\rho J = \rho_0 \tag{2.17}$$

Conservation of linear momentum. This law demands that the change of linear momentum in time is equal to the sum of all external forces (volume and surface forces) acting on the body. It is described by the so-called *the momentum equation*:

$$\rho \frac{D\mathbf{v}}{Dt} = \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{b} \quad \text{or} \quad \rho \dot{v}_i = \sigma_{ji,j} + \rho b_i \tag{2.18}$$

Conservation of angular momentum. This laws requires that the Cauchy stress be a symmetric tensor.

Conservation of energy. This law states that the rate of change of the total energy in the body (consisting of the internal energy and kinetic energy) is equal to the rate of work done by the external forces plus the rate of work provided by heat flux $\mathbf{q}$ and energy sources.

$$\rho \frac{De}{Dt} = \mathbf{D} : \boldsymbol{\sigma} - \nabla \cdot \mathbf{q} + \rho s \tag{2.19}$$

where e is the specific internal energy; ρs indicates a heat source per unit volume.

2.1.6 Constitutive Models

All the equations given previously are material independent: they are valid for both solids and fluids. To model a material behavior, a *constitutive equation* or *constitutive relation*—a relation between kinetic quantities (e.g. stresses) as related to kinematic quantities (e.g. strains)—is needed. It is through constitutive equations that one can differentiate fluids from solids, concretes from rubbers etc. The first constitutive equation (constitutive law) was developed by Robert Hooke and is known as Hooke's law. It deals with the case of linear elastic materials. Since then, a plethora of constitutive models has been developed to characterize a diverse range of natural and engineering materials (Gurtin 1981; Marsden and Hughes 1983; Ogden 1984). For the sake of completeness, we summarize in Chap. 4 some commonly used constitutive models for elastic and plastic solids. It is worth noting that numerical simulations can only be as accurate as the utilized material models.

2.2 Strong Form

For a continuum body under purely mechanical loading (neglect heat exchange), the governing differential equations in an updated Lagrangian description include balance laws, constitutive equation, kinematics equation and boundary/initial conditions, which are collectively given as

$$
\begin{aligned}
&\frac{D\rho}{Dt} + \rho \nabla \cdot \mathbf{v} = 0 && \text{(conservation of mass)} \\
&\rho \frac{D\mathbf{v}}{Dt} = \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{b} && \text{(conservation of linear momentum)} \\
&\rho \frac{De}{Dt} = \mathbf{D} : \boldsymbol{\sigma} && \text{(conservation of energy)} \\
&\mathbf{D} = \text{sym}(\nabla \mathbf{v}) && \text{(strain measure)} \\
&\boldsymbol{\sigma}^{\nabla} = S_t^{\sigma D}(\mathbf{D}, \boldsymbol{\sigma}) && \text{(constitutive equation)} \\
&\mathbf{v}(\mathbf{X}, t = 0) = \mathbf{v}_0, \ \boldsymbol{\sigma}(\mathbf{x}, t = 0) = \boldsymbol{\sigma}_0 && \text{(initial conditions)} \\
&\mathbf{u} = \bar{\mathbf{u}} \text{ on } \Gamma_u && \text{(Dirichlet boundary conditions)} \\
&\mathbf{t} = \bar{\mathbf{t}} \text{ on } \Gamma_t && \text{(Neumann boundary conditions)}
\end{aligned}
\tag{2.20}
$$

where $\rho(\mathbf{X}, t)$ is the density, $\mathbf{v}(\mathbf{X}, t)$ denotes the velocity, $\boldsymbol{\sigma}(\mathbf{X}, t)$ is the Cauchy stress tensor, $\mathbf{b}$ is the specific body force and ∇ is the gradient operator with respect to the current configuration. The rate of deformation tensor is represented by $\mathbf{D}(\mathbf{X}, t)$, $\boldsymbol{\sigma}^{\nabla}$ denotes some objective stress rates which are necessary for large rotations.

The conservation of energy equation is used to update the internal energy for the equation of state and to check the energy conservation. As a Lagrangian description is used in MPM, conservation of mass, first equation in Eq. (2.20) is not solved. The formulation of the basic MPM is isothermal and thus, the energy equation is not solved either. We postpone the treatment of temperature effects to Sect. 10.3 in Chap. 10.

Known quantities include prescribed displacements $\bar{\mathbf{u}}$ (or equivalently prescribed velocities) on the Dirichlet boundary Γ_u, prescribed tractions $\bar{\mathbf{t}}$ on the traction boundary Γ_t, initial velocities $\mathbf{v}_0$ and initial stresses $\boldsymbol{\sigma}_0$. Recall that in a Lagrangian formulation, the material time derivative is simply a partial derivative with respect to time $\frac{D\varphi(\mathbf{X},t)}{Dt} \equiv \dot{\varphi} = \frac{\partial \varphi(\mathbf{X},t)}{\partial t}$. This is much simpler than the Eulerian formulation of the material time derivative.

As mentioned above, the independent variables in Lagrangian formulation are the material coordinate $\mathbf{X}$ and time t. The dependent variables include (i) mass density $\rho(\mathbf{X}, t)$ (one unknown), (ii) velocity field[1] $\mathbf{v}$ (3 unknowns in 3D), (iii) stress field $\boldsymbol{\sigma}$ (6 unknowns as Cauchy stress tensor is symmetric) and (iv) the deformation rate tensor $\mathbf{D}$ (6 unknowns). Therefore, in total, we have 16 unknowns in three dimensions. The governing equations include (i) conservation of mass (1 equation), (ii) conservation of momenta (3 equations), (iii) conservation of energy (1 equation), (iv) constitutive equations (6 equations that relate the six stress components to the six components of the deformation rate tensor) and (v) strain-displacement equations (6 equations). In total, we have 17 equations. However, since we are interested in non-adiabatic, non-isothermal processes, the conservation of energy is not a PDE, so finally we have 16 equations for 16 unknowns. Note that the conservation of energy is needed in the so-called equation of state that relates pressure, volume and specific energy.

[1] Having the velocity one can obtain the displacement and acceleration fields.

2.3 Weak Form and Spatial Discretization

The MPM, which can be considered as an updated Lagrangian FEM, also employs a weak formulation. The weak form of the momentum equation, the second equation in Eq. (2.20), is given by see e.g. Belytschko et al. (2000) or Appendix A.1

$$\int_{\Omega} \rho \delta u_i a_i \mathrm{d}\Omega + \int_{\Omega} \rho \frac{\partial \delta u_i}{\partial x_j} \sigma_{ij}^s \mathrm{d}\Omega = \int_{\Omega} \rho \delta u_i b_i \mathrm{d}\Omega + \int_{\Gamma_t} \rho \delta u_i \bar{t}_i^s \mathrm{d}\Gamma \tag{2.21}$$

where Ω denotes the current configuration, σ_{ij}^s is the specific Cauchy stresses i.e., $\sigma_{ij}^s = \sigma_{ij}/\rho$; subscripts $i, j = 1, 2, 3$ are used to denote components of vectors and tensors; $\delta\mathbf{u}$ is the virtual displacement field or the test functions; $\mathbf{a}$ the acceleration field. The specific traction vector is denoted by $\bar{t}_i^s$. Note that the above was written in indicial notation which will facilitate the derivation of the discrete equations. From this weak form, one can proceed as Sulsky et al. (1994) in what follows to obtain the semi-discrete equations for the ULMPM or one can derive the semi-discrete equations for MPM from the ones of the ULFEM by considering the particles as quadrature points, see Sect. 2.4.

The whole material domain Ω is discretized by a set of material sub-domains Ω_p, and it is assumed that the whole mass of a material sub-domain is concentrated at the corresponding material point, which means that the mass density field is expressed as

$$\rho(\mathbf{x}, t) = \sum_{p=1}^{n_p} m_p \delta(\mathbf{x} - \mathbf{x}_p) \tag{2.22}$$

where δ is the Dirac delta function with dimension of the inverse of volume. Note that m_p denotes the mass of particle p. Substitution of Eq. (2.22) into Eq. (2.21) results in

$$\sum_{p=1}^{n_p} m_p \delta u_i(\mathbf{x}_p) a_i(\mathbf{x}_p) + \sum_{p=1}^{n_p} m_p \left.\frac{\partial \delta u_i}{\partial x_j}\right|_{(\mathbf{x}_p)} \sigma_{ij}^s(\mathbf{x}_p) = \sum_{p=1}^{n_p} m_p \delta u_i(\mathbf{x}_p) b_i(\mathbf{x}_p) + \sum_{p=1}^{n_p} m_p \delta u_i(\mathbf{x}_p) \bar{t}_i^s(\mathbf{x}_p) h^{-1} \tag{2.23}$$

where use was made of the identity $\int f(\mathbf{x})\delta(\mathbf{x} - \mathbf{x}_p) = f(\mathbf{x}_p)$. As particles constitute a volume, one needs to introduce a boundary layer thickness h in the calculation of the external force due to the traction (Zhang et al. 2016). We postpone the discussion of external traction to Sect. 5.2. Briefly, as a particle method of which there lacks an explicit representation of the solid boundary, it is difficult to apply a Neumann boundary condition in the MPM.

Next, the space is discretized by a finite element mesh (or a grid) so that any spatially varying field can be approximated. The mesh consists of n_n nodes with shape functions ϕ_I associated with each node I; $x_{iI}(t)$ is the i component of the position vector of node I. In 3D, one writes $\mathbf{x}_I = (x_{1I}, x_{2I}, x_{3I}) = (x_I, y_I, z_I)$. Subscript I

denotes the value of grid nodes, and subscript p denotes the value of particles. Thanks to the use of a FE mesh, evaluation of shape functions and derivatives are standard (thus efficient) and does not involve neighbor search as in other meshfree methods such as SPH or EFG.

The FE approximation of the motion is given by

$$x_i(\mathbf{X}, t) = \sum_{I=1}^{n_n} \phi_I(\mathbf{X}) x_{iI}(t) \tag{2.24}$$

As can be seen, the shape functions are expressed in terms of the Lagrangian coordinates $\mathbf{X}$ not the Eulerian coordinates $\mathbf{x}$, similar to Lagrangian finite elements (Belytschko et al. 2000).

Using Eq. (2.24) for the initial configuration one writes

$$X_i = \sum_{I=1}^{n_n} \phi_I(\mathbf{X}) X_{iI} \tag{2.25}$$

with X_{iI} is the i component of $\mathbf{X}_I$–the coordinates of node I in the initial configuration.

The displacement is thus approximated as

$$u_i = x_i - X_i = \sum_{I=1}^{n_n} \phi_I(\mathbf{X})(x_{iI} - X_{iI}) = \sum_{I=1}^{n_n} \phi_I(\mathbf{X}) u_{iI}(t) \tag{2.26}$$

where u_{iI} designates the i component of the displacement vector of node I.

The velocity and acceleration fields are thus given by

$$v_i(\mathbf{X}, t) = \sum_{I=1}^{n_n} \phi_I(\mathbf{X}) v_{iI}(t) \tag{2.27}$$

and

$$a_i(\mathbf{X}, t) = \sum_{I=1}^{n_n} \phi_I(\mathbf{X}) a_{iI}(t) \tag{2.28}$$

where v_{iI}, a_{iI} are the i component of the velocity and acceleration vectors of node I, respectively. Note that Eq. (2.27) is not needed in the derivation of the semi-discrete equations given in this section, it will be used later, for example to compute the velocity gradient and update the particle position.

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

$$\delta u_i(\mathbf{X}) = \sum_{I=1}^{n_n} \phi_I(\mathbf{X}) \delta u_{iI} \tag{2.29}$$

i.e., the virtual displacement field is approximated using the same shape functions. The spatial derivatives of δu_i is thus given by

$$\frac{\partial \delta u_i}{\partial x_j} = \sum_{I=1}^{n_n} \frac{\partial \phi_I}{\partial x_j} \delta u_{iI} \tag{2.30}$$

Substituting the FE approximations of δu_i, a_i and $\frac{\partial \delta u_i}{\partial x_j}$ evaluated at the particles using Eqs. (2.28)–(2.30) into Eq. (2.23) leads to

$$\sum_{p=1}^{n_p} m_p \left[\sum_{I=1}^{n_n} \phi_I(\mathbf{x}_p)\delta u_{iI}\right]\left[\sum_{J=1}^{n_n} \phi_J(\mathbf{x}_p) a_{iJ}\right] + \sum_{p=1}^{n_p} m_p \left[\sum_{I=1}^{n_n} \left.\frac{\partial \phi_I}{\partial x_j}\right|_{(\mathbf{x}_p)} \delta u_{iI}\right] \sigma_{ij}^s(\mathbf{x}_p) =$$
$$\sum_{p=1}^{n_p} m_p \left[\sum_{I=1}^{n_n} \phi_I(\mathbf{x}_p)\delta u_{iI}\right] b_i(\mathbf{x}_p) + \sum_{p=1}^{n_p} m_p \left[\sum_{I=1}^{n_n} \phi_I(\mathbf{x}_p)\delta u_{iI}\right] \bar{t}_i^s(\mathbf{x}_p) h^{-1} \tag{2.31}$$

As δu_{iI} are arbitrary,[2] we obtain the following set of equations (1, 2, 3 equations for each node I, $I = 1, \ldots, n_n$ for 1D, 2D, 3D, respectively)

$$\sum_{p=1}^{n_p} m_p \phi_I(\mathbf{x}_p)\left(\sum_{J=1}^{n_n} \phi_J(\mathbf{x}_p) a_{iJ}\right) + \sum_{p=1}^{n_p} m_p \left.\frac{\partial \phi_I}{\partial x_j}\right|_{(\mathbf{x}_p)} \sigma_{ij}^s(\mathbf{x}_p) =$$
$$\sum_{p=1}^{n_p} m_p \phi_I(\mathbf{x}_p) b_i(\mathbf{x}_p) + \sum_{p=1}^{n_p} m_p \phi_I(\mathbf{x}_p) \bar{t}_i^s(\mathbf{x}_p) h^{-1} \tag{2.32}$$

which can be written in the following compact form

$$m_{IJ}\mathbf{a}_J = \mathbf{f}_I^{\text{ext}} + \mathbf{f}_I^{\text{int}}, \quad I = 1, 2, \ldots, n_n \tag{2.33}$$

where $m_{IJ}, \mathbf{f}_I^{\text{ext}}, \mathbf{f}_I^{\text{int}}$ are the IJ component of the consistent mass matrix, the external force vector and the internal force vector, respectively. This equation is exactly identical to that of the FEM. Equation (2.33) is usually referred to as the semi-discrete equation as just space was discretized.

The IJ component of the consistent mass matrix is given by

$$m_{IJ} = \sum_{p=1}^{n_p} m_p \phi_I(\mathbf{x}_p)\phi_J(\mathbf{x}_p) \tag{2.34}$$

Note that the mass matrix is not constant as in the FEM but changes in time because the material points move while the grid nodes are reset after a time step.

[2] Boundary conditions will be imposed on the discrete equations later.

The external force vector is written as

$$\mathbf{f}_I^{\text{ext}} = \sum_{p=1}^{n_p} m_p \phi_I(\mathbf{x}_p)\mathbf{b}(\mathbf{x}_p) + \sum_{p=1}^{n_p} m_p \phi_I(\mathbf{x}_p)\bar{\mathbf{t}}^s(\mathbf{x}_p)h^{-1} \tag{2.35}$$

and the internal force vector as

$$\mathbf{f}_I^{\text{int}} = -\sum_{p=1}^{n_p} m_p/\rho_p \boldsymbol{\sigma}_p \nabla\phi_I(\mathbf{x}_p) = -\sum_{p=1}^{n_p} V_p \boldsymbol{\sigma}_p \nabla\phi_I(\mathbf{x}_p) \tag{2.36}$$

where $\nabla\phi_I = \left({}^{\partial\phi_I}/_{\partial x_1}, {}^{\partial\phi_I}/_{\partial x_2}, {}^{\partial\phi_I}/_{\partial x_3}\right)^{\mathrm{T}}$ denotes the gradient of the shape function; V_p is the volume of particle p; $\boldsymbol{\sigma}_p$ is the 3×3 Cauchy stress matrix of particle p. Recall that the particle density is defined as the ratio of the particle mass to particle volume. Note that the definition of the nodal internal force is slightly different from the one in the nonlinear FE literature – there is no minus sign in FEM formulation. We decided to be consistent with the MPM notation so that confusions would not occur for beginners.

Remark 10 To understand the compact expression for the internal force given in Eq. (2.36), we write the internal force vector explicitly as follows

$$\begin{aligned}
f_{xI}^{\text{int}} &= -\sum_{p=1}^{n_p} V_p \left[(\sigma_{xx})_p \frac{\partial\phi_I}{\partial x}(\mathbf{x}_p) + (\sigma_{xy})_p \frac{\partial\phi_I}{\partial y}(\mathbf{x}_p) + (\sigma_{xz})_p \frac{\partial\phi_I}{\partial z}(\mathbf{x}_p) \right] \\
f_{yI}^{\text{int}} &= -\sum_{p=1}^{n_p} V_p \left[(\sigma_{xy})_p \frac{\partial\phi_I}{\partial x}(\mathbf{x}_p) + (\sigma_{yy})_p \frac{\partial\phi_I}{\partial y}(\mathbf{x}_p) + (\sigma_{yz})_p \frac{\partial\phi_I}{\partial z}(\mathbf{x}_p) \right] \\
f_{zI}^{\text{int}} &= -\sum_{p=1}^{n_p} V_p \left[(\sigma_{zx})_p \frac{\partial\phi_I}{\partial x}(\mathbf{x}_p) + (\sigma_{zy})_p \frac{\partial\phi_I}{\partial y}(\mathbf{x}_p) + (\sigma_{zz})_p \frac{\partial\phi_I}{\partial z}(\mathbf{x}_p) \right]
\end{aligned} \tag{2.37}$$

where we used the second term in Eq. (2.32) for f_{iI}^{int}. Simplifications for 1D and 2D follow straightforwardly.

Remark 11 Since the Neumann boundary Γ_t where the traction is prescribed is not accurately defined in MPM, it is difficult to compute the external force due to non-zero traction. And this difficulty applies to the enforcement of non-zero heat flux in thermal and thermo-mechanical analysis. We refer to Sect. 5.2.3 for a discussion on this topic.

We have presented the derivation of Eq. (2.33), the semi-discrete equation following the MPM way as Sulsky et al. (1994) did. In the next section, another derivation is provided to see the link to the updated Lagrangian finite element method.

2.4 MPM as FEM with Particles as Integration Points

In what follows, we show that the MPM semi-discrete equation can be obtained directly from the updated Lagrangian FE equations by considering the particles as the integration points. This derivation usually seems to be the most straightforward for readers with experiences in the FEM. More importantly, it shows one major drawback of the standard MPM: the quadrature approximation nature of the method. Note that this way of deriving the MPM equations is not suitable to obtain the GIMP and CPDI formulations. That is why one must master the previous way which begins with the weak form.

The UL finite element mass matrix, internal force and external force vectors are given by Belytschko et al. (2000)

$$\begin{aligned} m_{IJ} &= \int_{\Omega} \rho \phi_I \phi_J \mathrm{d}\Omega \\ \mathbf{f}_I^{\text{int}} &= -\int_{\Omega} \nabla \phi_I \boldsymbol{\sigma} \mathrm{d}\Omega \\ \mathbf{f}_I^{\text{ext}} &= \int_{\Omega} \rho \phi_I \mathbf{b} \mathrm{d}\Omega \end{aligned} \tag{2.38}$$

where we have skipped the traction terms in the external force for simplicity.

Integrals in Eq. (2.38) are computed using *numerical integration* or *numerical quadrature* rules. The integrand is evaluated at a finite set of points called *integration points* or Gauss points and a weighted sum of these values is used to approximate the integral. For a function f, one writes

$$\int_{\Omega} f \mathrm{d}\Omega = \sum_g f(\mathbf{x}_g) w_g \tag{2.39}$$

in which w_g denotes the weight of integration point g and $\mathbf{x}_g$ is the position of this point. It can be shown that if the material points are taken as the integration points with the weights being its volumes then the FEM equations reduce to the MPM ones. For examples,

$$\mathbf{f}_I^{\text{int}} = -\int_{\Omega} \nabla \phi_I \boldsymbol{\sigma} \mathrm{d}\Omega = -\sum_p \nabla \phi_I(\mathbf{x}_p) \boldsymbol{\sigma}_p V_p \tag{2.40}$$

which is exactly Eq. (2.36).

In the MPM, as only the particle positions and volumes are updated, the deformed domain cannot be tiled without gaps as in the FEM, cf. Fig. 2.1. Thus, the integration measure is not exactly preserved in the MPM. In other words, the sum of the particle volumes is not exactly equal to the volume of the deformed domain. Therefore, the error due to numerical quadrature in the MPM (more precisely the standard MPM formulation) is significant. And this is just the first source of quadrature error in MPM.

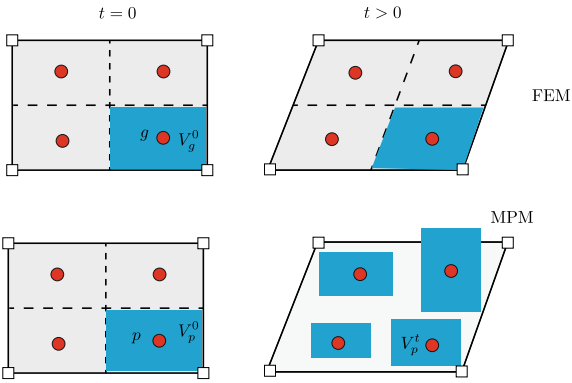


Fig. 2.1 Numerical quadrature in the FEM and MPM: quadrature measure error in MPM

Understanding this issue is crucial to reduce this error in developing methods. This is discussed in subsequent chapters of this book.

The second source of quadrature error emerges from the fact that particles are independently located with respect to the background grid. Therefore, the particle based quadrature does not respect the continuity of the grid functions or the support of these functions. This is similar to other Galerkin meshfree method, see e.g. Dolbow and Belytschko (1999). In the context of the MPM, Steffen et al. (2008a) presented a detailed analysis of this quadrature error.

Solutions to the quadrature error of the MPM include the utilization of smooth grid basis functions such as GIMP, B-spline, CPDI, or the adoption of finite element quadrature rules (commonly used in the geotechnical engineering community with unstructured background grids) and the FEMLIP of Moresi et al. (2003, 2007) where the quadrature weights are calculated on the fly such that affine functions can be exactly re-constructed. Among these techniques, CPDI is the most accurate in terms of quadrature.

2.5 Temporal Discretization and Resulting MPM Algorithms

Up to this point, we have obtained the semi-discrete, cf. Eq. (2.33). The full discrete equation is obtained by performing a time integration of this equation as described therein. We present mass lumping in Sect. 2.5.1 to avoid inverting the mass matrix, a time consuming step. In contrary to the FEM where the velocity field is stored at the nodes, the grid velocities are, in the MPM, nullified after every step when the grid is reset. Therefore, at the beginning of a time step t, one needs to project the particle velocities to the grid nodes to serve as the starting point for the time advancement. This crucial step is discussed in Sect. 2.5.2. Section 2.5.3 presents the first complete MPM algorithm named USL (update stress last) which is probably the most popular

MPM algorithm. A slight modification of the USL and dubbed MUSL (modified USL) is given in Sect. 2.5.4 to enhance the stability of MPM simulations.

2.5.1 Lumped Mass Matrix

It is obvious from Eq. (2.33) that, to obtain the acceleration one needs to solve a system of linear equations at every time step. The size of this system is $3n_n \times 3n_n$ in 3D where n_n—the number of nodes of the grid—can be a very large number. To avoid this, a lumped mass matrix is adopted. This lumped mass matrix is a diagonal matrix of which the diagonal terms are given by

$$m_I = \sum_{J=1}^{n_n} m_{IJ} = \sum_{J=1}^{n_n} \sum_{p=1}^{n_p} m_p \phi_I(\mathbf{x}_p)\phi_J(\mathbf{x}_p) = \sum_{p=1}^{n_p} m_p \phi_I(\mathbf{x}_p) \tag{2.41}$$

where Eq. (2.34) was used in the second equality and the partition of unity (PU) property of FE shape functions, $\sum_J \phi_J(\mathbf{x}) = 1, \ \forall \mathbf{x}$, was used in the third equality.

Remark 12 A lumped mass matrix is more than just a computational simplification; it also gives better results for impulsive loadings (Benson 1992). Note also that the use of a lumped mass matrix results in energy dissipation, see e.g. Zienkiewicz and Taylor (2006) (Sect. 16.2.4).

Remark 13 As the momentum equation is resolved at the grid nodes and not the material points, the conservation of mass must be satisfied at the nodes. By using Eq. (2.41) one can write

$$\sum_I m_I = \sum_I \left(\sum_{p=1}^{n_p} m_p \phi_I(\mathbf{x}_p) \right) = \sum_{p=1}^{n_p} m_p \left(\sum_I \phi_I(\mathbf{x}_p) \right) = \sum_p m_p \tag{2.42}$$

which proved that mass is conserved at the grid nodes as well.

With this lumped mass matrix, one gets the following system of ordinary differential equations (ODE) in time:

$$m_I \mathbf{a}_I(t) = \mathbf{f}_I(t) := \mathbf{f}_I^{\text{ext}}(t) + \mathbf{f}_I^{\text{int}}(t) \tag{2.43a}$$

$$\mathbf{a}_I = \frac{d\mathbf{v}_I(t)}{dt} \tag{2.43b}$$

for all the nodes I.

Let's denote by t_f the simulation time. Then, for time discretization, the temporal domain $0 \leq t \leq t_f$ is divided into n_T time steps with time increment $\Delta t = t_f / n_T$.

Therefore, Eq. (2.43) has to be satisfied for every time instances $t_k = k\Delta t$ with $k = 0, 1, \ldots, n_T$. The most straightforward method to advance the solution in time i.e., solving the semi-discrete equations, is an explicit formulation in which the solution is advanced in time from t (i.e., t_k) to $t + \Delta t$ (or t_{k+1}) without solving a system of linear algebra equations. The forward Euler method is such an scheme and will be discussed in Sect. 2.5.3. Note that explicit schemes demand the use of quite small time steps Δt for stability.

A word about notation is in order. We use the superscript t to denote quantities at time instant t which are known and superscript $t + \Delta t$ for unknown quantities at the next time instant.

2.5.2 Calculation of Nodal Velocities (Momenta)

At the beginning of every time step, the particle velocities need to be mapped to the nodes. This step is not present in the FEM and is necessary as the grid is reset at the end of a time step and the nodal velocities for that step are lost. More precisely, the nodal momenta are mapped from the particles using the shape functions (Burgess et al. 1992)

$$(m\mathbf{v})_I^t = \sum_p \phi_I(\mathbf{x}_p^t)(m\mathbf{v})_p^t \tag{2.44}$$

or the particle momenta are projected to the grid nodes.

In what follows, we prove that a weighted least square approximation is needed for the momenta projection (Eq. (2.44)) as there are more particles than nodes. For the sake of presentation, let us consider a one dimensional grid made of one cell with two nodes of which velocities are denoted by v_i. Moreover, within this cell, there are two particles at x_1 and x_2.

The idea is to minimize the following function

$$J = \frac{1}{2}\left[m_1(V_1 - (\phi_1(x_1)v_1 + \phi_2(x_1)v_2))^2 + m_2(V_2 - (\phi_1(x_2)v_1 + \phi_2(x_2)v_2))^2\right] \tag{2.45}$$

which is a weighted least squares where the weights are the particle mass. In the above equation, V_i are the particle velocities (not volumes) and m_i are the particle masses. Differentiating J with respect to v_1 and v_2 and making them zeros one obtains

$$\begin{aligned}
[m_1V_1\phi_{11} + m_2V_2\phi_{12}] &= (\phi_{11}v_1 + \phi_{21}v_2)m_1\phi_{11} + (\phi_{12}v_1 + \phi_{22}v_2)m_2\phi_{12} \\
[m_1V_1\phi_{21} + m_2V_2\phi_{22}] &= (\phi_{11}v_1 + \phi_{21}v_2)m_1\phi_{21} + (\phi_{12}v_1 + \phi_{22}m_2)m_2\phi_{22}
\end{aligned} \tag{2.46}$$

where use was made of the short notation $\phi_{ij} = \phi_i(x_j)$, the above equation can be rewritten as follows

$$\begin{bmatrix} \phi_{11}\phi_{11}m_1 + \phi_{12}\phi_{12}m_2 & \phi_{11}\phi_{21}m_1 + \phi_{22}\phi_{12}m_2 \\ \phi_{11}\phi_{21}m_1 + \phi_{12}\phi_{22}m_2 & \phi_{21}\phi_{21}m_1 + \phi_{22}\phi_{22}m_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} m_1 V_1 \phi_1(x_1) + m_2 V_2 \phi_1(x_2) \\ m_1 V_1 \phi_2(x_1) + m_2 V_2 \phi_2(x_2) \end{bmatrix} \tag{2.47}$$

which is a system of linear algebra equations to solve for v_1 and v_2. It can be observed that the coefficient matrix of this linear system is exactly the consistent mass matrix. To avoid the solution of such a system, a row sum lumping technique is used which results in the following

$$\begin{bmatrix} \phi_1(x_1)m_1 + \phi_1(x_2)m_2 & 0 \\ 0 & \phi_2(x_1)m_1 + \phi_2(x_2)m_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} m_1 V_1 \phi_1(x_1) + m_2 V_2 \phi_1(x_2) \\ m_1 V_1 \phi_2(x_1) + m_2 V_2 \phi_2(x_2) \end{bmatrix} \tag{2.48}$$

which can be generalized to obtain Eq. (2.44) which concludes the proof.

Remark 14 We are checking whether linear momentum is conserved with Eq. (2.44). By using Eq. (2.44) one can write

$$\sum_I (m\mathbf{v})_I = \sum_I \left(\sum_{p=1}^{n_p} (m\mathbf{v})_p \phi_I(\mathbf{x}_p) \right) = \sum_{p=1}^{n_p} (m\mathbf{v})_p \left(\sum_I \phi_I(\mathbf{x}_p) \right) = \sum_p (m\mathbf{v})_p \tag{2.49}$$

which proved that linear momentum is conserved at grid nodes as well (as long as ϕ_I makes a partition of unity).

Remark 15 Even though this projection of particle velocity has been used nearly in all MPM simulations, we will see later in Sect. 9.3 that it is not able to provide an exact projection of a linear velocity field for arbitrary particle positions.

2.5.3 *Standard Formulation (USL)*

Equation (2.43a) is used to obtain the nodal accelerations $\mathbf{a}_I^t = \mathbf{f}_I^t / m_I^t$, and then the nodal velocity field is updated using the explicit Euler forward method as follows

$$\mathbf{v}_I^{t+\Delta t} = \mathbf{v}_I^t + \Delta t \mathbf{a}_I^t \tag{2.50}$$

where $\mathbf{v}_I^t$ denotes the nodal velocity at time t, which is known; Δt is the time increment. Theoretically, the nodes are moved to the new positions given by

$$\mathbf{x}_I^{t+\Delta t} = \mathbf{x}_I^t + \Delta t \mathbf{v}_I^{t+\Delta t} \tag{2.51}$$

Note that this nodal position update is rarely realized as the grid would be reset in the beginning of the next time step anyway as done in most MPM implementations. However, it should be emphasized that grid resetting is not a requirement. Grid nodes can be updated using Eq. (2.51) until the grid is distorted or the grid can be replaced

with any other suitable grid. This is possible because all information has been stored at the particles.

Once the grid has been updated, the grid velocities are used to update the particle state including position, velocity, volume, deformation gradient, stresses etc. We discuss the update of the particle positions and velocities first as there are different options which might confuse new comers to the field. The options are PIC, FLIP and a blended PIC-FLIP.

In the PIC way, the particle velocity is obtained using the total grid velocity. On the other hand, according to the FLIP way, the particle velocity is obtained using the grid velocity increment. These are summarized in the following equations

$$\text{PIC:} \quad \mathbf{v}_p^{t+\Delta t} = \sum_I \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t} \tag{2.52}$$

$$\text{FLIP:} \quad \mathbf{v}_p^{t+\Delta t} = \mathbf{v}_p^t + \sum_I \phi_I(\mathbf{x}_p^t)\left[\mathbf{v}_I^{t+\Delta t} - \mathbf{v}_I^t\right] \tag{2.53}$$

As PIC replaces the particle velocity by the grid velocity, information is loss (there are many more particles than grid nodes), and thus PIC has numerical dissipation (or in other words, energy is not conserved for problems without dissipation). FLIP overcomes that by adding the grid velocity increment to the particle only (Brackbill and Ruppel 1986).

A combination of PIC and FLIP was first introduced by Zhu and Bridson (2005); Stomakhin et al. (2013) in the computer graphics community. It was used later in the engineering community, e.g. by Leroch et al. (2018). In this blended PIC-FLIP, a mix of PIC and FLIP is used for the particle velocity

$$\mathbf{v}_p^{t+\Delta t} = \alpha\left(\mathbf{v}_p^t + \sum_I \phi_I(\mathbf{x}_p^t)\left[\mathbf{v}_I^{t+\Delta t} - \mathbf{v}_I^t\right]\right) + (1-\alpha)\sum_I \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t} \tag{2.54}$$

$$\mathbf{x}_p^{t+\Delta t} = \mathbf{x}_p^t + \Delta t \sum_I \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t} \tag{2.55}$$

where $\alpha = 1$ corresponds to the FLIP and $\alpha = 0$ corresponds to the PIC. We refer to Fig. 2.2 for an illustration on the influence of α and Sect. 6.15.1 for more detail. Note that Leroch et al. (2018) updated the particle position as $\mathbf{x}_p^{t+\Delta t} = \mathbf{x}_p^t + \Delta t \mathbf{v}_p^{t+\Delta t}$ which we found to give similar results to the standard way.

Remark 16 The particle velocity update is actually computed as follows

$$\begin{aligned}\mathbf{v}_p^{t+\Delta t} &= \alpha\left(\mathbf{v}_p^t + \sum_I \phi_I(\mathbf{x}_p^t)\left[\mathbf{v}_I^{t+\Delta t} - \mathbf{v}_I^t\right]\right) + (1-\alpha)\sum_I \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t} \\ &= \alpha\mathbf{v}_p^t + \sum_I \phi_I(\mathbf{x}_p^t)\left[\mathbf{v}_I^{t+\Delta t} - \alpha\mathbf{v}_I^t\right]\end{aligned} \tag{2.56}$$

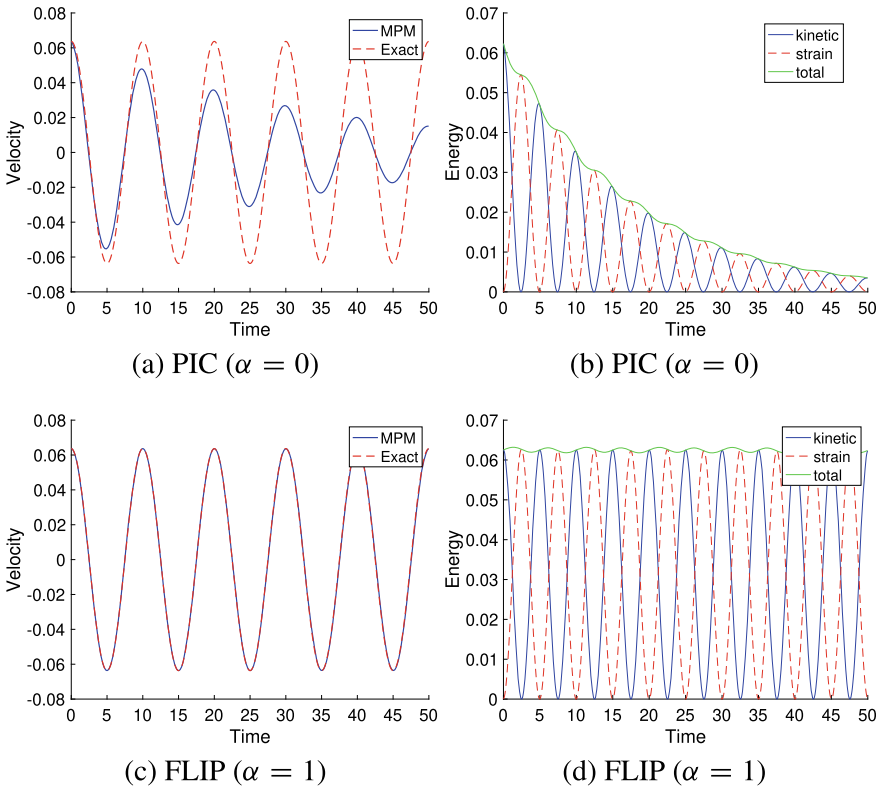


Fig. 2.2 Vibration of a bar: PIC versus FLIP

Finally, particle stresses are updated. This is known as the update stress last (USL) formulation in the MPM literature. There exists other formulations as discussed in Remark 21. Depending on the constitutive models being used, one might need to compute the gradient deformation $\mathbf{F}$, the velocity gradient $\mathbf{L}$ and the rate of deformation $\mathbf{D}$ etc. For example, one may need to compute the particle velocity gradients, defined in Eq. (2.11), then compute the gradient deformation using the relation $\dot{\mathbf{F}} = \mathbf{L}\mathbf{F}$, and finally using the continuity equation $\rho J = \rho_0$ to determine the updated volume. This is typically for hyperelastic solids. They are given by

$$\mathbf{L}_p^{t+\Delta t} \equiv \nabla \mathbf{v}_p^{t+\Delta t} = \sum_I \nabla \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t} \tag{2.57a}$$

$$\frac{\mathbf{F}^{t+\Delta t} - \mathbf{F}^t}{\Delta t} = \mathbf{L}^{t+\Delta t}\mathbf{F}^t, \quad \Rightarrow \mathbf{F}_p^{t+\Delta t} = (\mathbf{I} + \mathbf{L}_p^{t+\Delta t}\Delta t)\mathbf{F}_p^t \tag{2.57b}$$

$$V_p^{t+\Delta t} = J V_p^0, \quad J = \det \mathbf{F}_p^{t+\Delta t} \tag{2.57c}$$

$$\rho_p^{t+\Delta t} = \rho_0 / J \tag{2.57d}$$

where $\mathbf{L}_p$ is a 3×3 matrix of which components are $L_{ij} = v_{i,j}$ (in three dimensions). In the above equation, $\mathbf{I}$ is the identity matrix, and $\mathbf{F}_p$ is a 3×3 matrix with the initial matrix being $\mathbf{I}$ i.e., $\mathbf{F}_p^0 = \mathbf{I}$. We have added the equation to update the particle density, which is for example needed to calculate the equations of state. Note that there exists other ways to compute the deformation gradient with higher accuracy, albeit with more complexities.

Remark 17 The velocity gradient $\mathbf{L}$ is actually computed as

$$\mathbf{L} = \begin{bmatrix} \sum_I \phi_{I,x} v_{xI} & \sum_I \phi_{I,y} v_{xI} & \sum_I \phi_{I,z} v_{xI} \\ \sum_I \phi_{I,x} v_{yI} & \sum_I \phi_{I,y} v_{yI} & \sum_I \phi_{I,z} v_{yI} \\ \sum_I \phi_{I,x} v_{zI} & \sum_I \phi_{I,y} v_{zI} & \sum_I \phi_{I,z} v_{zI} \end{bmatrix} = \sum_I \begin{bmatrix} v_{xI} \\ v_{yI} \\ v_{zI} \end{bmatrix} \begin{bmatrix} \phi_{I,x} & \phi_{I,y} & \phi_{I,z} \end{bmatrix} \tag{2.58}$$

for 3D problems. Simplifications for 1D and plane strain/stress 2D problems are straightforward. For axi-symmetric problems, see Sect. 2.7.

For a hypoelastic constitutive model, one needs to compute the strain increment $\Delta \mathbf{e}_p = \left(\text{sym}\mathbf{L}_p^{t+\Delta t}\right) \Delta t$ and using it to compute the stress increment $\Delta \boldsymbol{\sigma}_p$. The updated particle stresses are given by

$$\boldsymbol{\sigma}_p^{t+\Delta t} = \boldsymbol{\sigma}_p^t + \Delta \boldsymbol{\sigma}_p \tag{2.59}$$

For complex constitutive models, it might be required to calculate other quantities to update the particle stresses. We refer to Chap. 4 for some common material models for elastic, hyperelastic and elasto-plastic solids and Sect. 10.1 for fluids and gaseous. As the material points are Lagrangian, existing stress update algorithms developed mainly by the FEM community can be readily reused in the MPM. We refer to the textbooks of Simo and Hughes (1998); de Souza Neto et al. (2011) for detail. And this brings us to the first complete explicit ULMPM algorithm given in Algorithm 1. As can be seen, it is a very simple algorithm, which can be coded straightforwardly. And yet, it has been used to solve many challenging solid mechanics problems. Note that m_p lacks a time label because it is never changed to ensure mass conservation.

Algorithm 1 Solution procedure of explicit MPM (USL, cut-off).

Initialization
  Set up the Cartesian grid, set time $t = 0$
  Set up particle data: $\mathbf{x}_p^0, \mathbf{v}_p^0, \boldsymbol{\sigma}_p^0, \mathbf{F}_p^0, V_p^0, m_p, \rho_p^0$
end
while $t < t_f$ **do**
  Reset grid quantities: $m_I^t = 0$, $(m\mathbf{v})_I^t = \mathbf{0}$, $\mathbf{f}_I^{\text{ext},t} = \mathbf{0}$, $\mathbf{f}_I^{\text{int},t} = \mathbf{0}$
  Mapping from particles to nodes (P2G)
    Compute nodal mass $m_I^t = \sum_p \phi_I(\mathbf{x}_p^t) m_p$
    Compute nodal momentum $(m\mathbf{v})_I^t = \sum_p \phi_I(\mathbf{x}_p^t)(m\mathbf{v})_p^t$
    Compute external force $\mathbf{f}_I^{\text{ext},t} = \sum_p \phi_I(\mathbf{x}_p) m_p \mathbf{b}(\mathbf{x}_p)$
    Compute internal force $\mathbf{f}_I^{\text{int},t} = -\sum_p V_p^t \boldsymbol{\sigma}_p^t \nabla\phi_I(\mathbf{x}_p^t)$
    Compute nodal force $\mathbf{f}_I^t = \mathbf{f}_I^{\text{ext},t} + \mathbf{f}_I^{\text{int},t}$
  end
  Update the momenta $(m\mathbf{v})_I^{t+\Delta t} = (m\mathbf{v})_I^t + \mathbf{f}_I^t \Delta t$
  Fix Dirichlet nodes I e.g. $(m\mathbf{v})_I^t = \mathbf{0}$ and $(m\mathbf{v})_I^{t+\Delta t} = \mathbf{0}$
  Update particles (G2P)
    Get nodal velocities $\mathbf{v}_I^t = (m\mathbf{v})_I^t / m_I^t$ and $\mathbf{v}_I^{t+\Delta t} = (m\mathbf{v})_I^{t+\Delta t} / m_I^t$
    Update particle velocities $\mathbf{v}_p^{t+\Delta t} = \alpha\left(\mathbf{v}_p^t + \sum_I \phi_I(\mathbf{x}_p^t)\left[\mathbf{v}_I^{t+\Delta t} - \mathbf{v}_I^t\right]\right) + (1 - \alpha)\sum_I \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t}$
    Update particle positions $\mathbf{x}_p^{t+\Delta t} = \mathbf{x}_p^t + \Delta t \sum_I \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t}$
    Compute velocity gradient $\mathbf{L}_p^{t+\Delta t} = \sum_I \nabla\phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t}$
    Updated gradient deformation tensor $\mathbf{F}_p^{t+\Delta t} = (\mathbf{I} + \mathbf{L}_p^{t+\Delta t}\Delta t)\mathbf{F}_p^t$
    Update volume $V_p^{t+\Delta t} = \det \mathbf{F}_p^{t+\Delta t} V_p^0$
    Compute the rate of deformation matrix $\mathbf{D}_p^{t+\Delta t} = 0.5(\mathbf{L}_p^{t+\Delta t} + (\mathbf{L}_p^{t+\Delta t})^{\mathrm{T}})$
    Compute the strain increment $\Delta\boldsymbol{\epsilon}_p = \Delta t \mathbf{D}_p^{t+\Delta t}$
    Update stresses: $\boldsymbol{\sigma}_p^{t+\Delta t} = \boldsymbol{\sigma}_p^t + \Delta\boldsymbol{\sigma}_p(\Delta\boldsymbol{\epsilon}_p)$, or $\boldsymbol{\sigma}_p^{t+\Delta t} = \boldsymbol{\sigma}_p^{t+\Delta t}(\mathbf{F}_p^{t+\Delta t})$
  end
  Advance time $t = t + \Delta t$
  Error calculation: if needed (e.g. for convergence tests)
end while

Remark 18 There are some remarks on Algorithm 1. First, we have omitted the contribution to the external force due to non-zero traction. This is because it is more difficult to deal with than with the FEM as discussed in Sect. 5.2.3. Second, we have assumed that a constant time step $\Delta t = \text{const}$ was used. In practice, varying time steps are often adopted, see Sect. 2.8 for details. Finally, we presented the so-called momentum formulation. Note that this momentum formulation is very common but it does not improve the stability of the MPM.

Small mass issue. There is a numerical issue in this formulation: the division operator in $\mathbf{a}_I^t = \mathbf{f}_I^t / m_I^t$ would yield infinite acceleration if the mass m_I^t is small. In turn, the velocity gradient in Eq. (2.57a) would be infinite as well and this would spoil the particle stresses (Sulsky et al. 1995b). This happens when a particle is very close to a node which has only one particle within its support, cf. Fig. 2.3. This usually happens with nodes located close to the material interface. For multiple bodies simulations,

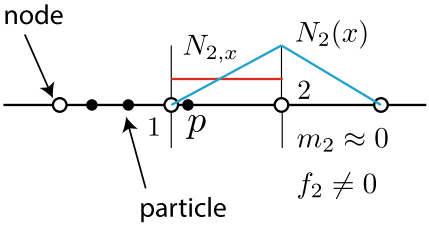


Fig. 2.3 Troubled nodes with nearly zero mass resulting in infinite acceleration (node 2)

those interface nodes are often in contact and therefore contact algorithms have to carefully address the small nodal mass issue.

To identify the trouble more clearly, we turn to the example given in Fig. 2.3. We assume that there is only one element and one single particle, with mass denoted by m_p, located very close to node 1. Furthermore, we assume the old nodal velocities are zero i.e., $v_1^t = v_2^t = 0$.

The nodal masses are given by

$$\begin{aligned} m_1 &= m_p N_1 \\ m_2 &= m_p N_2 \quad \text{(small value)} \end{aligned} \tag{2.60}$$

And the nodal accelerations are thus

$$\begin{aligned} a_1 &= f_1/m_1 \\ a_2 &= f_2/m_2 \quad \text{(very large value)} \end{aligned} \tag{2.61}$$

where f_1 and f_2 are the nodal forces. The updated nodal velocities are given by using Eq. (2.50)

$$\begin{aligned} v_1^{t+\Delta t} &= \Delta t f_1/m_1 \\ v_2^{t+\Delta t} &= \Delta t f_2/m_2 \quad \text{(very large value)} \end{aligned} \tag{2.62}$$

The updated particle position and velocity are given by according to Eqs. (2.54) and (2.55) with $\alpha = 1$

$$\begin{aligned} v_p^{t+\Delta t} &= v_p^t + \Delta t(N_1 f_1/m_1 + N_2 f_2/m_2) = v_p^t + \Delta t(f_1/m_p + f_2/m_p) \\ x_p^{t+\Delta t} &= x_p^t + \Delta t(N_1 \Delta t f_1/m_1 + N_2 \Delta t f_2/m_2) = x_p^t + \Delta t^2(f_1/m_p + f_2/m_p) \end{aligned} \tag{2.63}$$

where Eq. (2.60) was used for the nodal masses m_1 and m_2. Note that the updated particle position and velocity are normal as they are smoothed out by the shape functions.

Next, one computes the velocity gradient needed for stress updating

$$L_p^{t+\Delta t} = -\frac{1}{h} v_1^{t+\Delta t} + \frac{1}{h} v_2^{t+\Delta t} \quad \text{(very large value)} \tag{2.64}$$

In conclusion, the problem lies in the use of $v_2^{t+\Delta t}$ in computing the velocity gradient but not in updating the particle velocity and position. A technique to solve this issue, proposed by Sulsky et al. (1995b), will be presented in Sect. 2.5.4.

Cutoff technique. In this technique, a small positive threshold is introduced to cure the small mass issue. Accordingly, the nodal velocities are computed as

$$\mathbf{v}_I^{t+\Delta t} = \begin{cases} \dfrac{(m\mathbf{v})_I^{t+\Delta t}}{m_I^t} & \text{if} m_I^t > tol \\ 0 & \text{otherwise} \end{cases} \tag{2.65}$$

This algorithm requires an extra parameter (a cutoff value). Yet how to chose it is not clear. Even if a good cutoff value can be chosen, it produces an undesirable constraint which should not be in the system. More advanced techniques are presented in what follows.

Remark 19 Wallstedt and Guilkey (2008) have studied a number of families of time integration schemes for use with GIMP including Runge Kutta, Runge-Kutta-Nystrom, Adams-Bashforth-Moulton (ABM), and Predictor-Corrector Newmark methods. They reported that few of these methods have been able to achieve their formal orders of accuracy. Not only is the MPM used for highly discontinuous and nonlinear problems but the spatial error of the method tend to overwhelm any improvement that a temporally high order method might offer. They also showed that the central difference scheme, commonly used in nonlinear finite element codes (Belytschko et al. 2000), is exactly the same as USL but for one crucial difference: initialization of particle velocity to a negative half time step.

Remark 20 As can be seen from Algorithm 1, the MPM algorithm is very similar to the ULFEM, cf. Algorithm 24 in Appendix D. There are just a few differences. First, the nodal mass and velocity have to be re-calculated at the beginning of every time step. This is natural as the grid does not store permanent information. Second, one needs to update the particle's position and velocity. In the ULFEM, the integration points' position are fixed and one does not need to calculate their velocity. Based on this observation, per time step, the MPM is about 2–3 times slower than the ULFEM. Yet, for very large deformation problems, Ma et al. (2009a) showed that their in-house MPM code is much faster than the commercial LS-DYNA FEM.

2.5.4 Modified Update Stress Last (MUSL)

In the MUSL of Sulsky et al. (1995b), the updated particle velocities are mapped back to the nodes to get the nodal velocities using

$$(m\mathbf{v})_I^{t+\Delta t} = \sum_p \phi_I(\mathbf{x}_p)(m\mathbf{v})_p^{t+\Delta t} \tag{2.66}$$

and thus

$$\mathbf{v}_I^{t+\Delta t} = \frac{(m\mathbf{v})_I^{t+\Delta t}}{m_I^t} = \frac{\sum_p \phi_I(\mathbf{x}_p)(m\mathbf{v})_p^{t+\Delta t}}{\sum_p \phi_I(\mathbf{x}_p) m_p} = \frac{\sum_p \phi_I(\mathbf{x}_p)(m\mathbf{v})_p^{t+\Delta t}}{m_I^t} \tag{2.67}$$

In the second equality, the appearance of the shape functions in both numerator and denominator cancels out its role and the numerical problem regarding very large velocity gradient is thus cured. Applying this to the example given in Fig. 2.3, the updated velocity at the troubled node is now given by

$$v_2^{t+\Delta t} = (1/m_2)\left[\Delta t\left(\frac{f_1}{m_p} + \frac{f_2}{m_p}\right) m_p\right] N_2 = \Delta t\left(\frac{f_1}{m_p} + \frac{f_2}{m_p}\right) \tag{2.68}$$

where the first of Eq. (2.63) was used with a simplification that $v_p^t = 0$. Apparently $v_2^{t+\Delta t}$ is not infinite and can be safely used for computing the velocity gradient. The resulting algorithm, dubbed MUSL, is given in Algorithm 2. Other name for this algorithm is the double mapping USL as the particle momenta are extrapolated to the nodes twice—at the beginning of the step and after updating the nodal momenta. We use a tilde $\tilde{\square}$ to denote the temporary grid velocities (Line 14). The differences of MUSL compared with USL are in Lines 16–21.

Remark 21 It was Bardenhagen (2002) who introduced the term Update Stress Last (USL) and presented an Update Stress First (USF) algorithm where stresses are updated in the P2G step. He found that while USL is dissipative (i.e., suffers from numerical dissipation), USF conserves energies well. Nairn (2003) analyzed the USF and MUSL for a 2D elastic vibration problem. He found that USL is very unstable, that the MUSL approach slowly dissipated energy while the USF approach slowly increased energy. Wallstedt and Guilkey (2008) used the method of manufactured solutions to test temporal and spatial convergence of GIMP with USF and USL. Their results show that USL is superior in terms of stability and convergence.

Remark 22 We have so far presented just explicit dynamics MPM. For implicit dynamics and quasi-static MPM formulations, please refer to the discussion in Sect. 1.5.4. It is quite straightforward to develop these algorithms as the MPM is very similar to the updated Lagrangian FEM. That is expressions for the geometric and material tangent matrices developed for ULFEM can be readily reused, see Belytschko et al. (2000). A clear presentation of an implicit dynamics MPM is given in Iaconeta et al. (2017).

Remark 23 An explicit MPM code can also be used for quasi-static problems . Even though using an explicit code for simple static simulations is not efficient (due to many time steps for a long simulation period), an explicit code is the only option for challenging static simulations where implicit solvers would crash (e.g. the solver does not converge). Global and local damping can be added to mitigate the effects of stress waves for static simulations (Al-Kafaji 2013).

Algorithm 2 Solution procedure of explicit MPM (MUSL).

Initialization
  Set up the Cartesian grid, set time $t = 0$
  Set up particle data: $\mathbf{x}_p^0$, $\mathbf{v}_p^0$, $\boldsymbol{\sigma}_p^0$, $\mathbf{F}_p^0$, V_p^0, m_p, ρ_p^0
end
while $t < t_f$ **do**
  Reset grid quantities: $m_I^t = 0$, $(m\mathbf{v})_I^t = \mathbf{0}$, $\mathbf{f}_I^{\text{ext},t} = \mathbf{0}$, $\mathbf{f}_I^{\text{int},t} = \mathbf{0}$
  Mapping from particles to nodes (P2G)
    Compute nodal mass $m_I^t = \sum_p \phi_I(\mathbf{x}_p^t) m_p$
    Compute nodal momentum $(m\mathbf{v})_I^t = \sum_p \phi_I(\mathbf{x}_p^t)(m\mathbf{v})_p^t$
    Compute external force $\mathbf{f}_I^{\text{ext},t} = \sum_p \phi_I(\mathbf{x}_p) m_p \mathbf{b}(\mathbf{x}_p)$
    Compute internal force $\mathbf{f}_I^{\text{int},t} = -\sum_p V_p^t \boldsymbol{\sigma}_p^t \nabla\phi_I(\mathbf{x}_p^t)$
    Compute nodal force $\mathbf{f}_I^t = \mathbf{f}_I^{\text{ext},t} + \mathbf{f}_I^{\text{int},t}$
  end
  Update the momenta $(m\tilde{\mathbf{v}})_I^{t+\Delta t} = (m\mathbf{v})_I^t + \mathbf{f}_I^t \Delta t$
  Fix Dirichlet nodes I e.g. $(m\mathbf{v})_I^t = \mathbf{0}$ and $(m\tilde{\mathbf{v}})_I^{t+\Delta t} = \mathbf{0}$
  Update particle velocities and grid velocities (double mapping)
    Get nodal velocities $\tilde{\mathbf{v}}_I^{t+\Delta t} = (m\tilde{\mathbf{v}})_I^{t+\Delta t} / m_I^t$
    Update particle positions $\mathbf{x}_p^{t+\Delta t} = \mathbf{x}_p^t + \Delta t \sum_I \phi_I(\mathbf{x}_p^t)\tilde{\mathbf{v}}_I^{t+\Delta t}$
    Update particle velocities $\mathbf{v}_p^{t+\Delta t} = \alpha\left(\mathbf{v}_p^t + \sum_I \phi_I(\mathbf{x}_p^t)\left[\tilde{\mathbf{v}}_I^{t+\Delta t} - \mathbf{v}_I^t\right]\right) + (1-\alpha)\sum_I \phi_I(\mathbf{x}_p^t)\tilde{\mathbf{v}}_I^{t+\Delta t}$
    Update grid velocities $(m\mathbf{v}_I)^{t+\Delta t} = \sum_p \phi_I(\mathbf{x}_p^t)(m\mathbf{v})_p^{t+\Delta t}$
    Fix Dirichlet nodes $(m\mathbf{v})_I^{t+\Delta t} = \mathbf{0}$
  end
  Update particles (G2P)
    Get nodal velocities $\mathbf{v}_I^{t+\Delta t} = (m\mathbf{v})_I^{t+\Delta t} / m_I^t$
    Compute velocity gradient $\mathbf{L}_p^{t+\Delta t} = \sum_I \nabla\phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t}$
    Updated gradient deformation tensor $\mathbf{F}_p^{t+\Delta t} = (\mathbf{I} + \mathbf{L}_p^{t+\Delta t}\Delta t)\mathbf{F}_p^t$
    Update volume $V_p^{t+\Delta t} = \det\mathbf{F}_p^{t+\Delta t} V_p^0$
    Update stresses $\boldsymbol{\sigma}_p^{t+\Delta t} = \boldsymbol{\sigma}_p^t + \Delta\boldsymbol{\sigma}_p$
  end
  Advance time $t = t + \Delta t$
  Error calculation: if needed (e.g. for convergence tests)
end while

2.5.5 *Update Stress First (USF)*

USF is the last method introduced (Bardenhagen 2002). In USF, stresses are updated at the beginning of the time step, not at the end (see Algorithm 3). According to Nairn (2003), USF is another way to mitigate the small mass problem discussed in Sect. 2.5.4.

Bardenhagen (2002) found that USF conserves energy better than USL. To know more about this problem of energy conservation is the MPM, please refer to Sect. 9.1.3, where it is covered in detail.

Algorithm 3 Solution procedure of explicit MPM (USF).

Initialization
  Set up the Cartesian grid, set time $t = 0$
  Set up particle data: $\mathbf{x}_p^0$, $\mathbf{v}_p^0$, $\boldsymbol{\sigma}_p^0$, $\mathbf{F}_p^0$, V_p^0, m_p, ρ_p^0
end
while $t < t_f$ **do**
  Reset grid quantities: $m_I^t = 0$, $(m\mathbf{v})_I^t = \mathbf{0}$, $\mathbf{f}_I^{\text{ext},t} = \mathbf{0}$, $\mathbf{f}_I^{\text{int},t} = \mathbf{0}$
  Mapping from particles to nodes (P2G)
    Compute nodal mass $m_I^t = \sum_p \phi_I(\mathbf{x}_p^t) m_p$
    Compute nodal momentum $(m\mathbf{v})_I^t = \sum_p \phi_I(\mathbf{x}_p^t)(m\mathbf{v})_p^t$
    Compute velocity gradient $\mathbf{L}_p^t = \sum_I \nabla\phi_I(\mathbf{x}_p^t)\mathbf{v}_I^t$
    Updated gradient deformation tensor $\mathbf{F}_p^t = (\mathbf{I} + \mathbf{L}_p^t \Delta t)\mathbf{F}_p^{t-\Delta t}$
    Update volume $V_p^t = \det \mathbf{F}_p^t V_p^0$
    Compute the rate of deformation matrix $\mathbf{D}_p^t = 0.5(\mathbf{L}_p^t + (\mathbf{L}_p^t)^{\mathrm{T}})$
    Compute the strain increment $\Delta\boldsymbol{\epsilon}_p = \Delta t \mathbf{D}_p^t$
    Update stresses: $\boldsymbol{\sigma}_p^t = \boldsymbol{\sigma}_p^t + \Delta\boldsymbol{\sigma}_p(\Delta\boldsymbol{\epsilon}_p)$, or $\boldsymbol{\sigma}_p^t = \boldsymbol{\sigma}_p^t(\mathbf{F}_p^t)$
    Compute external force $\mathbf{f}_I^{\text{ext},t} = \sum_p \phi_I(\mathbf{x}_p) m_p \mathbf{b}(\mathbf{x}_p)$
    Compute internal force $\mathbf{f}_I^{\text{int},t} = -\sum_p V_p^t \boldsymbol{\sigma}_p^t \nabla\phi_I(\mathbf{x}_p^t)$
    Compute nodal force $\mathbf{f}_I^t = \mathbf{f}_I^{\text{ext},t} + \mathbf{f}_I^{\text{int},t}$
  end
  Update the momenta $(m\mathbf{v})_I^{t+\Delta t} = (m\mathbf{v})_I^t + \mathbf{f}_I^t \Delta t$
  Fix Dirichlet nodes I e.g. $(m\mathbf{v})_I^t = \mathbf{0}$ and $(m\mathbf{v})_I^{t+\Delta t} = \mathbf{0}$
  Update particles (G2P)
    Get nodal velocities $\mathbf{v}_I^t = (m\mathbf{v})_I^t / m_I^t$ and $\mathbf{v}_I^{t+\Delta t} = (m\mathbf{v})_I^{t+\Delta t} / m_I^t$
    Update particle velocities $\mathbf{v}_p^{t+\Delta t} = \alpha\left(\mathbf{v}_p^t + \sum_I \phi_I(\mathbf{x}_p^t)\left[\mathbf{v}_I^{t+\Delta t} - \mathbf{v}_I^t\right]\right) + (1 - \alpha)\sum_I \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t}$
    Update particle positions $\mathbf{x}_p^{t+\Delta t} = \mathbf{x}_p^t + \Delta t \sum_I \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t}$
  end
  Advance time $t = t + \Delta t$
  Error calculation: if needed (e.g. for convergence tests)
end while

2.6 Total Lagrangian MPM (TLMPM)

2.6.1 Motivation: Numerical Fracture

Numerical fracture occurs when the particles move so far out of the cell where they originally locate that a gap of one cell or more is created between them. To demonstrate this issue, we simulated the deformation of a hyperelastic square of unit size subjected to a downwards body force using the ULMPM with linear functions.

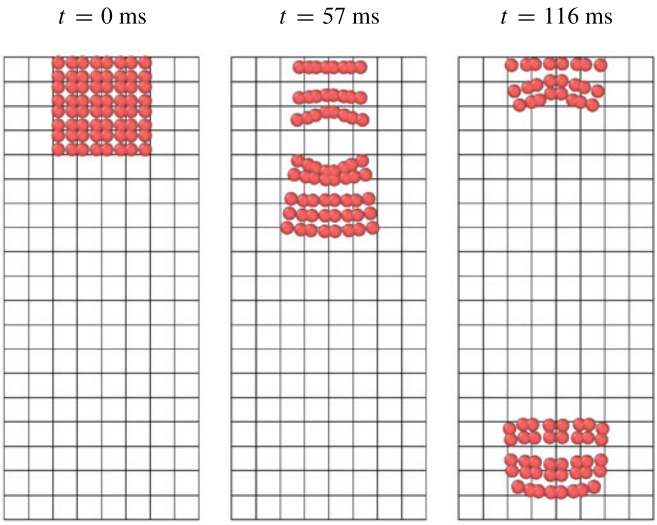
We can see from Fig. 2.4a that the solid continuously deforms and that after some time, it splits into two parts that are not connected anymore as shown by the fact that the displacement of the bottom right corner is parabolic at $t > 80$ ms (Fig. 2.4c). Such un-physical fracture depends only on the background grid cell-size and the weighting functions and is therefore not related to any physics. This unphysical numerical fracture would ultimately limit the accuracy of the MPM to model fracture of materials under large deformation. To mitigate this issue one can use either one of these options or all of them: (i) more particles per cell, (ii) smoother basis functions such as B-splines and (iii) particle splitting.

On the other hand, the TLMPM, to be presented in this section, is free of the numerical fracture trouble; the same simulation of the vertical bar using the TLMPM (Fig. 2.4b) shows the expected behaviour: the solid stretches and returns to its original position–no fracture occurs. Thus, the TLMPM is effective at removing "numerical" fracture without having to use techniques that require tracking of the particle domains (such as CPDI which ironically suffers from mesh-distortion).

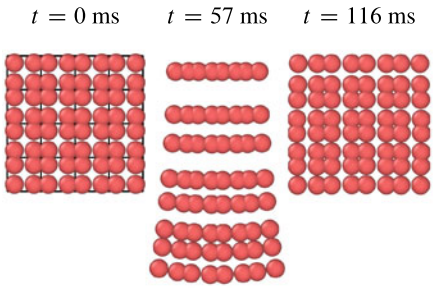
2.6.2 Derivation of TLMPM

A total Lagrangian MPM (TLMPM) was presented in de Vaucorbeil et al. (2020) where the stress and strain are Lagrangian, i.e., they are defined with respect to the reference configuration (for example, the 1st PK stress is employed), the spatial derivatives are computed with respect to the material coordinates. The corresponding weak form therefore involves integrals over the reference configuration. The result is a very efficient and easy to implement MPM that does not suffer cell-crossing error and numerical fracture. Furthermore, the TLMPM has a better quadrature approximation since the particles are always located at the optimal quadrature points. For all that, the inherent no-slip no-penetration contact capability in the ULMPM ceases to exist for the TLMPM.

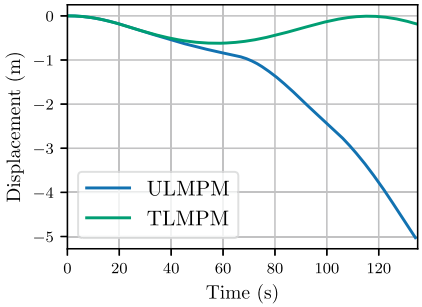
The algorithm is nearly identical to the standard MPM, see Algorithm 4. The differences lie in (1) the 1st PK stress is employed in the internal force, (2) the spatial derivatives are computed with respect to $\mathbf{X}_p$ (rather than $\mathbf{x}_p^t$), and (3) the deformation gradient and the velocity gradient are calculated differently. Furthermore, the nodal mass, the weighting functions, and gradients are computed once. And for some consitutive models adopting the Cauchy stress, one might need to convert it to the 1st PK stress.



(a) ULMPM



(b) TLMPM



(c) Vertical displacement

Fig. 2.4 Vertical bar simulations with coarse background grid illustrating **a** the problem of numerical fracture crippling the ULMPM and **b** how this is mitigated by the use of TLMPM. Note that in both simulations, the solid domain is identical but the magnification is different here. Also note that in the TLMPM simulations, the background grid is only present at the beginning of the simulation as it is only defined in the reference configuration, i.e., $t = 0$. In **c** is plotted the displacement of the particle at the bottom right corner. The solid's upper face is fixed and its Young's modulus, Poisson's ratio and density are respectively 200 MPa, 0.3 and 1050 kg/m^3. The body force magnitude is -1050 m/s^2 (de Vaucorbeil and Nguyen 2021b)

The deformation gradient can be computed as follows

$$\dot{\mathbf{F}}_p = \frac{\mathbf{F}_p^{t+\Delta t} - \mathbf{F}^t}{\Delta t} \Rightarrow \mathbf{F}_p^{t+\Delta t} = \mathbf{F}_p^t + \Delta t \dot{\mathbf{F}}_p, \quad \dot{\mathbf{F}}_p = \sum_I \nabla_0 \phi_I(\mathbf{X}_p) \mathbf{v}_I^{t+\Delta t} \tag{2.69}$$

or alternatively, $\mathbf{F}$ can be computed using the relation $\mathbf{F} = \mathbf{I} + \frac{\partial \mathbf{u}}{\partial \mathbf{X}}$:

$$\mathbf{F}_p^{t+\Delta t} = \mathbf{I} + \sum_I \nabla_0 \phi_I(\mathbf{X}_p)(\mathbf{x}_I^{t+\Delta t} - \mathbf{X}_I) \tag{2.70}$$

Our experiences show that the two ways yield identical results.

The velocity gradient $\mathbf{L}$ is then computed as

$$\mathbf{L} := \frac{\partial \mathbf{v}}{\partial \mathbf{x}} = \frac{\partial \mathbf{v}}{\partial \mathbf{X}} \frac{\partial \mathbf{X}}{\partial \mathbf{x}} = \dot{\mathbf{F}} \mathbf{F}^{-1} \tag{2.71}$$

From which one can compute the strain rate $\mathbf{D}$.

Remark 24 The rate of the deformation gradient $\dot{\mathbf{F}}$ is actually computed as

$$\dot{\mathbf{F}} = \begin{bmatrix} \sum_I \phi_{I,X} v_{xI} & \sum_I \phi_{I,Y} v_{xI} & \sum_I \phi_{I,Z} v_{xI} \\ \sum_I \phi_{I,X} v_{yI} & \sum_I \phi_{I,Y} v_{yI} & \sum_I \phi_{I,Z} v_{yI} \\ \sum_I \phi_{I,X} v_{zI} & \sum_I \phi_{I,Y} v_{xI} & \sum_I \phi_{I,Z} v_{zI} \end{bmatrix} = \sum_I \begin{bmatrix} v_{xI} \\ v_{yI} \\ v_{zI} \end{bmatrix} \begin{bmatrix} \phi_{I,X} & \phi_{I,Y} & \phi_{I,Z} \end{bmatrix} \tag{2.72}$$

for 3D problems.

Remark 25 Similar to the ULMPM, the TLMPM can be derived following two ways. In the first way, one can start from the strong form of the governing equations in the TL form and develop the corresponding weak form. In the second way, one can directly use the TLFEM semi-discrete equations and use the particles as the integration points. Since we have done these steps for the ULMPM, we do not repeat them for the TLMPM.

Remark 26 Even though the TLMPM is very similar to the TLFEM, there are subtle differences. First, the TLMPM does not need a mesh conforming to the solid under consideration. Second, modeling contact can be done in the spirit of particle methods. Third, the TLMPM provides an ideal test bed for developing high order MPM algorithms as it eliminates many issues of the ULMPM.

Algorithm 4 Solution procedure of explicit TLMPM (MUSL).

Initialization
  Set up particle data: $\mathbf{X}_p, \mathbf{v}_p^0, \boldsymbol{\sigma}_p^0, \mathbf{F}_p^0, V_p^0, m_p, \rho_p^0$
  Compute nodal mass $m_I = \sum_p \phi_I(\mathbf{X}_p) m_p$
  Compute and store weighting and gradient $\phi_I(\mathbf{X}_p)$ and $\nabla_0 \phi_I(\mathbf{X}_p)$
end
while $t < t_f$ **do**
  Reset grid quantities: $(m\mathbf{v})_I^t = \mathbf{0}, \mathbf{f}_I^{\text{ext},t} = \mathbf{0}, \mathbf{f}_I^{\text{int},t} = \mathbf{0}$
  Mapping from particles to nodes (P2G)
    Compute nodal momentum $(m\mathbf{v})_I^t = \sum_p \phi_I(\mathbf{X}_p)(m\mathbf{v})_p^t$
    Compute external force $\mathbf{f}_I^{\text{ext},t}$
    Compute internal force $\mathbf{f}_I^{\text{int},t} = -\sum_{p=1}^{n_p} V_p^0 \boldsymbol{P}_p^t \nabla_0 \phi_I(\mathbf{X}_p)$
    Compute nodal force $\mathbf{f}_I^t = \mathbf{f}_I^{\text{ext},t} + \mathbf{f}_I^{\text{int},t}$
  end
  Update the momenta $(m\tilde{\mathbf{v}})_I^{t+\Delta t} = (m\mathbf{v})_I^t + \mathbf{f}_I^t \Delta t$
  Fix Dirichlet nodes I e.g. $(m\tilde{\mathbf{v}})_I^{t+\Delta t} = \mathbf{0}$ and $(m\mathbf{v})_I^t = \mathbf{0}$
  Update particle velocities and grid velocities (double mapping)
    Get nodal velocities $\tilde{\mathbf{v}}_I^{t+\Delta t} = (m\tilde{\mathbf{v}})_I^{t+\Delta t}/m_I^t$
    Update particle velocities $\mathbf{v}_p^{t+\Delta t} = \alpha\left(\mathbf{v}_p^t + \sum_I \phi_I(\mathbf{X}_p)\left[\tilde{\mathbf{v}}_I^{t+\Delta t} - \mathbf{v}_I^t\right]\right) + (1 - \alpha)\sum_I \phi_I(\mathbf{X}_p)\tilde{\mathbf{v}}_I^{t+\Delta t}$
    Update grid velocities $(m\mathbf{v}_I)^{t+\Delta t} = \sum_p \phi_I(\mathbf{X}_p)(m\mathbf{v})_p^{t+\Delta t}$
    Fix Dirichlet nodes $(m\mathbf{v})_I^{t+\Delta t} = \mathbf{0}$
  end
  Update particle (G2P)
    Compute $\dot{\mathbf{F}}_p^{t+\Delta t} = \sum_I \nabla_0 \phi_I(\mathbf{X}_p)\mathbf{v}_I^{t+\Delta t}$
    Updated gradient deformation tensor $\mathbf{F}_p^{t+\Delta t} = \mathbf{F}_p^t + \Delta t \dot{\mathbf{F}}_p^{t+\Delta t}$
    Velocity gradient $\mathbf{L}_p^{t+\Delta t} = \dot{\mathbf{F}}_p^{t+\Delta t}(\mathbf{F}_p^{t+\Delta t})^{-1}$
    Update stresses $\boldsymbol{\sigma}_p^{t+\Delta t} = \boldsymbol{\sigma}_p^t + \Delta \boldsymbol{\sigma}_p$
    Covert stresses to 1st PK stresses $\mathbf{P}_p^{t+\Delta t} = g(\boldsymbol{\sigma}_p^{t+\Delta t})$ using Table 2.1
    Update particle positions (for visualization) $\mathbf{x}_p^{t+\Delta t} = \mathbf{x}_p^t + \Delta t \sum_I \phi_I(\mathbf{X}_p)\mathbf{v}_I^{t+\Delta t}$
  end
end while

The formulation presented so far can be used for either 1D, plane stress/strain 2D or 3D problems. For axi-symmetric problems, slight modifications are needed (see next section).

2.7 Axi-Symmetric MPM

Structures of revolution (SOR) subject to loading which is symmetric about the axis of revolution can be effectively modeled using the so-called two dimensional axi-symmetric formulations. As can be seen from Fig. 2.5, a SOR is obtained by revolving a generating cross section around the axis of revolution 360°. The spatial discretization is thus only performed on the 2D cross section. The resulting axi-

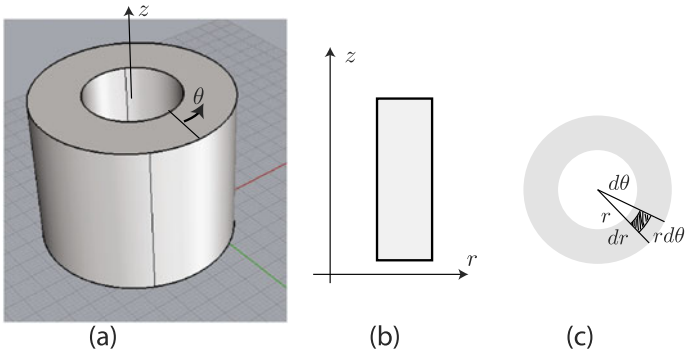


Fig. 2.5 A solid of revolution (**a**) and its two dimensional representation (**b**). We consider a cylindrical coordinate system with coordinates (r, θ, z), and with z along the axis of symmetry

symmetric formulation is quite similar to the 2D one. The major modeling difference is the appearance of the circumferential (or hoop) strain and stress ($\epsilon_{\theta\theta}$ and $\sigma_{\theta\theta}$). Axi-symmetric ULMPM formulations have been presented in Sulsky and Schreyer (1996), Ma et al. (2013), Nairn and Guilkey (2015) and we present them in Sect. 2.7.1. Also discussed is the axi-symmetric for the TLMPM (Sect. 2.7.2).

2.7.1 Axi-Symmetric ULMPM

For the ULMPM, the following are modifications for axi-symmetric problems:

- Particle mass per radian (which varies from particle to particle):

$$m_p := \int_{\Omega_p} \rho r \, \mathrm{d}\Omega = \rho A_p r_p^0 \tag{2.73}$$

  where the coordinates of particle p are $\mathbf{x}_p^t := (r_p^t, z_p^t)$. This can be obtained as follows. If we denote the solid of revolution by Ω_3 and its 2D domain is Ω we then can write the following

$$\begin{aligned} \int_{\Omega_3} f(r,z)\mathrm{d}\Omega &= \int\int\int f(r,z) r \mathrm{d}r\mathrm{d}\theta\mathrm{d}z \\ &= \int_0^{2\pi} \left(\int_\Omega r f(r,z)\mathrm{d}\Omega \right) \mathrm{d}\theta = 2\pi \int_\Omega r f(r,z)\mathrm{d}\Omega \end{aligned} \tag{2.74}$$

  where $\mathrm{d}\Omega = \mathrm{d}r\mathrm{d}z$. By using this formula with $f = \rho$, one gets Eq. (2.73).
- The nodal internal force vector is given by

$$
\begin{aligned}
f_{rI}^{\text{int}} &= -\sum_{p=1}^{n_p} V_p \left[(\sigma_{rr})_p \frac{\partial \phi_I}{\partial r}(\mathbf{x}_p) + (\sigma_{rz})_p \frac{\partial \phi_I}{\partial z}(\mathbf{x}_p) + (\sigma_{\theta\theta})_p \frac{\phi_I(\mathbf{x}_p)}{r_p} \right] \\
f_{zI}^{\text{int}} &= -\sum_{p=1}^{n_p} V_p \left[(\sigma_{rz})_p \frac{\partial \phi_I}{\partial r}(\mathbf{x}_p) + (\sigma_{zz})_p \frac{\partial \phi_I}{\partial z}(\mathbf{x}_p) \right]
\end{aligned}
\tag{2.75}
$$

which is only slightly different from the 2D case—the only modification to f_{rI}^{int} is the third term which comes from the contribution of the energy $\sigma_{\theta\theta}\epsilon_{\theta\theta}$. In the above equation, V_p is the particle volume per radian.

- The initial particle volume per radian is given by

$$V_p^0 = m_p/\rho^0 \tag{2.76}$$

- The velocity gradient matrix is written as

$$\mathbf{L} = \begin{bmatrix} L_{rr} & L_{rz} & 0 \\ L_{zr} & L_{zz} & 0 \\ 0 & 0 & L_{\theta\theta} \end{bmatrix}, \quad L_{\theta\theta} = \sum_I \frac{\phi_I(\mathbf{x}_p)}{r_p} v_{rI} \tag{2.77}$$

The last component $L_{\theta\theta}$ comes from the fact that the hoop strain is defined as $\epsilon_{\theta\theta} = u_r/r$.

Remark 27 If GIMP or CPDI are used, one needs to use the axi-symmetric forms of the GIMP/CPDI weighting functions. Details can be found in Nairn and Guilkey (2015). For other weighting functions, the expressions for 2D plane and 3D problems can be directly used.

2.7.2 Axi-Symmetric TLMPM

For the TLMPM, the following are modifications for axi-symmetric problems:

- Particle mass per radian (which varies from particle to particle):

$$m_p := \int_{\Omega_p^0} \rho_0 R \, \mathrm{d}\Omega = \rho_0 A_p^0 R_p \tag{2.78}$$

where the coordinates of particle p in the reference configuration are $\mathbf{X}_p := (R_p, Z_p)$.

- The nodal internal force vector is given by

$$f_{rI}^{\text{int}} = -\sum_{p=1}^{n_p} V_p^0 \left[(P_{rr})_p \frac{\partial \phi_I}{\partial R}(\mathbf{X}_p) + (P_{rz})_p \frac{\partial \phi_I}{\partial Z}(\mathbf{X}_p) + (P_{\theta\theta})_p \frac{\phi_I(\mathbf{X}_p)}{R_p} \right]$$
$$f_{zI}^{\text{int}} = -\sum_{p=1}^{n_p} V_p^0 \left[(P_{rz})_p \frac{\partial \phi_I}{\partial R}(\mathbf{X}_p) + (P_{zz})_p \frac{\partial \phi_I}{\partial Z}(\mathbf{X}_p) \right] \tag{2.79}$$

which is only slightly different from the 2D case—the only modification to f_{rI}^{int} is the third term which comes from the contribution of the energy $P_{\theta\theta}\epsilon_{\theta\theta}$. In the above equation, V_p is the particle volume per radian.

- The initial particle volume per radian is given by

$$V_p^0 = m_p/\rho^0 \tag{2.80}$$

- The time derivative of the deformation gradient matrix is written as

$$\dot{\mathbf{F}} = \begin{bmatrix} \dot{F}_{rr} & \dot{F}_{rz} & 0 \\ \dot{F}_{zr} & \dot{F}_{zz} & 0 \\ 0 & 0 & \dot{F}_{\theta\theta} \end{bmatrix}, \quad \dot{F}_{\theta\theta} = \sum_I \frac{\phi_I(\mathbf{X}_p)}{R_p} v_{rI}. \tag{2.81}$$

2.8 Adaptive Time Step

As explicit time integrations are only conditionally stable, explicit dynamics MPM must employ a time step smaller than a critical value so that errors will not be so amplified from time step to time step that the error will quickly swamp the solution. In typical explicit MPM simulations, an adaptive time step is employed i.e., the time step is adjusted according to the particle velocities instead of being fixed. One first computes the dilatational wave speed c_{dil}:

$$c_{\text{dil}} = \sqrt{\frac{\lambda + 2\mu}{\rho}} = \sqrt{\frac{K + \frac{4}{3}G}{\rho}} \tag{2.82}$$

where λ, μ are the Lamé constants and K is the bulk modulus and $G = \mu$ denotes the shear modulus.

Next, one computes the maximum wave speed using the following equation (Anderson Jr 1987)

$$\mathbf{c} = (\max_p(c_{\text{dil}} + |v_{xp}|), \max_p(c_{\text{dil}} + |v_{yp}|), \max_p(c_{\text{dil}} + |v_{zp}|)) \tag{2.83}$$

where v_{xp} is the x component of particle p's velocity. For hyper-velocity impact problems, the above equation, where the particle velocity is taken into account, is very much needed.

The time step Δt is then chosen as follows

$$\Delta t = \alpha \min\left(\frac{h_x}{c_x}, \frac{h_y}{c_y}, \frac{h_z}{c_z}\right) \tag{2.84}$$

where (h_x, h_y, h_z) are the cell spacings and α is a time step multiplier ranging from 0 to 1. This factor is needed as the stability analysis was done for linear problems. The above formulation was implemented in the Uintah MPM code.

Remark 28 When a model contains a few very stiff elements, the efficiency of explicit time integration is severely compromised. This is because the time step of the entire model is decided by these very stiff elements. Sub-cycling is a technique where the problem is divided into a number of sub-domains and each sub-domain is integrated in time with its own stable time steps, see Belytschko et al. (2000). This technique has just recently been taken in the computer graphics and they introduced the so-called asynchronous material point method (Hu and Fang 2017).

2.9 Particle/Element Inversion

For extreme large deformation problems, the MPM suffers from the negative Jacobian issue i.e., the element in trouble gets converted. To demonstrate this issue, we consider one 1D element in Fig. 2.6. Due to conversion, node 1 moves to the right of node 2 (its displacement is larger than the element length: $u_1 > L$), and assume, for simplicity, that node 2 is stationary i.e., its displacement is zero. The deformation gradient tensor in this 1D case is given by

$$F = 1 + \frac{\partial u}{\partial X} = 1 + \frac{\partial N_1}{\partial X} u_1 = 1 - u_1/L < 0 \tag{2.85}$$

This is particularily problematic in TLMPM where particles cannot swap positions like in the MPM. A local negative volume can provoke unexpected consequences such as dramatic instabilities. Particle or element inversion happen when large com-

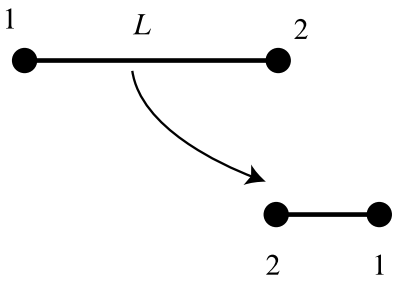


Fig. 2.6 Negative jacobian issue when elements get inverted: 1D illustration

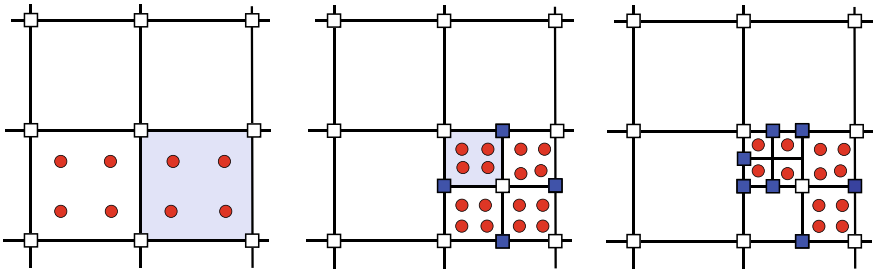


Fig. 2.7 Multi-level grid refinement and particle splitting. Shaded cells are the ones needed to be refined. Solid squares denote hanging nodes

pression and shear strains are experienced. In such cases, the time step needs to be decreased. However, a robust formulation that works well in all circumstances is yet to be found.

2.10 Adaptivity

A basic MPM implementation adopts a fixed uniform Cartesian (or unstructured) grid and a fixed number of material points. In order to better resolve regions of high gradients while keeping a reasonable computational cost, adaptive grid has been proposed, see Sect. 2.10.1. Accompany with grid refinement is particle splitting in which original particles are split into new ones to be added to the new grid cells. Another situation where particle splitting is needed is to prevent numerical fracture. This particle splitting is discussed in Sect. 2.10.2 (Fig. 2.7).

2.10.1 Grid Adaptive Refinement

Adaptive refinement MPM was reported in Tan and Nairn (2002) for fracture mechanics applications. Details are not provided on how to handle hanging nodes. Ma et al. (2006) presented a multi-level grid refinement for GIMP. The standard grid functions are modified to handle hanging nodes, and the modified functions are convoluted with the particle characteristic functions, in the conventional GIMP way, to obtain the final weighting function $\phi_I(\mathbf{x}_p)$. A similar method was given in the community of computer graphics (Gao et al. 2017). Cheon and Kim (2019) reports a similar grid refinement, for the standard MPM, within the context of phase-field fracture simulation. In the field of free surface flows, Mao et al. (2015) reported an adaptive MPM to accurately handle free surfaces. The algorithm allows particle splitting and merging.

2.10.2 Particle Splitting and Merging

We confine to 2D problems for simplicity. Assume that a given particle is split into 4 particles. The mass and volume of the new four particles are 1/4 of the corresponding values for the original particle. All intrinsic material state properties (e.g., density, deformation gradient, stress, damage, etc.) are set equal to that of the original particle (Homel et al. 2016). When it comes to when one should perform particle splitting there exists different criteria, see Ma et al. (2009b), Gracia et al. (2019). Basically, when a particle is stretched too much, it is split. The splitting criterion uses the local particle data such as $\mathbf{F}$, its original/current size and the grid cell size.

We have presented the basic MPM formulation for explicit solid dynamics. This MPM formulation, as simple as it is, can simulate collision of solids involving large deformation where the contact is no-slip, see Fig. 1.24. Frictional contact can be incorporated into this model quite straightforwardly (see Sect. 8.1). However, no details about $\phi_I(\mathbf{x})$ and $\nabla\phi_I(\mathbf{x})$ are specified yet. The next section is devoted to this topic.

References

Al-Kafaji, I.K.J.: Formulation of a Dynamic Material Point Method (MPM) for Geomechanical Problems. PhD thesis, University of Stuttgart (2013)

Anderson Jr, C.E.: An overview of the theory of hydrocodes. Int. J. Impact Eng. **5**(1–4), 33–59 (1987)

Bardenhagen, S.G.: Energy Conservation Error in the Material Point Method for Solid Mechanics. J. Comput. Phys. **180**(1), 383–403 (2002)

Belytschko, T., Liu, W.K., Moran, B.: Nonlinear Finite Elements for Continua and Structures. Wiley, Chichester, England (2000)

Benson, D.J.: Computational methods in Lagrangian and Eulerian hydrocodes. Comput. Methods Appl. Mech. Eng. **99**(2–3), 235–394 (1992)

Brackbill, J.U., Ruppel, H.M.: FLIP: a method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. J. Comput. Phys. **65**(2), 314–343 (1986)

Burgess, D., Sulsky, D., Brackbill, J.U.: Mass matrix formulation of the FLIP particle-in-cell method. J. Comput. Phys. **103**(1), 1–15 (1992)

Cheon, Y-J., Kim, H-G.: An adaptive material point method coupled with a phase-field fracture model for brittle materials. Int. J. Numer. Methods Eng. (2019)

de Vaucorbeil, A., Nguyen, V.P.: Modeling contacts with a total lagrangian matertial point method. Comput. Methods Appl. Mech. Eng. **360**, 112783 (2021). https://doi.org/10.1016/j.cma.2019.112783

de Souza Neto, E.A., Peric, D., Owen, D.R.J.: Computational Methods for Plasticity: Theory and Applications. Wiley (2011)

de Vaucorbeil, A., Phu Nguyen, V., Hutchinson, C.R.: A total-Lagrangian material point method for solid mechanics problems involving large deformations. Comput. Methods Appl. Mech. Eng. **360**, 112783 (2020). https://doi.org/10.1016/j.cma.2019.112783

Dolbow, J., Belytschko, T.: Numerical integration of the galerkin weak form in meshfree methods. Comput. Mech. **23**(3), 219–230 (1999)

Gao, M., Tampubolon, A.P., Jiang, C., Sifakis, E.: An adaptive generalized interpolation material point method for simulating elastoplastic materials. ACM Trans. Graphics (TOG) **36**(6), 223 (2017)

Gracia, F., Villard, P., Richefeu, V.: Comparison of two numerical approaches (DEM and MPM) applied to unsteady flow. Comput. Particle Mech. 1–19 (2019)

Gurtin, M.E.: An Introduction to Continuum Mechanics. Academic, New York (1981)

Holzapfel, G.A.: Nonlinear Solid Mechanics. Wiley, New York (2000)

Homel, M.A., Brannon, R.M., Guilkey, J.: Controlling the onset of numerical fracture in parallelized implementations of the material point method (MPM) with convective particle domain interpolation (CPDI) domain scaling. Int. J. Numer. Meth. Eng. **107**(1), 31–48 (2016)

Hu, Y., Fang, Y.: An asynchronous material point method. In: ACM SIGGRAPH 2017 Posters, p. 60. ACM (2017)

Iaconeta, I., Larese, A., Rossi, R., Guo, Z.: Comparison of a material point method and a Galerkin meshfree method for the simulation of cohesive-frictional materials. Materials **10**(10), 1150 (2017)

Leroch, S., Eder, S.J., Ganzenmüller, G., Murillo, L.J.S., Rodríguez Ripoll, M.: Development and validation of a meshless 3D material point method for simulating the micro-milling process. J. Mater. Process. Technol. **262**, 449–458 (2018)

Ma, J., Lu, H., Komanduri, R.: Structured mesh refinement in generalized interpolation material point (GIMP) method for simulation of dynamic problems. Comput. Model. Eng. Sci. **12**, 213–227 (2006)

Ma, S., Zhang, X., Qiu, X.M.: Comparison study of MPM and SPH in modeling hypervelocity impact problems. Int. J. Impact Eng **36**(2), 272–282 (2009)

Ma, S., Zhang, X., Lian, Y., Zhou, X.: Simulation of high explosive explosion using adaptive material point method. Comput. Model. Eng. Sci. (CMES) **39**(2), 101 (2009)

Ma, X., Zhang, D.Z., Giguere, P.T., Liu, C.: Axisymmetric computation of taylor cylinder impacts of ductile and brittle materials using original and dual domain material point methods. Int. J. Impact Eng **54**, 96–104 (2013)

Malvern, L.E.: Introduction to the Mechanics of a Continuous Medium. Prentice-Hall International, Englewood Cliffs, New Jersey (1969)

Mao, S., Chen, Q., Li, D., Feng, Z.: Modeling of free surface flows using improved material point method and dynamic adaptive mesh refinement. J. Eng. Mech. **142**(2), 04015069 (2015)

Marsden, J.E., Hughes, T.J.R.: Mathematical Foundations of Elasticity. Prentice-Hall, Englewood Cliffs, New Jersey (1983)

Moresi, L., Dufour, F., Mühlhaus, H.-B.: A Lagrangian integration point finite element method for large deformation modeling of viscoelastic geomaterials. J. Comput. Phys. **184**(2), 476–497 (2003)

Moresi, L., Quenette, S., Lemiale, V., Mériaux, C., Appelbe, B., Mühlhaus, H.-B.: Computational approaches to studying nonlinear dynamics of the crust and mantle. Phys. Earth Planet. Inter. **163**(1–4), 69–82 (2007)

Nairn, J.A.: Material Point Method Calculations with Explicit Cracks. Comput. Model. Eng. Sci. **4**(6), 649–663 (2003)

Nairn, J.A., Guilkey, J.E.: Axisymmetric form of the generalized interpolation material point method. Int. J. Numer. Meth. Eng. **101**(2), 127–147 (2015)

Ogden, R.W.: Non-linear Elastic Deformations. Ellis Harwood Ltd, Chichester, England (1984)

Simo, J.C., Hughes, T.J.R.: Computational Inelasticity. Springer, London (1998)

Steffen, M., Kirby, R.M., Berzins, M.: Analysis and reduction of quadrature errors in the material point method (MPM). Int. J. Numer. Meth. Eng. **76**(6), 922–948 (2008)

Stomakhin, A., Schroeder, C., Chai, L., Teran, J., Selle, A.: A material point method for snow simulation. ACM Trans. Graphics **32**(4), 1 (2013)

Sulsky, D., Schreyer, H.L.: Axisymmetric form of the material point method with applications to upsetting and Taylor impact problems. Comput. Methods Appl. Mech. Eng. **139**, 409–429 (1996)

Sulsky, D., Chen, Z., Schreyer, H.L.: A particle method for history-dependent materials. Comput. Methods Appl. Mech. Eng. **5**, 179–196 (1994)
Sulsky, D., Zhou, S.J., Schreyer, H.L.: Application of a particle-in-cell method to solid mechanics. Comput. Phys. Commun. **87**(1–2), 236–252 (1995)
Tan, H., Nairn, J.A.: Hierarchical, adaptive, material point method for dynamic energy release rate calculations. Comput. Methods Appl. Mech. Eng. **191**(19–20), 2123–2137 (2002)
Wallstedt, P.C., Guilkey, J.E.: An evaluation of explicit time integration schemes for use with the generalized interpolation material point method. J. Comput. Phys. **227**(22), 9628–9642 (2008)
Zhang, X., Chen, Z., Liu, Y.: The Material Point Method: A Continuum-Based Particle Method for Extreme Loading Cases. Academic (2016b)
Zhu, Y., Bridson, R.: Animating sand as a fluid. ACM Trans. Graphics **24**(3), 965–972 (2005)
Zienkiewicz, O.C., Taylor, R.L.: The Finite Element Method for Solid and Structural Mechanics, 6th edn. Butterworth-Heinemann, Oxford, UK (2006)

Chapter 3
Various MPM Formulations

A general framework for the MPM has been presented in Chap. 2 in which the shape functions have not yet been specified. In this section, various shape functions ranging from the standard hat functions (or linear Lagrange functions), see Sect. 3.2, generalized interpolation MPM (Sect. 3.3), B-splines (Sect. 3.4), Bernstein functions (Sect. 3.5), convected particle domain integrator (Sect. 3.6) are discussed. Finally, the generalized particle in cell (GPIC) is presented in Sect. 3.7. GPIC is a combination of the TLFEM and ULMPM which is efficient and accurate for large deformation contact problems.

A note on terminology of $\phi_I(\mathbf{x})$ and $\nabla\phi_I(\mathbf{x})$ is worthy here. As will be seen in Sect. 3.3, $\phi_I(\mathbf{x})$ is constructed as a convolution of the linear/bilinear/trilinear FE shape functions $N_I(\mathbf{x})$ with the particle characteristic function. Therefore, it is not rigorous to call $\phi_I(\mathbf{x})$ grid basis functions. Therefore, in the remaining of this book, $\phi_I(\mathbf{x})$ is referred to as weighting function and $\nabla\phi_I(\mathbf{x})$ the weighting gradient.

Remark 29 We restrict, for now, our discussion to MPM formulations using a Cartesian grid. There exists MPM variants that adopt unstructured grids. See Sect. 5.4 for a discussion on this topic.

3.1 Properties of Weighting Functions

The weighting functions $\phi_I(\mathbf{x})$ should satisfy all the following properties in addition to being smooth and continuous across the cell boundaries

Partition of Unity (PU) $\sum_I \phi_I(\mathbf{x}) = 1$ for all $\mathbf{x}$.

Compact support $\phi_I(\mathbf{x}) \neq 0$ for just points $\mathbf{x}$ close to node I.

V. P. Nguyen et al., *The Material Point Method*, Scientific Computation,
https://doi.org/10.1007/978-3-031-24070-6_3

Non-negativity $\phi_I(\mathbf{x}) \geq 0$ for all $\mathbf{x}$.

Kronecker delta property $\phi_I(\mathbf{x}_J) = \delta_{IJ}$.

The PU property defines completeness (i.e., the ability to represent rigid motions and constant strains) which is required for convergence, see Hughes (2000) and Appendix A.2. Compact support is for efficiency and non-negativity ensures positive nodal mass when a lumped mass is used. The use of second-order finite elements implies the use of shape functions having negative values in their domain. This may lead to negative mass at some of the grid points, possibly causing instability of the solution scheme (Andersen and Andersen 2010). The Kronecker delta property should be satisfied at least at the solid boundaries so that enforcement of Dirichlet boundary conditions is straightforward.

3.2 Standard Linear Basis Functions

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

$$N_I^x(x) = \begin{cases} 1 - |x - x_I|/h_x & \text{if} |x - x_I| \leq h_x \\ 0 & \text{else} \end{cases} \tag{3.1}$$

where h_x denotes the nodal spacing or element size in the x direction. Its derivatives are given by

$$N_{I,x}^x(x) \equiv \frac{dN_I^x(x)}{dx} = \begin{cases} -\text{sign}(x - x_I)/h_x & \text{if} |x - x_I| \leq h_x \\ 0 & \text{else} \end{cases} \tag{3.2}$$

where $\text{sign}(x)$ is the signum function.

For 2D, the shape functions are simply the tensor-product of the two shape functions along the x and y directions

$$N_I(x, y) = N_I^x(x) N_I^y(y) \tag{3.3}$$

And the derivatives of the shape functions are given by

$$\nabla N_I(x, y) = \begin{bmatrix} N_{I,x}^x(x) N_I^y(y) \\ N_I^x(x) N_{I,y}^y(y) \end{bmatrix} \tag{3.4}$$

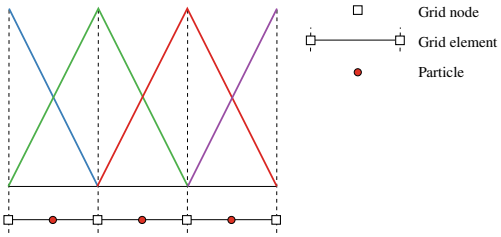


Fig. 3.1 Hat shape functions for a series of three elements in 1D

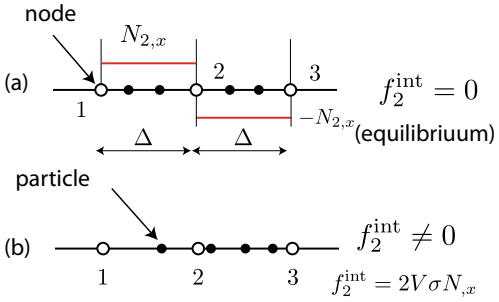


Fig. 3.2 Cell crossing issue in MPM. As the shape functions are linear, the derivatives of the shape functions which are used to form the divergence are piecewise constant over the elements. More important is that the derivatives are discontinuous across the cell boundary (change sign)

In the first MPM, the weighting functions are simply this hat function i.e., $\phi_I(\mathbf{x}) = N_I(\mathbf{x})$. This MPM is refered to as the standard MPM in this text. An illustration of the hat shape functions is given in Fig. 3.1 for a series of three elements in 1D.

Cell crossing instability. The original MPM with C^0 shape functions suffers from the so-called cell crossing instability. To illustrate this phenomenon, consider a 1D MPM discretization shown in Fig. 3.2 in which all particles have the same stress (i.e., uniform stress state), the same volume and a uniform element size. Each element has two particles (Fig. 3.2a). The internal force at node 2 is identically zero in the absence of body force. When a particle has just moved to a new cell, Fig. 3.2b, the internal force at node 2 is non-zero resulting in non-equilibrium. To demonstrate the issue of this non-equilibrium, we consider a simple one dimensional MPM simulation shown in Fig. 3.3a where two particles are moving with a constant velocity towards each other. The grid consists of 6 cells with cell size is 1/6. While the particle stresses are identically zero, before collision, and thus causing no issue as the particles travel, after collision they become non-zero. And right after the particles cross the cells, the stress field is spoiled and there is a sudden increase of the strain energy (Fig. 3.3b/c).

The cell-crossing issue is more severe in static analyses where there is no inertia force. For structure and material failure modeling where the onset of failure is most often based on stresses, cell-crossing issue must be completely avoided. Fine meshes are preferred for accuracy, but they are more prone to cell crossing. A simple but inefficient way to prevent this is to adopt more material points and smaller time steps, cf. Fig. 3.4 where three particles per cell are used. Another option is to not to reset the grid at the end of every steps as done in Guilkey et al. (2006): particles never

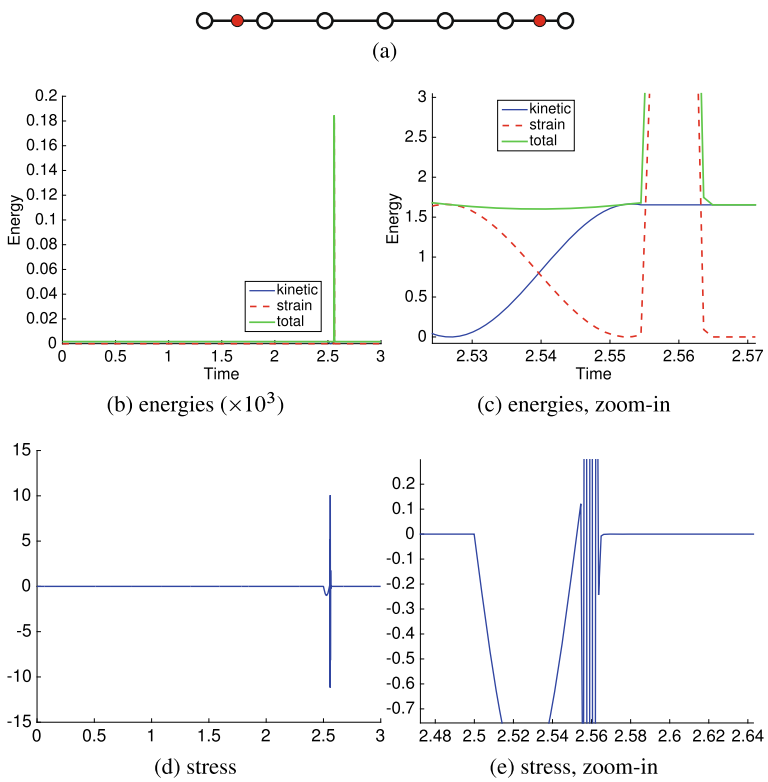


Fig. 3.3 Simple example to demonstrate the cell-crossing issue of MPM: **a** problem setup, **b** plot of strain and kinetic energies showing the erroneous strain energy and **c** zoom-in plot. The stress of the left particle is depicted in (**c**, **d**)

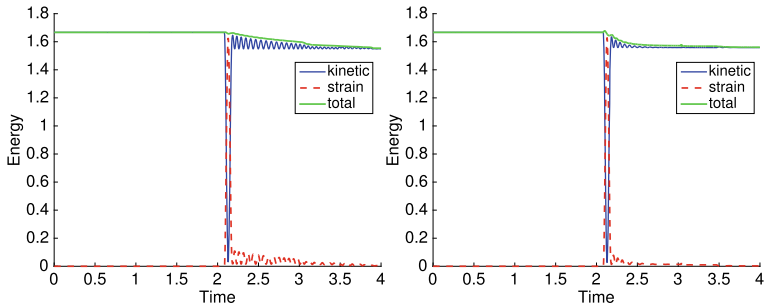


Fig. 3.4 Using more particles per cell can reduce the impact of the cell-crossing issue

move out of the deformed grid. The resulting algorithm is very close to the ULFEM but the extent of mesh distortion is much lower as the initial mesh contains elements with right angle corners.

Better methods to mitigate this error include the use of high order B-splines basis functions, the generalized interpolation material point (GIMP) method (and its variants such as CPDI), the use of modified gradient of shape functions in the dual domain MPM (Zhang et al. 2011) and the total Lagrangian MPM (Sect. 2.6). All these methods also improve the quadrature of the MPM.

For quasi-static problems, Beuth et al. (2011) proposed to use Gauss integration to remove the cell-crossing error. The idea is that Gauss points never leave the elements. In their formulation, also used in later work of Jassim et al. (2013), one uses Gauss points not MPs, whose data are interpolated from the MPs, to integrate the internal force vector (which is the cause of the problem) for fully filled elements. In Alonso and Zabala (2011) a simple procedure that can be used to reduce this type of instability is to consider a constant stress at each cell equal to the stress average of the particles that are currently within the cell. In this case, the internal forces are obtained in the same way as in the FEM when one point of integration is used, using the gradient of the shape functions evaluated at the cell center. This idea is used later in the improved MPM of Sulsky and Gong (2016), see Chap. 9.

3.3 Generalized Interpolation Material Point (GIMP)

There are different ways to develop GIMP and Bardenhagen and Kober (2004) derived the formulation using the Petrov-Galerkin method. We adopt a simpler view taken by Steffen et al. (2008b)–the projection from particles to nodes for a scalar field g can be written as

$$g_I = \sum_p g_p N_I(\mathbf{x}_p) = \sum_p g_p \frac{\int_\Omega N_I(\mathbf{x})\delta(\mathbf{x}-\mathbf{x}_p)\mathrm{d}\Omega}{\int_\Omega \delta(\mathbf{x}-\mathbf{x}_p)\mathrm{d}\Omega} \tag{3.5}$$

where δ is the Dirac delta and N_I are the standard grid functions, presented in Sect. 3.2. In GIMP, one replaces the Dirac delta with a general function $\chi(\mathbf{x})$ called the *particle characteristic function* and the resulting projection is given by

$$g_I = \sum_p g_p \phi_I(\mathbf{x}_p) \tag{3.6}$$

where $\phi_I(\mathbf{x}_p)$ is given by

$$\phi_{Ip} \equiv \phi_I(\mathbf{x}_p) = \frac{1}{V_p}\int_{\Omega_p} \chi(\mathbf{x}-\mathbf{x}_p) N_I(\mathbf{x})\mathrm{d}\Omega \tag{3.7}$$

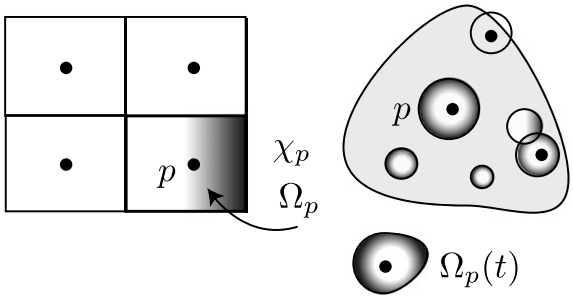


Fig. 3.5 Particle domain and particle characteristic function in GIMP: rectangular particle domains can fill the initial material domain without overlapping (left) and overlapping circular particle domains commonly used in meshfree methods (right). Note also that particle domain is evolving in time as the material deforms

and the short notation ϕ_{Ip} was introduced to represent $\phi_I(\mathbf{x}_p)$; Ω_p denotes the particle domain. Figure 3.5 illustrates the concept of particle domains in the MPM and in meshfree approximations. GIMP weighting functions, as defined by the above equation, are the convolution of the characteristic function and the grid basis function normalized by the particle volume.

The particle characteristic functions must satisfy the partition of unity property in the reference undeformed configuration (Bardenhagen and Kober 2004)

$$\sum_p \chi_p(\mathbf{x}, t = 0) = 1 \quad \forall \mathbf{x} \tag{3.8}$$

Note that Bardenhagen and Kober (2004) placed no such constraint on the characteristic functions in the deformed configuration due to the potential existence of gaps between the different particles' domains. However, the CPDI of Sadeghirad et al. (2011) ensures the PU of $\chi_p(\mathbf{x}, t)$ in the deformed configuration as it closely tracks the particle domains.

Typically piece-wise constant particle characteristic functions that satisfy the PU given in Eq. 3.8 are used

$$\chi_p(\mathbf{x}) = \begin{cases} 1 & \text{if} \mathbf{x} \in \Omega_p \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

which implies that the material domain is partitioned into non-overlapping rectangular particle domains (in 2D) as shown in Fig. 3.5. This characteristic function is known as the "top-hat" function. This particular particle characteristic function results in the following GIMP weighting functions (simplification of Eq. 3.7)

$$\phi_{Ip} = \frac{1}{V_p} \int_{\Omega_p} N_I(\mathbf{x}) \mathrm{d}\Omega \tag{3.10}$$

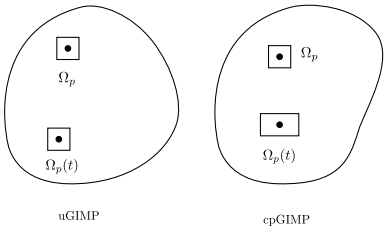


Fig. 3.6 Tracking particle domain in GIMP: space cannot be tiled in a general multi-dimension domain using rectilinear Ω_p

Since the GIMP functions depend on the particle domain Ω_p which in turn evolves in time, the GIMP functions are particle specific and time-dependent. Differences in how the integral in Eq. 3.10 is evaluated and how the particle domains are defined/updated result in various GIMP formulations. Figure 3.6 illustrates existing GIMP methods to be discussed in what follows.

In the approach commonly referred to as uGIMP (unchanged GIMP) Ω_p is kept fixed and the integral in Eq. 3.10 can be exactly integrated resulting in analytical expressions for ϕ_{Ip}. Therefore, uGIMP is more effective than GIMP formulation. However, as material deforms, the unstretched particle domains cannot fill the material space. A more complicated approach, known as cpGIMP (contiguous particle GIMP), updates the particle domain using the deformation gradient $\mathbf{F}$ without taking shear deformation into account. Analytical expression for the weighting functions and its derivatives are available. Nonetheless, the updated particle domain is a axis-aligned rectangle in 2D and space cannot be tiled particularly for shearing. Andersen (2009) used Gaussian quadrature to numerically evaluate the GIMP basis functions on the fully updated particle domain (the particle domain is a parallelogram in 2D). But, it is very computationally expensive and thus should not be employed. Convected Particle Domain Interpolation (CPDI) (Sadeghirad et al. 2011, 2013) is the next logical development of GIMP where particles are given parallelogram-shaped domains that are constantly updated using the deformation gradient evaluated at the particle location. The novelty in CPDI is that the integrals in Eq. 3.10 are also analytically evaluated thanks to the use of alternative basis functions. CPDI will be presented in Sect. 3.6.

3.3.1 uGIMP

In uGIMP and cpGIMP, the following one dimensional particle characteristic function is employed

$$\chi_p(x) = \begin{cases} 1 & \text{if} |x - x_p| \leq l_p/2 \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

Here l_p is the current particle size. The initial particle size is determined by dividing the cell spacing h_x by the number of particles per cell. Equation 3.10 is reduced to

$$\phi_{Ip} = \frac{1}{l_p} \int_{x_p - l_p/2}^{x_p + l_p/2} N_I(x) \mathrm{d}x \tag{3.12}$$

and after substitution of the standard FE hat function $N_I(x)$, cf. Eq. 3.1, into the above, one obtains the uGIMP function (Steffen et al. 2008b)

$$\phi(x) = \begin{cases} 1 - (4x^2 + l_p^2)/(4h_x l_p) & \text{if} |x| < 0.5 l_p \\ 1 - |x|/h & \text{if} 0.5 l_p \leq |x| \leq h_x - 0.5 l_p \\ (h + l_p/2 - |x|)^2/(2h_x l_p) & \text{if} h_x - 0.5 l_p \leq |x| < h_x + 0.5 l_p \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

of which an example is given in Fig. 3.7.[1] The GIMP functions are C^1 across the cell boundaries, have support in adjacent cells and their next nearest neighbors. Note also that there are four non-zero basis functions within one cell. But, for a given particle in a cell there are only three non-zero functions. 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.

The first derivative of the GIMP weighting function is given by

$$\phi_{,x} = \begin{cases} -8x/(4h_x l_p) & \text{if} |x| < 0.5 l_p \\ -(1/h_x)\mathrm{sign}(x) & \text{if} 0.5 l_p \leq |x| \leq h_x - 0.5 l_p \\ -\mathrm{sign}(x)(h_x + l_p/2 - |x|)/(h_x l_p) & \text{if } h_x - 0.5 l_p \leq |x| < h_x + 0.5 l_p \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

where $\mathrm{sign}(x)$ is the *signum* function.

In 2D the particle domain is a rectangle defined as $l_p^x \times l_p^y$ and the GIMP functions are the tensor product of the 1D functions i.e., $\phi(x, y) = \phi(x, l_p^x)\phi(y, l_p^y)$. An example of 2D uGIMP functions is given in Fig. 3.8. In 3D, the particle domain is a cube and the weighting function is defined similarly. For a given particle p there are 9/27 non-zero basis functions ϕ_{Ip} for 2D and 3D problems, respectively.

3.3.2 cpGIMP

In the cpGIMP, the particle domain is tracked by updating the particle sizes in the x and y directions

$$\begin{aligned} l_p^x(t + \Delta t) &= l_p^x(t = 0) F_{xx}(t + \Delta t) \\ l_p^y(t + \Delta t) &= l_p^y(t = 0) F_{yy}(t + \Delta t) \end{aligned} \tag{3.15}$$

[1] Matlab scripts used to generate this plot are presented in Sect. C.1.

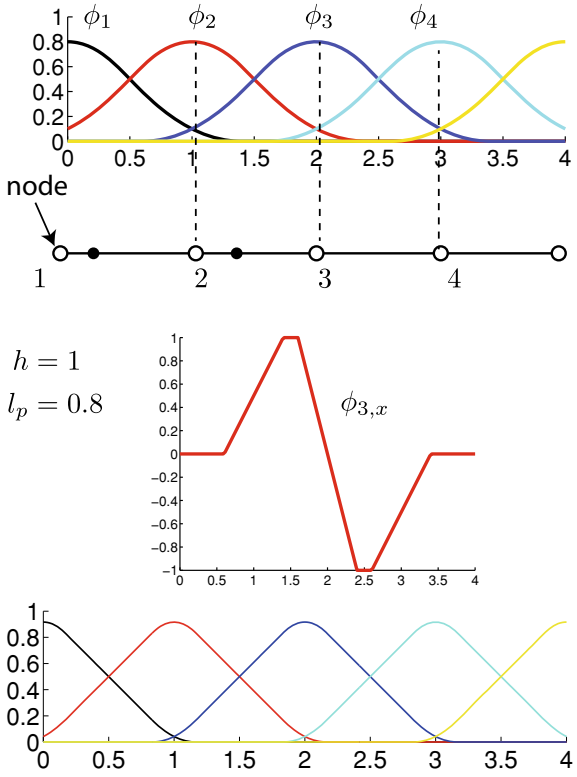


Fig. 3.7 One dimensional GIMP basis functions $\phi_I = \phi(x - x_I)$: basis functions (top row), first derivative (middle row) and GIMP functions are reduced to MPM basis functions as l_p decreases (bottom row)

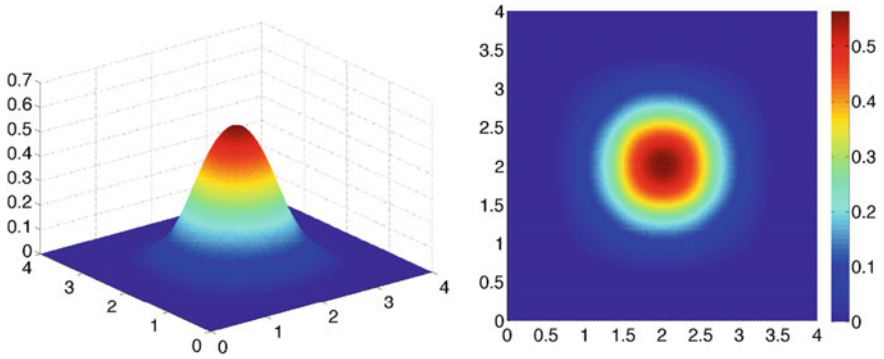


Fig. 3.8 Two dimensional GIMP basis functions: illustrated for node at (2, 2) on a square domain 4×4 discretized by 16 elements of spacing 1 (de Vaucorbeil et al. 2020)

i.e., the deformed domain is stretched in orthogonal directions but is never sheared. In other words, particle domains that start as squares (in 2D) or cubes (in 3D) will deform to rectangles (in 2D) or orthogonal boxes (in 3D), respectively. Accordingly, the cpGIMP is limited to problems for which deformation is along the grid directions so that off-diagonal deformation gradient components are negligible.

3.4 B-Splines Basis Functions

Steffen et al. (2008a, b) showed that for simple problems, the use of cubic splines improves the spatial convergence properties of the MPM as grid-crossing errors are reduced. Cubic B-splines were also adopted in the amazing snow animations given in Stomakhin et al. (2013) for the Frozen movie. There are different ways to construct the B-splines basis functions, namely through a recurrence or a convolution concept. The latter was used in Steffen et al. (2008a, b); Stomakhin et al. (2013) and other MPM references. However, we present herein only the B-splines constructed using a recursive formula not only because we are familiar with them based on our work on isogeometric analysis but also because they are general.

3.4.1 Recursive B-Splines

Given a knot vector $\Xi^1 = \{\xi_1, \xi_2, \ldots, \xi_{n+k+1}\}$, which is defined as an ordered set of increasing parameter values, the associated set of B-spline basis functions $\{N_{i,k}\}_{i=1}^{n}$ are defined recursively by the Cox-de-Boor formula (Piegl and Tiller 1996), starting with the zeroth order basis function ($k = 0$)

$$N_{i,0}(\xi) = \begin{cases} 1 & \text{if} \xi_i \leq \xi < \xi_{i+1}, \\ 0 & \text{otherwise}, \end{cases} \tag{3.16}$$

and for a polynomial order $k \geq 1$

$$N_{i,k}(\xi) = \frac{\xi - \xi_i}{\xi_{i+k} - \xi_i} N_{i,k-1}(\xi) + \frac{\xi_{i+k+1} - \xi}{\xi_{i+k+1} - \xi_{i+1}} N_{i+1,k-1}(\xi). \tag{3.17}$$

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

High order B-spline basis functions are C^{k-1} not C^0 as high order Lagrange polynomial basis, the connectivity of elements is, therefore, different from standard finite elements. Elements are defined as non-zero knot spans. Note that the B-splines functions are not interpolatory except at the boundaries when open knots are used. Open knots are those where the first and last knots are repeated $p + 1$ times. Open knots facilitate the imposition of Dirichlet boundary conditions. B-splines elements

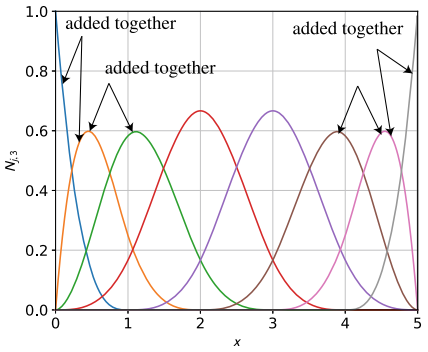


Fig. 3.9 One dimensional cubic ($k = 3$) B-spline basis functions on an open uniform knot $\Xi = \{0, 0, 0, 0, 1, 2, 3, 4, 5, 5, 5, 5\}$ (de Vaucorbeil et al. 2020). There are 8 nodes (control points in CAD terminology) and 5 elements (or knot spans in CAD). At any point, there are 4 ($= k + 1$) non-zero basis functions. Therefore each element has 4 nodes. The first element's connectivity is [1, 2, 3, 4] i.e. particles locate in this element contribute to nodes 1, 2, 3 and 4. The second element's connectivity is [2, 3, 4, 5] and so on (Hughes et al. 2005). These functions are modified to get ones in Fig. 3.10

are used extensively in Isogeometric Analysis (Hughes et al. 2005) which is a computational paradigm that reduces the gap between Computer Aided Design (CAD) and Finite Element Analysis (FEA).

To illustrate B-splines, we consider cubic B-spline basis functions for a uniform knot vector $\Xi = \{0, 0, 0, 0, 1, 2, 3, 4, 5, 5, 5, 5\}$ (Fig. 3.9). The knot vector is made of 5 knot spans and there are 8 basis functions. Even though it is possible to use this B-splines in the MPM, see Sect. 6.8.1, we want to modify them to get only 6 basis functions centered at the six nodes, or to have exactly $n + 1$ nodes (and basis functions) for a mesh of n cells (1D). This is just a matter of implementation as the B-splines grid is now exactly the same as the grid that uses hat functions. Furthermore, analytical forms for these modified B-splines can be derived and thus result in a more efficient code. In what follows, we present boundary modified cubic B-splines and refer to Appendix C.3 for quadratic splines.

3.4.2 Boundary Modified B-Splines

To use the knot spans as elements in the manner of Cartesian grid commonly used in the MPM, one needs, in this example, 6 shape functions not 8. Therefore, there are two more basis functions than the number of shape functions required. The right number of shape function is obtained by (1) replacing the first function by itselft plus one third of the second function, and (2) combining the two basis functions (on each side) that do not peak at the junction between two elements to obtain the new one

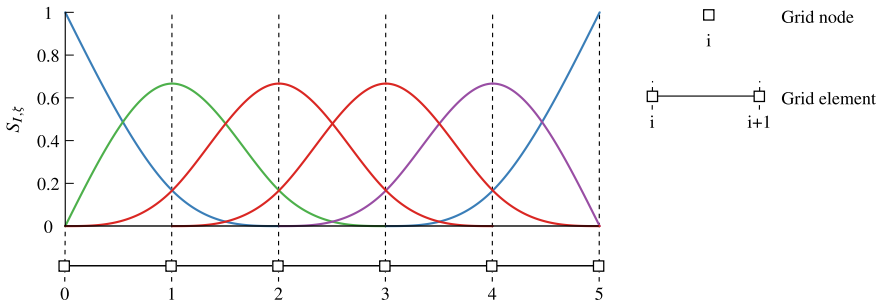


Fig. 3.10 Cubic B-spline shape functions for a series of five elements in 1D. Note that there are now just 6 basis functions (de Vaucorbeil et al. 2020)

dimensional shape functions, now denoted by $S_{I,\zeta}$ where ζ corresponds to any axis x, y or z, plotted on Fig. 3.10. By doing this, the partition of unity is respected and all elements have the same size. We refer to Sect. C.3 for a detailed derivation.

Because of the presence of boundaries, there are four different types of shape functions $S_{i,\zeta}$ which differ by the position of node I with respect to the boundaries. They are represented on Fig. 3.10 by different colours and their expressions are, with $r = (\zeta_p - \zeta_i)/h$

- Shape functions of **type 1** (blue in Fig. 3.10): the node I is located at the boundary, i.e. $\zeta_I = \zeta_B$, and have the following form:

$$S^1_{I,\zeta}(r) = \begin{cases} \frac{1}{6}r^3 + r^2 + 2r + \frac{4}{3}, & -2 \le r \le -1 \\ -\frac{1}{6}r^3 + r + 1, & -1 \le r \le 0 \\ \frac{1}{6}r^3 - r + 1, & 0 \le r \le 1 \\ -\frac{1}{6}r^3 + r^2 - 2r + \frac{4}{3}, & 1 \le r \le 2 \end{cases}, \tag{3.18}$$

- Shape functions of **type 2** (green in Fig. 3.10): the node I is located on the right side of the closest boundary one cell away from it, i.e. $\zeta_I = \zeta_B + h$, and have the following form:

$$S^2_{I,\zeta}(r) = \begin{cases} -\frac{1}{3}r^3 - r^2 + \frac{2}{3}, & -1 \le r \le 0 \\ \frac{1}{2}r^3 - r^2 + \frac{2}{3}, & 0 \le r \le 1 \\ -\frac{1}{6}r^3 + r^2 - 2r + \frac{4}{3}, & 1 \le r \le 2 \end{cases} \tag{3.19}$$

- Shape functions of **type 3**: the node I is located at least two cells away from any boundary, i.e. $\zeta_I \ge \zeta_B + 2h$, and have the following form:

$$S^3_{I,\xi}(r) = \begin{cases} \frac{1}{6}r^3 + r^2 + 2r + \frac{4}{3}, & -2 \le r \le -1 \\ -\frac{1}{2}r^3 - r^2 + \frac{2}{3}, & -1 \le r \le 0 \\ \frac{1}{2}r^3 - r^2 + \frac{2}{3}, & 0 \le r \le 1 \\ -\frac{1}{6}r^3 + r^2 - 2r + \frac{4}{3}, & 1 \le r \le 2 \end{cases} \tag{3.20}$$

- Shape functions of **type 4**: the node I is located on the left side of the closest boundary, one cell away from it, i.e. $\zeta_I = \zeta_B - h$, and have the following form:

$$S^4_{I,\xi}(r) = \begin{cases} \frac{1}{6}r^3 + r^2 + 2r + \frac{4}{3}, & -2 \le r \le -1 \\ -\frac{1}{2}r^3 - r^2 + \frac{2}{3}, & -1 \le r \le 0 \\ \frac{1}{3}r^3 - r^2 + \frac{2}{3}, & 0 \le r \le 1 \end{cases} \tag{3.21}$$

The two dimensional and three-dimensional shape functions are obtained as the product of the one-dimensional shape functions as:

$$\begin{cases} \phi_I(\mathbf{x}_p) & = S_{I,x}\left(\dfrac{x_p - x_I}{h_x}\right) \times S_{I,y}\left(\dfrac{y_p - y_I}{h_y}\right) \text{ in 2D} \\ \phi_I(\mathbf{x}_p) & = S_{I,x}\left(\dfrac{x_p - x_I}{h_x}\right) \times S_{I,y}\left(\dfrac{y_p - y_I}{h_y}\right) \times S_{I,z}\left(\dfrac{z_p - z_I}{h_z}\right) \text{ in 3D} \end{cases} \tag{3.22}$$

These four types of one-dimensional shape functions $S_{i,\zeta}$ translate into $4^2 = 16$ two dimensional shape functions, and into $4^3 = 64$ three dimensional shape functions: each node I having a different type along the respective axes x, y, and z. Unless otherwise stated, in the following, when cubic B-splines are used, each background cell is populated by 2, 4 and 8 material points in 1D, 2D and 3D, respectively. These particles will be located at positions defined by $\xi_1 = 0.2113$ and $\xi_2 = 0.7887$. And these locations are determined from the Gauss quadrature rule.

3.5 Bernstein Functions

Bernstein polynomials form a basis for the Bézier elements used in isogeometric analysis (Hughes et al. 2005). These polynomials are used in CAD to construct the so-called Bézier curves/surfaces (Piegl and Tiller 1996). The univariate Bernstein basis functions of order k are defined over the biunit interval [0, 1] as:

$$B_{i,k}(\xi) = \binom{k}{i} \xi^i \, (1 - \xi)^{k-i} \tag{3.23}$$

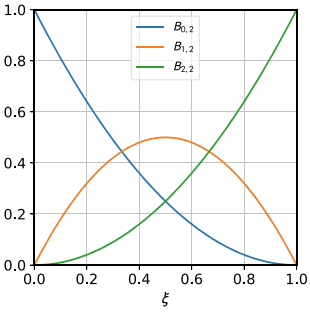


Fig. 3.11 Bernstein basis polynomials of degree 2

where the binomial coefficient $\binom{k}{i} = \frac{k!}{i!(k-i)!}$ for $1 \leq i \leq k+1$. Bernstein polynomials of degree 2 are plotted in Fig. 3.11. These polynomials form a partition of unity: $\sum_{i=1}^{k} B_{i,k}(\xi) = 1$.

Each cell of the background mesh is made of 3 (in 1D), 9 (in 2D), or 27 (in 3D) nodes. As some of these nodes are common to different cells, we express the shape functions as a function of the normalized distance between a particle p and a node I, i.e. $(\mathbf{x}_I - \mathbf{x}_p)/h$:

$$\phi_I(\mathbf{x}_p) = S_{I,x}\left(\frac{x_I - x_p}{h_x}\right) \times S_{I,y}\left(\frac{y_I - y_p}{h_y}\right) \times S_{I,z}\left(\frac{z_I - z_p}{h_z}\right) \tag{3.24}$$

where $S_{I,\zeta}$ are the shape functions along the axis ζ (i.e. x, y or z). The shape function depends on the position of the nodes in a cell: if it is located on an edge or the cell center along the axis i, they take two different forms: if the node I is located on an edge of a mesh element along the axis ζ:

$$S_{i,\zeta}(r) = B_{0,2}(|r|) = \begin{cases} (1-|r|)^2 & \text{if } -1 \leq r \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.25}$$

otherwise, i.e. if the node I is located at the center (or inside) an element along the axis ζ:

$$S_{i,\zeta}(r) = B_{1,2}\left(|r| + \frac{1}{2}\right) = \begin{cases} \frac{1}{2} - 2r^2 & \text{if } -1/2 \leq r \leq 1/2 \\ 0 & \text{otherwise} \end{cases} \tag{3.26}$$

Figure 3.12 shows these functions over a grid of three cells. As Bernstein functions are smooth but still C^0 across the cell boundaries, they are not recommended for adoption in any MPM variant except the TLMPM.

In this work, unless otherwise stated, when using Bernstein shape functions, each background cell is populated by 3, 9 and 27 material points in 1D, 2D and 3D, respectively. The particles will be located at positions defined by $\xi_{1,2 \text{ and } 3} = 0.1127$, 0.5, and 0.8873.

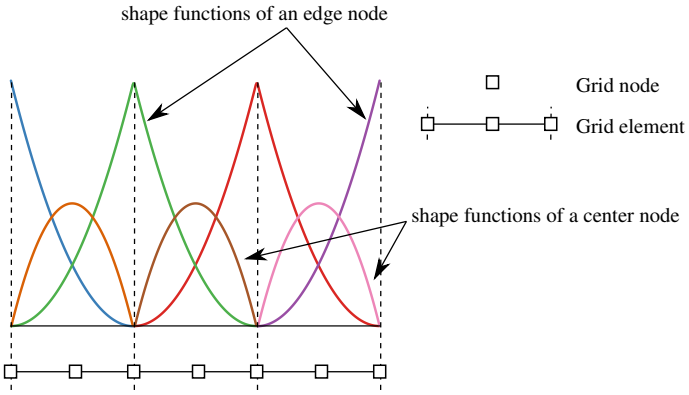


Fig. 3.12 Bernstein quadratic shape functions for a series of three elements in 1D. Note that Bernstein functions are still C^0

3.6 Convected Particle Domain Interpolation

The first method that can fully track particle domains is the Convected Particle Domain Interpolation (CPDI), developed in Sadeghirad et al. (2011) where 2D particle domains are approximated as parallelograms which still induce some gaps. Later on, quadrilateral particle domain was presented in Sadeghirad et al. (2013). Nguyen et al. (2017) extended CPDI to triangular particle domains, tetrahedral domains, and also to arbitrary polygon/polyhedral domains. All these formulations are presented in this section. We refer to Sect. 3.7.4 for an interpretation of CPDI as a way of projecting quantities defined over a Lagrangian mesh to a Cartesian grid.

3.6.1 One Dimensional Linear CPDI (CPDI-L2)

In order to have a better understanding of the CPDI shape functions, we herein derive the one dimensional CPDI shape functions. Visualization of 1D CPDI functions were provided in the original reference (Sadeghirad et al. 2011) but without details. If the 1D particle domain is represented as a two-node line element, we have

$$\begin{aligned} \phi_I(x_p) &= 0.5N_I(x_p^1) + 0.5N_I(x_p^2) \\ d\phi_I(x_p) &= \frac{-1}{l_p}N_I(x_p^1) + \frac{1}{l_p}N_I(x_p^2) \end{aligned} \tag{3.27}$$

where x_p^1, x_p^2 are the corners of the particle domain i.e., the nodes of the line element.

For a visualization of these functions, we consider a grid of three cells with four nodes as shown in Fig. 3.13. There is one particle with $l_p = 1 = h_x$ that moves from

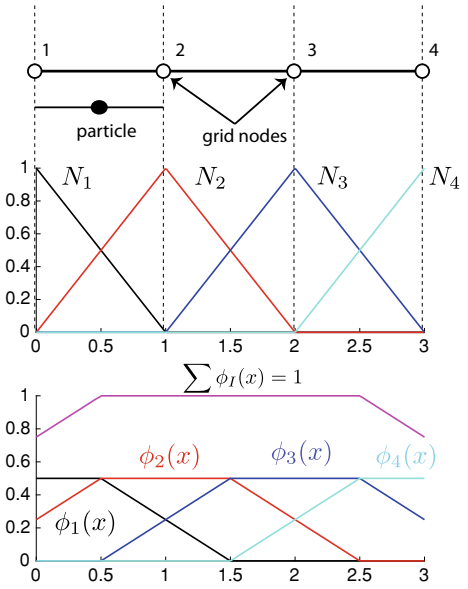


Fig. 3.13 One dimensional CPDI-L2 shape functions (bottom figure). Also shown are the standard FE basis functions (middle figure) and the grid/particle (top figure)

the left (node 1) to the right (node 4). By using Eq. 3.27, one can compute the CPDI basis functions of all four nodes as plotted in Fig. 3.13 (bottom figure). Also depicted is the standard FE shape functions–the well known hat functions (middle figure). As can be seen from the figure, the CPDI basis functions form a partition of unity, but they are not interpolator i.e., they do not satisfy the Kronecker property. It should be noted that as we did not use extra cells the CPDI functions do not form a PU for intervals $0 \leq x \leq 0.5$ and $2.5 \leq x \leq 3$.

3.6.2 Convected Particle Domain Interpolation (CPDI-R4)

In the first version of the CPDI family, the particle domain is tracked using the particle deformation gradient $\mathbf{F}$ in the way that the deformed particle domain is a parallelogram as shown in Fig. 3.14. The domain is defined by (i) the particle position and (ii) the domain vectors. The latter at time $t + \Delta t$ are given by

$$\begin{aligned} \mathbf{r}_1^{t+\Delta t} &= \mathbf{F}^{t+\Delta t}\mathbf{r}_1^0 \\ \mathbf{r}_2^{t+\Delta t} &= \mathbf{F}^{t+\Delta t}\mathbf{r}_2^0 \end{aligned} \tag{3.28}$$

where $\mathbf{r}_1^0$ and $\mathbf{r}_2^0$ are the initial domain vectors. Since the initial particle domain is a rectangle (that is why we label it as CPDI-R4 where R stands for rectangles and 4 is the number of nodes of one particle domain), then we have

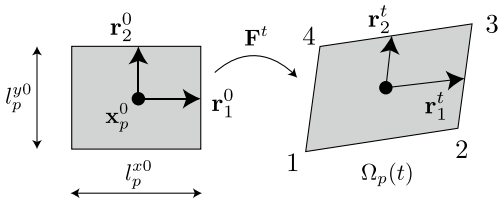


Fig. 3.14 Particle domain as a parallelogram in CPDI is defined by the particle position and the domain vectors

$$\mathbf{r}_1^0 = \begin{bmatrix} 0.5l_p^{x0} \\ 0 \end{bmatrix}, \quad \mathbf{r}_2^0 = \begin{bmatrix} 0 \\ 0.5l_p^{y0} \end{bmatrix} \tag{3.29}$$

where l_p^{x0} and l_p^{y0} are the initial particle sizes in the x and y direction, respectively.

The main issue in any GIMP methods is how to perform the integral in Eq. 3.10 effectively. Particularly when one allows the deformed particle domain to be of arbitrary shape and thus located arbitrarily with respect to the background grid. In CPDI, this is achieved by approximating the grid hat functions $N_I(\mathbf{x})$ over the deformed particle domain Ω_p using yet another basis functions

$$N_I(\mathbf{x}) \approx N_I^{\text{app}}(\mathbf{x}) = \sum_{c=1}^{4} M_c(\mathbf{x}) N_I(\mathbf{x}_c) \tag{3.30}$$

where $N_I(\mathbf{x}_c)$ are the conventional grid functions evaluated at the corner c of the particle domain and $M_c(\mathbf{x})$, or precisely $M_c(\xi, \eta)$ where (ξ, η) are the so-called parent coordinates, are the basis functions of the four-node quadrilateral elements (Q4). The original (N_I) and alternative basis functions ($N_I^{\text{app}}(\mathbf{x})$) differ from each other in the interior of the particle domain. And yet, the alternative basis function identically equals the exact basis function at the particle corners and hence on the particle edges since $M_c(\mathbf{x})$ are interpolation functions. This property makes the CPDI evaluation of nodal internal forces exact in 1D (Sadeghirad et al. 2011).

The GIMP basis functions in Eq. 3.10 now becomes

$$\begin{aligned} \phi_{Ip} &= \frac{1}{V_p} \int_{\Omega_p} N_I^{\text{app}}(\mathbf{x}) \mathrm{d}\Omega = \frac{1}{V_p} \int_{\Omega_p} \left[\sum_{c=1}^{4} M_c(\mathbf{x}) N_I(\mathbf{x}_c) \right] \mathrm{d}\Omega \\ &= \frac{1}{V_p} \sum_{c=1}^{4} \left[\int_{\Omega_p} M_c(\mathbf{x}) \mathrm{d}\Omega \right] N_I(\mathbf{x}_c) \end{aligned} \tag{3.31}$$

and similarly the gradient $\nabla \phi_{Ip}$ is written by

$$\begin{aligned}\nabla\phi_{Ip} &= \frac{1}{V_p}\int_{\Omega_p}\nabla N_I^{\text{app}}(\mathbf{x})\mathrm{d}\Omega = \frac{1}{V_p}\int_{\Omega_p}\left[\sum_{c=1}^{4}\nabla M_c(\mathbf{x})N_I(\mathbf{x}_c)\right]\mathrm{d}\Omega \\ &= \frac{1}{V_p}\sum_{c=1}^{4}\left[\int_{\Omega_p}\nabla M_c(\mathbf{x})\mathrm{d}\Omega\right]N_I(\mathbf{x}_c)\end{aligned} \tag{3.32}$$

Integrals in Eqs. (3.31) and (3.32) can be computed exactly and the resulting CPDI basis functions and first derivatives are written as (Sadeghirad et al. 2011)

$$\begin{aligned}\phi_{Ip} &= \frac{1}{4}\sum_{c=1}^{4}N_I(\mathbf{x}_c) \equiv \sum_{c=1}^{4}w_c^f N_I(\mathbf{x}_c), \quad w_c^f = 1/4 \\ \nabla\phi_{Ip} &= \frac{1}{V_p}\left\{(N_I(\mathbf{x}_1) - N_I(\mathbf{x}_3))\begin{bmatrix}r_{1y}-r_{2y}\\ r_{2x}-r_{1x}\end{bmatrix} + (N_I(\mathbf{x}_2) - N_I(\mathbf{x}_4))\begin{bmatrix}r_{1y}+r_{2y}\\ -r_{1x}-r_{2x}\end{bmatrix}\right\} \\ &\equiv \sum_{c=1}^{4}\mathbf{w}_c^g N_I(\mathbf{x}_c)\end{aligned} \tag{3.33}$$

where (r_{1x}, r_{1y}) are the components of $\mathbf{r}_1$; w_c^f and $\mathbf{w}_c^g$ are the so-called function/gradient weights. As can be seen from Eq. 3.33, the basis function of node I evaluated at particle p is the sum of the conventional grid functions evaluated at the four corners of the particle domain. The gradient is the weighted sum of the conventional grid functions evaluated at the four corners of the particle domain. Note that the coefficient $1/V_p$ in the gradient is different from the original formula of Sadeghirad et al. (2011) $(1/(2V_p))$ because we adopted different domain vectors.

It can be observed that the function weights sum to unity and the gradient weights sum to zero. These properties are the consequence of the PU of the FE shape functions M_c. For example, one can write

$$\sum_c M_c(\mathbf{x}) = 1 \rightarrow \int_{\Omega_p}\sum_c M_c\mathrm{d}\Omega = V_p \rightarrow \sum_c\left(\frac{1}{V_p}\int_{\Omega_p}M_c\mathrm{d}\Omega\right) = 1 \tag{3.34}$$

This observation is useful for verifying the derivation of CPDI functions. Based on this, it can be straightforwardly shown that the CPDI functions satisfy the partition of unity. We refer to Fig. 3.13 for a demonstration of this property.

The position of the four corners (they are numbered counter clock wise as shown in Fig. 3.14) are computed from the particle position and the particle domain vectors

$$\begin{aligned}\mathbf{x}_1 &= \mathbf{x}_p - \mathbf{r}_1 - \mathbf{r}_2 \\ \mathbf{x}_2 &= \mathbf{x}_p + \mathbf{r}_1 - \mathbf{r}_2 \\ \mathbf{x}_3 &= \mathbf{x}_p + \mathbf{r}_1 + \mathbf{r}_2 \\ \mathbf{x}_4 &= \mathbf{x}_p - \mathbf{r}_1 + \mathbf{r}_2\end{aligned} \tag{3.35}$$

and the particle domain volume is

$$V_p = A_p = 4\,||\mathbf{r}_1 \times \mathbf{r}_2|| \tag{3.36}$$

3.6.3 *Quadrilateral Convected Particle Domain Interpolation (CPDI-Q4)*

As parallelograms cannot fill space without gaps, cf. Fig. 3.15 and thus Sadeghirad et al. (2013) presented an improved CPDI method where particles are represented as quadrilaterals in 2D, cf. Fig. 3.16. This enhancement was referred to as CPDI2 by the authors. Herein, we label it the CPDI-Q4 to reflect that a particle resembles a Q4 finite element. This minor revision removes overlaps or gaps between particle domains and it also provides flexibility in choosing particle domain shape in the initial configuration.

The CPDI-Q4 weighting function and its derivatives are written by Sadeghirad et al. (2013)

$$\begin{aligned}\phi_{Ip} = \frac{1}{24V_p}\big[&(6V_p - a - b)N_I(\mathbf{x}_1) + (6V_p - a + b)N_I(\mathbf{x}_2) \\ &+ (6V_p + a + b)N_I(\mathbf{x}_3) + (6V_p + a - b)N_I(\mathbf{x}_4)\big]\end{aligned} \tag{3.37}$$

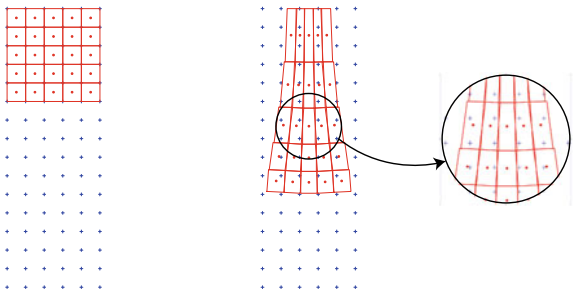


Fig. 3.15 Gaps in CPDI-R4. This is a compliant bar under influence of a large gravity force (de Vaucorbeil et al. 2020)

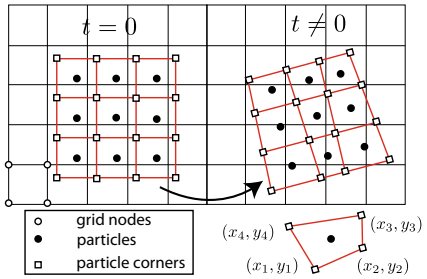


Fig. 3.16 Particle domains as (bilinear) quadrilaterals in CPDI-Q4. Note that the particle domain corners only play a role in defining the basis functions and they do not carry any material quantities (Nguyen et al. 2017)

$$\nabla\phi_{Ip} = \frac{1}{2V_p}\left\{N_I(\mathbf{x}_1)\begin{bmatrix} y_2 - y_4 \\ x_4 - x_2 \end{bmatrix} + N_I(\mathbf{x}_2)\begin{bmatrix} y_3 - y_1 \\ x_1 - x_3 \end{bmatrix} + N_I(\mathbf{x}_3)\begin{bmatrix} y_4 - y_2 \\ x_2 - x_4 \end{bmatrix} + N_I(\mathbf{x}_4)\begin{bmatrix} y_1 - y_3 \\ x_3 - x_1 \end{bmatrix}\right\} \tag{3.38}$$

where $a = (x_4 - x_1)(y_2 - y_3) - (x_2 - x_3)(y_4 - y_1)$ and $b = (x_3 - x_4)(y_1 - y_2) - (x_1 - x_2)(y_3 - y_4)$. The volume (actually area) of the particle domain is given by $V_p = 0.5[(x_1y_2 - x_2y_1) + (x_2y_3 - x_3y_2) + (x_3y_4 - x_4y_3) + (x_4y_1 - x_1y_4)]$. There was a typo is Sadeghirad et al. (2013) and the above equations are correct. Nguyen et al. (2017) provided a derivation and Sect. C.2 discusses another derivation using a computer algebra system.

The particle corners are updated using the updated grid velocities (as if we did before for the particles)

$$\mathbf{x}_c^{t+\Delta t} = \mathbf{x}_c^t + \Delta t \sum_I N_I(\mathbf{x}_c)\mathbf{v}_I^{t+\Delta t} \tag{3.39}$$

By using the grid velocities to update the particle corners, no gaps between particle domains will be produced. If needed, e.g. for particle visualization, the particle positions can be computed as the centers of the particle domains:

$$\mathbf{x}_o \equiv \frac{1}{V_p}\int_{\Omega_p} \mathbf{x}\mathrm{d}\Omega = \sum_c w_f^c \mathbf{x}_c \tag{3.40}$$

where in the second equality the mapping $\mathbf{x} = M_c\mathbf{x}_c$ was used.

3.6.4 *Triangular Convected Particle Domain Interpolation (CPDI-T3)*

It is straightforward to extend the CPDI to the case where the particle domains are linear (or three-node) triangles as illustrated in Fig. 3.17. This is beneficial for geometries where a discretization in terms of triangles is available. The CPDI-T3 weighting functions and first derivatives are given by Nguyen et al. (2017)

$$\phi_{Ip} = \frac{1}{3}\left[N_I(\mathbf{x}_1) + N_I(\mathbf{x}_2) + N_I(\mathbf{x}_3)\right] \tag{3.41}$$

$$\nabla\phi_{Ip} = \frac{1}{2V_p}\left\{N_I(\mathbf{x}_1)\begin{bmatrix} y_2 - y_3 \\ x_3 - x_2 \end{bmatrix} + N_I(\mathbf{x}_2)\begin{bmatrix} y_3 - y_1 \\ x_1 - x_3 \end{bmatrix} + N_I(\mathbf{x}_3)\begin{bmatrix} y_1 - y_2 \\ x_2 - x_1 \end{bmatrix}\right\} \tag{3.42}$$

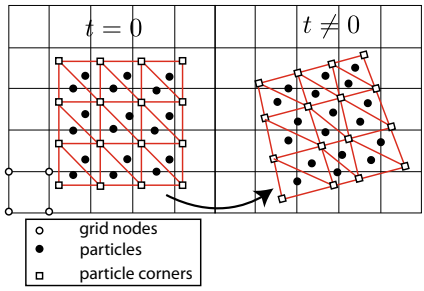


Fig. 3.17 Particle domains as (linear) triangles in CPDI-T3

3.6.5 *Three Dimensional Linear Tetrahedron CPDI (CPDI-Tet4)*

If the particles are represented by linear tetrahedron elements, the corresponding CPDI-Tet4 weighting and gradient weighting functions are given by Nguyen et al. (2017)

$$\begin{aligned}\phi_{Ip} &= \frac{1}{4}N_I(\mathbf{x}_1) + \frac{1}{4}N_I(\mathbf{x}_2) + \frac{1}{4}N_I(\mathbf{x}_3) + \frac{1}{4}N_I(\mathbf{x}_4)\\ \nabla\phi_{Ip} &= \frac{1}{6V_p}\left\{ N_I(\mathbf{x}_1)\begin{bmatrix}a_1\\b_1\\c_1\end{bmatrix} + N_I(\mathbf{x}_2)\begin{bmatrix}a_2\\b_2\\c_2\end{bmatrix} + N_I(\mathbf{x}_3)\begin{bmatrix}a_3\\b_3\\c_3\end{bmatrix} + N_I(\mathbf{x}_4)\begin{bmatrix}a_4\\b_4\\c_4\end{bmatrix}\right\}\end{aligned} \tag{3.43}$$

where

$$\begin{aligned}&a_1 = y_{42}z_{32} - y_{32}z_{42}, a_2 = y_{31}z_{43} - y_{34}z_{13}, a_3 = y_{24}z_{14} - y_{14}z_{24}, a_4 = y_{13}z_{21} - y_{12}z_{31}\\ &b_1 = x_{32}z_{42} - x_{42}z_{32}, b_2 = x_{43}z_{31} - x_{13}z_{34}, b_3 = x_{14}z_{24} - x_{24}z_{14}, b_4 = x_{21}z_{13} - x_{31}z_{12}\\ &c_1 = x_{42}y_{32} - x_{32}y_{42}, c_2 = x_{31}y_{43} - x_{34}y_{13}, c_3 = x_{24}y_{14} - x_{14}y_{24}, c_4 = x_{13}y_{21} - x_{12}y_{31}\end{aligned} \tag{3.44}$$

with $x_{ij} = x_i - x_j$ and $y_{ij} = y_i - y_j$; $6V_p = x_{21}(y_{23}z_{34} - y_{34}z_{23}) + x_{32}(y_{34}z_{12} - y_{12}z_{34}) + x_{43}(y_{12}z_{23} - y_{23}z_{12})$. Note that V_p is a signed quantity and a proper node numbering was used to have a positive value. This CPDI-Tet4 has been used in Sinaie et al. (2018) to model thin-walled metallic tubes under impacts and in Leavy et al. (2019) for mesoscale 3D simulations of polycrystalline materials.

3.6.6 *Polygonal and Polyhedral CPDI*

Voronoi diagrams or Voronoi tessellations have widespread applications in computational geometry, city planning, computer graphics, geophysics, and meteorology etc. It is easy to make a simple Voronoi diagram. Just throw a random scattering of points (or seeds) across a plane, connect these sites with lines (linking each point to

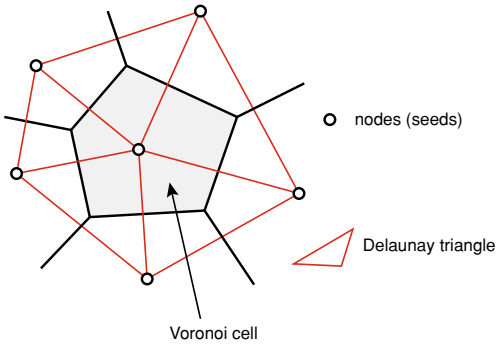


Fig. 3.18 Voronoi diagrams and its dual–the Delaunay triangles

those which are closest to it), and then bisect each of these lines with a perpendicular, cf. Fig. 3.18. Each cell in the diagram encloses a particular site, and the surface of the cell contains all the points on the plane that are closer to that site than to any other. The properties of Voronoi diagrams have been studied extensively and we refer the readers to the review paper (Aurenhammer 1991).

A centroidal Voronoi tessellation (CVT) is a special type of Voronoi diagrams. A Voronoi tessellation is called centroidal when the generating seed of each Voronoi cell is also its mean–the center of mass with respect to a given density function (Du et al. 1999). The center of mass is the arithmetic mean of all points weighted by the local density. If a physical object has uniform density, then its center of mass is the same as the centroid of its shape. It can be viewed as an optimal partition corresponding to an optimal distribution of generators.

Nguyen et al. (2017) extended CPDI to arbitrary polyhedron. For sake of simplicity, we present the 2D polygonal CPDI. The idea is simple: the particle polygon of n sides is partitioned into n triangles as shown in Fig. 3.19. This allows us to rewrite the function ϕ_{Ip} in Eq. 3.10:

$$\phi_{Ip} = \frac{1}{V_p}\sum_{s=1}^{n}\left[\int_{\Omega_p^s} N_I(\mathbf{x})\mathrm{d}\Omega\right] \approx \frac{1}{V_p}\sum_{s=1}^{n}\left[\int_{\Omega_p^s} N_I^{\text{app}}(\mathbf{x})\mathrm{d}\Omega\right] \approx \frac{1}{V_p}\sum_{s=1}^{n}\left[\sum_{c=1}^{3}\bar{w}_f^c N_I(\mathbf{x}_c)\right] \tag{3.45}$$

where Ω_p^s denotes the sub-triangles and $\bar{w}_f^c$ are the function weights defined previously for the CPDI-T3 case: $\bar{w}_f^c = A_s/3$, $c = 1, 2, 3$, and A_s is the area of sub-triangle s.

Equation 3.45 can be rewritten in the following general form which is equally applicable to any n-sided polygons with $n \geq 4$

$$\phi_{Ip} = \sum_{c=1}^{n+1} w_f^c N_I(\mathbf{x}_c) \tag{3.46}$$

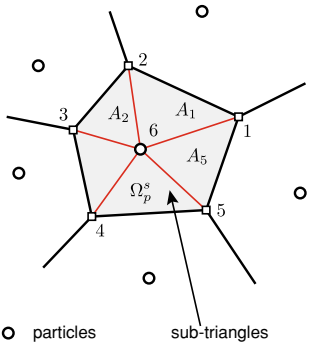


Fig. 3.19 Particle domains as a polygon in polygonal CPDI (Nguyen et al. 2017)

where for example $w_f^1 = (A_1/3 + A_2/3)/A$, and A denotes the area of the particle domain i.e., $A = \sum_s A_s$. Note that in our sub-sampling method in addition to the polygon vertices one needs to use the polygon's centroid as well.

In the same manner, the derivatives are given by

$$\nabla\phi_{Ip} \approx \frac{1}{V_p}\sum_{s=1}^{n}\left[\int_{\Omega_p^s}\nabla N_I^{\text{app}}(\mathbf{x})\mathrm{d}\Omega\right] \approx \frac{1}{V_p}\sum_{s=1}^{n}\left[\sum_{c=1}^{3}\bar{\mathbf{w}}_g^c N_I(\mathbf{x}_c)\right] \approx \sum_{c=1}^{n+1}\mathbf{w}_g^c N_I(\mathbf{x}_c) \tag{3.47}$$

where $\bar{\mathbf{w}}_g^c$ are the unnormalized gradient weights of the sub-triangle under consideration.

The proposed sub-sampling method is easy to be implemented and it is applicable to polygons of arbitrary sides. By numbering the particle corner nodes in a counter clockwise order, one simply loops over the polygon edges, for each edge a sub-triangle is formed and the function weights of this sub-triangle are computed and accumulated to the corresponding weights. The particles are stored as a finite element mesh consisting of elements of different types: quadrilaterals, pentagons, hexagons and heptagons etc.

Remark 30 The polyhedral CPDI could be the only MPM variant that represent the solid geometry most accurately (including the surfaces) and in the case that remeshing is needed (as the particle domains get distorted), no advection occurs. This is because the particles are the seeds for the Voronoi tesselation. If this is realized, the resulting method is quite similar to the PFEM. Yet this has not yet been implemented as remeshing is against the spirit of meshfree methods.

3.6.7 Complications in GIMP/CPDIs

There are some complications associated with GIMP and CPDI. First, for axisymmetric problems, the weighting and gradients must be modified (Nairn and Guilkey

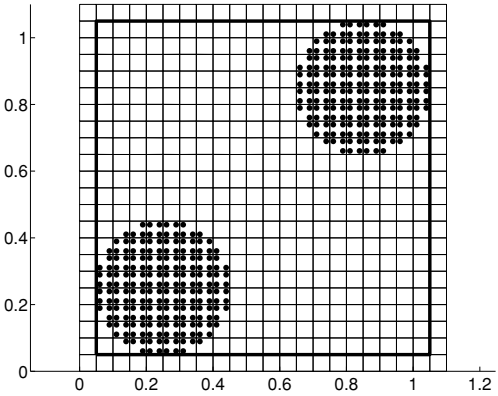


Fig. 3.20 Ghost cells in GIMP/CPDI: due to the larger extent of the GIMP basis functions' support, ghost cells have to be employed. Note that material points never move to ghost cells

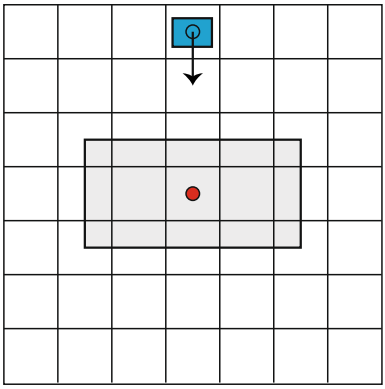


Fig. 3.21 Hole problem configuration: a background grid of 7×7 cells with two particles of which one is moving down with a constant velocity (de Vaucorbeil et al. 2020)

2015). Second, ghost cells have to be used and third, there are voids in CPDI. The latter two issues are discussed in this section.

High order C^1 GIMP shape functions and CPDIs requires the introduction of ghost cells (also referred to as extra cells) at the boundaries, cf. Fig. 3.20, so as that partition of unity (i.e., $\sum_I \phi_I(\mathbf{x}) = 1$) is satisfied. Without doing so would result in non-zero stresses for the particles on the bold lines even for rigid body motion. The use of ghost cells in GIMP is identical to extra cells in finite difference methods. However, these ghost cells induce a geometric error in order of h (Steffen 2009).

Next, we demonstrate holes can appear in the CPDIs.[2] The problem configuration is given in Fig. 3.21. The blue particle is moving down with a constant velocity whereas the red particle initial velocity is null. Note that we purposely assign the particle sizes for these two particles as follows. The red particle has a particle size in the horizontal direction *exceeds three grid spans* while the size of the blue particle is

[2] This was firstly discovered by Dr. Brannon at University of Uintah in a post on her blog.

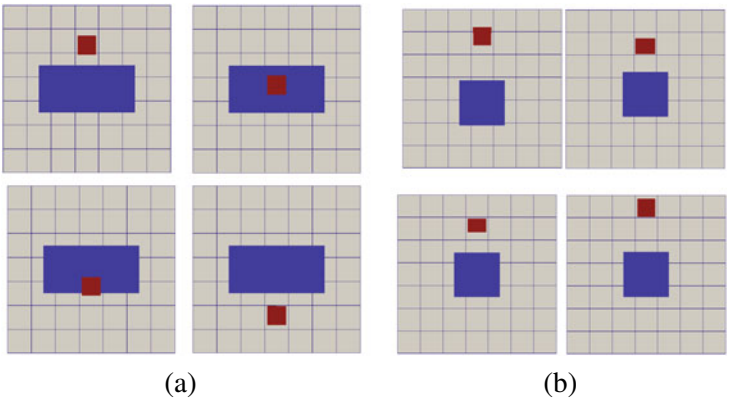


Fig. 3.22 Hole problem result: particle configurations at different time steps (de Vaucorbeil et al. 2020)

slightly smaller than the grid cell. Snapshots of the simulations are shown in Fig. 3.22a and it clearly demonstrates that CPDIs functions have holes which allow the blue particle goes through the red particle without any deformation. When the particle size is smaller than three grid spans (for the red particle) then no hole is created as demonstrated in Fig. 3.22b. This issue can be solved using particle splitting (Homel et al. 2016).

Parallelization complication of CPDI. Although the benefits of CPDI have been clearly demonstrated in a variety of test cases, its widespread use has been limited because of parallelization difficulties.

A common parallelization strategy for MPM codes is to use domain decomposition, in which the computational domain is decomposed into a number of sub-domains, often referred to as "patches"; each sub-domain in handled by different processing units. Particles near the boundary of a patch may contribute to, or receive data from, nodes on neighboring patches. For particles of a fixed dimension (in all MPMs except CPDI), defining these data dependencies is straightforward, and "ghost" data facilitate the calculation. We refer to Chap. 7 for a discussion of domain decomposition based parallelization of MPM codes.

The complication with CPDI is that particles may stretch unboundedly, leading to an unpredictable number of ghost nodes required to maintain the fidelity of the calculation.

Homel et al. (2016) has presented a solution to this problem. The idea is to scale the deformed particle domain so that the corner-to-corner distance of a single particle is less than the width of the ghost cell, if a single ghost row of cells is used. This is achieved by splitting the troubled particle into 2, 4, 8 particles in 1D, 2D and 3D, respectively.

3.7 The Generalized Particle in Cell Method

In the generalized particle in cell (GPIC) method, each solid is discretized by a number of finite elements, and all finite elements are embedded in a fixed background grid (Fig. 3.23). The background grid, for computational efficiency purposes, is a Cartesian grid with basis functions $\phi_I(\mathbf{x})$. The basis functions of the finite elements are denoted by $\phi_J^{\text{FE}}(\mathbf{x})$. So, we use subscript I for the nodes of the background grid and J for the nodes of the FE meshes.

At the nodes of the FE mesh(es), we store the mass, the internal forces, external forces due to body forces and external forces due to tractions. The internal variables e.g. damage/equivalent plastic strain, kinematic variables (e.g. deformation gradient tensor) and stresses are stored at the quadrature points of the FE meshes.

At the nodes of the Eulerian grid, we store the mass m_I, the momentum and the forces. They are obtained by projecting the mass, velocity and forces from the FE meshes in the MPM way. The equation of momentum is solved on this Eulerian grid.

We present the GPIC algorithms in Sect. 3.7.1 with details regarding the computation of the nodal mass and internal forces on the FE meshes in Sect. 3.7.2. For completeness, basis functions are presented in Sect. 3.7.3. The similarities of GPIC and CPDI are elucidated in Sect. 3.7.4 and an axi-symmetric formulation of GPIC is given in Sect. 3.7.5.

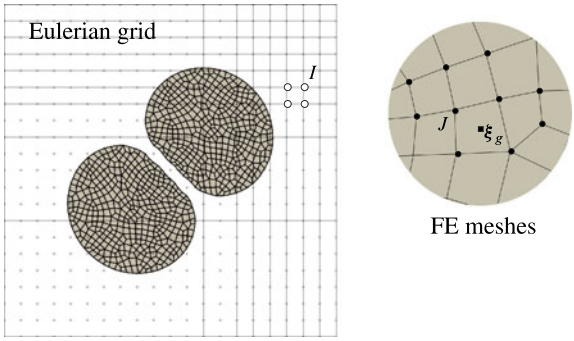


Fig. 3.23 GPIC: finite element meshes embedded in a background Eulerian grid (Nguyen et al. 2021)

3.7.1 General Algorithms

The flowchart of GPIC is as follows. At the beginning of the simulation, the masses of all FE nodes are calculated. The forces at these nodes are initialized to zeros. In the simulation loop, there are four steps. In step 1–meshes to grid (M2G)–one maps the FE quantities (mass, velocity and forces) to the Eulerian grid. In step 2, the momentum equation is solved on the Eulerian grid i.e., the grid velocities are updated. In step 3–grid to mesh (G2M)–the updated grid velocities are used to update the position, velocity and displacement of the FE nodes. And finally, in step 4–update mesh forces (UMF)–we compute the stresses at Gauss points and compute the internal/external forces at the FE nodes. For ease of implementation, a detailed flowchart is given in Algorithm 5 for the UL form of GPIC and Algorithm 6 for the TL form. For simplicity, these algorithms are presented only for one-point quadrature but extension to a general quadrature is straightforward. The TL form is different from the UL form only in the computation of the internal forces: the former employs the original configuration and the first Piola-Kirchoff stress and the latter adopts the current configuration and the Cauchy stress. We refer to Belytschko et al. (2000) for an excellent presentation of TL and UL formulations.

We use a subscript t for quantities at the beginning of a time step e.g. $\mathbf{x}_J^t$ and subscript $t + \Delta t$ for quantities at the end of the step. Note that the original configuration of the solid is designated by Ω_0 and the current configuration by Ω. The position of a FE node in Ω_0 is denoted by $\mathbf{X}$ and the position of that same node in Ω is $\mathbf{x}^t$. They are related via the displacement field $\mathbf{u}^t$: $\mathbf{x}^t = \mathbf{X} + \mathbf{u}^t$.

The mapping from FE meshes to the grid for the momenta, external and internal forces are achieved using the grid basis functions in the spirit of the MPM

$$\texttt{Mapping momenta}: \qquad (m\mathbf{v})_I^t = \sum_J \phi_I(\mathbf{x}_J^t)(m\mathbf{v})_J^t \tag{3.48a}$$

$$\texttt{Mapping external forces}: \qquad \mathbf{f}_I^{\text{ext},t} = \sum_J \phi_I(\mathbf{x}_J^t)\mathbf{f}_J^{\text{ext},t} \tag{3.48b}$$

$$\texttt{Mapping internal forces}: \qquad \mathbf{f}_I^{\text{int},t} = -\sum_J \phi_I(\mathbf{x}_J^t)\mathbf{f}_J^{\text{int},t} \tag{3.48c}$$

Note that gradients of the grid functions are never required. Therefore, GPIC is free of cell-crossing issue and highly efficient. In the graphics community, Hu et al. (2018) developed a moving least square MPM that also does not need basis gradients.

Algorithm 5 Solution procedure of explicit GPIC (UL): one-point quadrature.

Initialization
  Set up Eulerian grid and Lagrangian mesh
  Compute nodal mass $m_J = \sum_g \phi_J^{\mathrm{FE}}(\boldsymbol{\xi}_g)\rho_g w_g$
end
while $t < t_f$ **do**
  Reset grid quantities: $(m\mathbf{v})_I^t = \mathbf{0}, \mathbf{f}_I^{\mathrm{ext},t} = \mathbf{0}, \mathbf{f}_I^{\mathrm{int},t} = \mathbf{0}$
  Mapping from particles to nodes (M2G)
    Compute nodal momentum $(m\mathbf{v})_I^t = \sum_J \phi_I(\mathbf{x}_J^t)(m\mathbf{v})_J^t$
    Compute external force $\mathbf{f}_I^{\mathrm{ext},t} = \sum_J \phi_I(\mathbf{x}_J^t)\mathbf{f}_J^{\mathrm{ext},t}$
    Compute internal force $\mathbf{f}_I^{\mathrm{int},t} = -\sum_J \phi_I(\mathbf{x}_J^t)\mathbf{f}_J^{\mathrm{int},t}$
    Compute nodal force $\mathbf{f}_I^t = \mathbf{f}_I^{\mathrm{ext},t} + \mathbf{f}_I^{\mathrm{int},t}$
  end
  Update the momenta $(m\mathbf{v})_I^{t+\Delta t} = (m\mathbf{v})_I^t + \mathbf{f}_I^t \Delta t$
  Fix Dirichlet nodes I e.g. $(m\mathbf{v})_I^{t+\Delta t} = \mathbf{0}$ and $(m\mathbf{v})_I^t = \mathbf{0}$
  Update particle velocity, position & displacement (G2M)
    Get nodal velocities $\mathbf{v}_I^{t+\Delta t} = (m\mathbf{v})_I^{t+\Delta t}/m_I^t$
    Update mesh velocities $\mathbf{v}_J^{t+\Delta t} = \mathbf{v}_J^t + \sum_I \phi_I(\mathbf{x}_J^t)\left[\mathbf{v}_I^{t+\Delta t} - \mathbf{v}_I^t\right]$
    Update mesh positions $\mathbf{x}_J^{t+\Delta t} = \mathbf{x}_J^t + \Delta t \sum_I \phi_I(\mathbf{x}_J^t)\mathbf{v}_I^{t+\Delta t}$
    Update mesh incremental displacement $\mathbf{du}_J^{t+\Delta t} = \Delta t \sum_I \phi_I(\mathbf{x}_J^t)\mathbf{v}_I^{t+\Delta t}$
    Fix Dirichlet nodes K: $\mathbf{v}_K^{t+\Delta t} = \mathbf{0}, \mathbf{du}_K^{t+\Delta t} = \mathbf{0}, \mathbf{x}_K^{t+\Delta t} = \mathbf{X}_K$
  end
  Update stress and forces on the FE meshes (UMF)
    Update stress at element center $\boldsymbol{\sigma}(\boldsymbol{\xi}_0)$
    Compute internal force $\mathbf{f}_J^{\mathrm{int},t+\Delta t} = \boldsymbol{\sigma}(\boldsymbol{\xi}_0)\nabla\phi_J^{\mathrm{FE}}(\boldsymbol{\xi}_0)w(\boldsymbol{\xi}_0)$
    Compute external force $\mathbf{f}_J^{\mathrm{ext},t+\Delta t}$
  end
end while

Then, the momentum equation is solved and the grid velocity is updated in an exact way of the MPM. The faces of the Eulerian grid can play a role of rigid walls for which Dirichlet boundary conditions can be applied. In the G2M step, one projects the updated grid velocity to the FE meshes. For the velocity update, we follows the FLIP way of Brackbill and Ruppel (1986) by interpolating the grid velocity increment not the total grid velocity. This becomes the standard in the MPM because it avoids numerical dissipation which would occur if the total grid velocity was mapped to the mesh nodes. What is different from the MPM is that we also need to compute the displacement increments $\mathbf{du}_J$.

At this stage, we have the updated displacements at all nodes of the FE meshes, the last step is to update the internal forces $\mathbf{f}_J^{\mathrm{int},t+\Delta t}$ and external forces $\mathbf{f}_J^{\mathrm{ext},t+\Delta t}$ in a FEM manner. This is discussed in the next sub-section.

Algorithm 6 Solution procedure of explicit GPIC (TL): one-point quadrature.

Initialization
  Set up Eulerian grid and Lagrangian mesh
  Compute nodal mass $m_J = \sum_g \phi_J^{\mathrm{FE}}(\boldsymbol{\xi}_g)\rho_g w_g$
end
while $t < t_f$ **do**
  Reset grid quantities: $(m\mathbf{v})_I^t = \mathbf{0}, \mathbf{f}_I^{\mathrm{ext},t} = \mathbf{0}, \mathbf{f}_I^{\mathrm{int},t} = \mathbf{0}$
  Mapping from particles to nodes (M2G)
    Compute nodal momentum $(m\mathbf{v})_I^t = \sum_J \phi_I(\mathbf{x}_J^t)(m\mathbf{v})_J^t$
    Compute external force $\mathbf{f}_I^{\mathrm{ext},t} = \sum_J \phi_I(\mathbf{x}_J^t)\mathbf{f}_J^{\mathrm{ext},t}$
    Compute internal force $\mathbf{f}_I^{\mathrm{int},t} = -\sum_J \phi_I(\mathbf{x}_J^t)\mathbf{f}_J^{\mathrm{int},t}$
    Compute nodal force $\mathbf{f}_I^t = \mathbf{f}_I^{\mathrm{ext},t} + \mathbf{f}_I^{\mathrm{int},t}$
  end
  Update the momenta $(m\mathbf{v})_I^{t+\Delta t} = (m\mathbf{v})_I^t + \mathbf{f}_I^t \Delta t$
  Fix Dirichlet nodes I e.g. $(m\mathbf{v})_I^{t+\Delta t} = \mathbf{0}$ and $(m\mathbf{v})_I^t = \mathbf{0}$
  Update particle velocity, position & displacement (G2M)
    Get nodal velocities $\mathbf{v}_I^{t+\Delta t} = (m\mathbf{v})_I^{t+\Delta t}/m_I^t$
    Update mesh velocities $\mathbf{v}_J^{t+\Delta t} = \mathbf{v}_J^t + \sum_I \phi_I(\mathbf{x}_J^t)\left[\mathbf{v}_I^{t+\Delta t} - \mathbf{v}_I^t\right]$
    Update mesh positions $\mathbf{x}_J^{t+\Delta t} = \mathbf{x}_J^t + \Delta t \sum_I \phi_I(\mathbf{x}_J^t)\mathbf{v}_I^{t+\Delta t}$
    Update mesh incremental displacement $\mathbf{du}_J^{t+\Delta t} = \Delta t \sum_I \phi_I(\mathbf{x}_J^t)\mathbf{v}_I^{t+\Delta t}$
    Fix Dirichlet nodes K: $\mathbf{v}_K^{t+\Delta t} = \mathbf{0}, \mathbf{du}_K^{t+\Delta t} = \mathbf{0}, \mathbf{x}_K^{t+\Delta t} = \mathbf{X}_K$
  end
  Update stress and forces on the FE meshes (UMF)
    Update Cauchy stress at element center $\boldsymbol{\sigma}(\boldsymbol{\xi}_0)$
    Convert Cauchy stress to 1st PK stress $\mathbf{P}(\boldsymbol{\xi}_0) = J_F \boldsymbol{\sigma}(\boldsymbol{\xi}_0)(\mathbf{F}(\boldsymbol{\xi}_0))^{-\mathrm{T}}$
    Compute internal force $\mathbf{f}_J^{\mathrm{int},t} = \mathbf{P}(\boldsymbol{\xi}_0)\nabla_0\phi_J^{\mathrm{FE}}(\boldsymbol{\xi}_0)w(\boldsymbol{\xi}_0)$
    Compute external force $\mathbf{f}_J^{\mathrm{ext},t}$
  end
end while

Remark 31 The idea of computing the internal forces on a FE mesh and project it to an Eulerian grid was probably first presented by Lian et al. (2011); Hamad et al. (2015) where finite elements were used to model structural elements (reinforcement bars in Lian et al. (2011) and membranes in Hamad et al. (2015)). So, our method GPIC is an extension of their idea to solids. Interestingly, GPIC is very similar to CPDI of Sadeghirad et al. (2011). We present a comparison of CPDI and GPIC in Sect. 3.7.4.

3.7.2 Computation of Mass and Forces on FE Meshes

At the beginning of the simulation, we compute the nodal mass m_J for the FE meshes:

$$m_J = \int_{\Omega_0} \rho\phi_J^{\mathrm{FE}} \mathrm{d}\Omega = \sum_g \rho\phi_J^{\mathrm{FE}}(\boldsymbol{\xi}_g)w_g \tag{3.49}$$

where we have assumed that a lumped mass was adopted using the row sum technique and ρ is the material density. The second equation is the standard Gauss quadrature with Gauss points denoted by $\boldsymbol{\xi}_g$ and w_g is the weight. Unless otherwise stated, we use one-point quadrature with $\boldsymbol{\xi}_0$ denoting that point. In this case, we consider an element a particle.

The internal forces at node J are calculated as, using either an updated Lagrangian (UL) formulation or a total Lagrangian (TL) one:

$$\texttt{UL}: \qquad \mathbf{f}_J^{\text{int}} = \int_{\Omega} \boldsymbol{\sigma} \nabla \phi_J^{\text{FE}} \mathrm{d}\Omega = \sum_g \boldsymbol{\sigma}(\boldsymbol{\xi}_g) \nabla \phi_J^{\text{FE}}(\boldsymbol{\xi}_g) w_g \tag{3.50a}$$

$$\texttt{TL}: \qquad \mathbf{f}_J^{\text{int}} = \int_{\Omega_0} \mathbf{P} \nabla_0 \phi_J^{\text{FE}} \mathrm{d}\Omega = \sum_g \mathbf{P}(\boldsymbol{\xi}_g) \nabla_0 \phi_J^{\text{FE}}(\boldsymbol{\xi}_g) w_g \tag{3.50b}$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{P}$ is the first Piola-Kirchoff (1st PK) stress tensor. They are stored as 3×3 matrices for 3D problems. The gradient of the FE shape functions with respect to the current configuration and reference configuration are denoted by $\nabla \phi_J^{\text{FE}}$ and $\nabla_0 \phi_J^{\text{FE}}$, respectively. If the solid behaviour is described by a constitutive model using the Cauchy stress, one needs to convert it to the 1st PK stress in the TL formulation. For that conversion, J_F is the determinant of the gradient deformation tensor.

For a given FE element, we compute the deformation gradient tensor $\mathbf{F}$ at a Gauss point using its definition

$$\texttt{UL}: \qquad \mathbf{F} := \left(\mathbf{I} - \frac{\partial \mathbf{u}}{\partial \mathbf{x}} \right)^{-1} = \left(\mathbf{I} - \sum_J \nabla \phi_J^{\text{FE}}(\boldsymbol{\xi}_g)(\mathbf{x}_J^{t+\Delta t} - \mathbf{X}_J) \right)^{-1} \tag{3.51a}$$

$$\texttt{TL}: \qquad \mathbf{F} := \mathbf{I} + \frac{\partial \mathbf{u}}{\partial \mathbf{X}} = \mathbf{I} + \sum_J \nabla_0 \phi_J^{\text{FE}}(\boldsymbol{\xi}_g)(\mathbf{x}_J^{t+\Delta t} - \mathbf{X}_J) \tag{3.51b}$$

where $\mathbf{I}$ is the 3×3 identity matrix. And the loop over J is over the FE nodes of the element under consideration.

For elasto-plastic constitutive models, one needs the strain rate tensor $\mathbf{D}$. We present how to compute it for the TL formulation (as this one is robust for massive deformation whereas the UL is not). All these kinematic quantities are evaluated at the quadrature points of all elements. For ease of presentation, we did not specify this i.e., we write $\mathbf{L}$ for $\mathbf{L}(\boldsymbol{\xi}_g)$. First, one compute the rate of the deformation gradient $\dot{\mathbf{F}}$ as follows

$$\dot{\mathbf{F}} = \frac{1}{\Delta t} \sum_J \nabla_0 \phi_J^{\text{FE}}(\boldsymbol{\xi}_0) \mathbf{du}_J \tag{3.52}$$

Then, we compute the velocity gradient $\mathbf{L}$ by Belytschko et al. (2000)

$$\mathbf{L} = \dot{\mathbf{F}} \mathbf{F}^{-1} \tag{3.53}$$

And finally, $\mathbf{D}$ is computed as

$$\mathbf{D} = \frac{1}{2}(\mathbf{L} + \mathbf{L}^{\mathrm{T}}) \tag{3.54}$$

From that, one can compute the strain increment $\Delta t\mathbf{D}$ and use it for updating the stress.

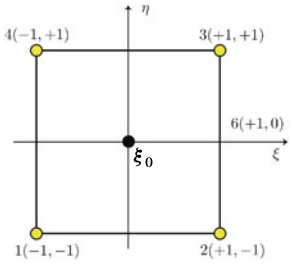
Our experiences show that the UL version of GPIC is not suitable for extremely large deformation problems. As the UL-GPIC is similar to the CPDI of Sadeghirad et al. (2011), this test just confirms the recent findings of Wang et al. (2019) that this method loses accuracy when the mesh becomes distorted. On the other hand, the TL-GPIC works well for massive tensile/compressive deformation. Owing to this, we recommend the use of the TL-GPIC.

3.7.3 *Finite Element Basis Functions*

As the FE basis functions and their derivatives are standard, we do not discuss them in detail here; for completeness, we present the four-node quadrilateral element in Fig. 3.24. For the TL formulation, one can compute $\nabla_0 \phi_J^{\mathrm{FE}}(\boldsymbol{\xi}_g)$ once in the initialization phase and store them for later use, which will significantly enhance the efficiency of GPIC.

To obtain the derivatives of the shape functions with respect to the spatial coordinates $\mathbf{x}$ we use the chain rule, say for a quadrilateral element

$$\begin{bmatrix} \dfrac{\partial \phi_J^{\mathrm{FE}}}{\partial \xi} \\ \dfrac{\partial \phi_J^{\mathrm{FE}}}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \phi_J^{\mathrm{FE}}}{\partial x}\dfrac{\partial x}{\partial \xi} + \dfrac{\partial \phi_J^{\mathrm{FE}}}{\partial y}\dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial \phi_J^{\mathrm{FE}}}{\partial x}\dfrac{\partial x}{\partial \eta} + \dfrac{\partial \phi_J^{\mathrm{FE}}}{\partial y}\dfrac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} \begin{bmatrix} \dfrac{\partial \phi_J^{\mathrm{FE}}}{\partial x} \\ \dfrac{\partial \phi_J^{\mathrm{FE}}}{\partial y} \end{bmatrix} \tag{3.55}$$



$$\phi_1^{\mathrm{FE}}(\xi,\eta) = \frac{1}{4}(1-\xi)(1-\eta)$$
$$\phi_2^{\mathrm{FE}}(\xi,\eta) = \frac{1}{4}(1+\xi)(1-\eta)$$
$$\phi_3^{\mathrm{FE}}(\xi,\eta) = \frac{1}{4}(1+\xi)(1+\eta)$$
$$\phi_4^{\mathrm{FE}}(\xi,\eta) = \frac{1}{4}(1-\xi)(1+\eta)$$

Fig. 3.24 The four-node quadrilateral element (Nguyen et al. 2021): the black filled circle denotes the Gauss point at the center with the weight equal 4. The final weight w_g used e.g. in Eq. 3.49 is thus 4 multiplied by the determinant of the transformation matrix in Eq. 3.57

Hence,

$$\begin{bmatrix} \dfrac{\partial \phi_J^{\mathrm{FE}}}{\partial x} \\ \dfrac{\partial \phi_J^{\mathrm{FE}}}{\partial y} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix}^{-1} \begin{bmatrix} \dfrac{\partial \phi_J^{\mathrm{FE}}}{\partial \xi} \\ \dfrac{\partial \phi_J^{\mathrm{FE}}}{\partial \eta} \end{bmatrix} \tag{3.56}$$

with

$$\begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \phi_1^{\mathrm{FE}}}{\partial \xi} & \dfrac{\partial \phi_2^{\mathrm{FE}}}{\partial \xi} & \dfrac{\partial \phi_3^{\mathrm{FE}}}{\partial \xi} & \dfrac{\partial \phi_4^{\mathrm{FE}}}{\partial \xi} \\ \dfrac{\partial \phi_1^{\mathrm{FE}}}{\partial \eta} & \dfrac{\partial \phi_2^{\mathrm{FE}}}{\partial \eta} & \dfrac{\partial \phi_3^{\mathrm{FE}}}{\partial \eta} & \dfrac{\partial \phi_4^{\mathrm{FE}}}{\partial \eta} \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix} \tag{3.57}$$

3.7.4 Equivalence Between CPDI and GPIC

We are demonstrating the equivalence between CPDI and GPIC method in two dimensions. Recall that the nodal internal force vector in the MPM is given by

$$\begin{aligned} f_{xI}^{\mathrm{int}} &= -V_p \left[(\sigma_{xx})_p \frac{\partial \phi_I}{\partial x}(\mathbf{x}_p) + (\sigma_{xy})_p \frac{\partial \phi_I}{\partial y}(\mathbf{x}_p) \right] \\ f_{yI}^{\mathrm{int}} &= -V_p \left[(\sigma_{xy})_p \frac{\partial \phi_I}{\partial x}(\mathbf{x}_p) + (\sigma_{yy})_p \frac{\partial \phi_I}{\partial y}(\mathbf{x}_p) \right] \end{aligned} \tag{3.58}$$

And the CPDI-Q4 derivatives are written as

$$\begin{bmatrix} \phi_{I,x}(\mathbf{x}_p) \\ \phi_{I,y}(\mathbf{x}_p) \end{bmatrix} = \frac{1}{2V_p} \left\{ N_I(\mathbf{x}_1) \begin{bmatrix} y_{24} \\ x_{42} \end{bmatrix} + N_I(\mathbf{x}_2) \begin{bmatrix} y_{31} \\ x_{13} \end{bmatrix} + N_I(\mathbf{x}_3) \begin{bmatrix} y_{42} \\ x_{24} \end{bmatrix} + N_I(\mathbf{x}_4) \begin{bmatrix} y_{13} \\ x_{31} \end{bmatrix} \right\} \tag{3.59}$$

Substitution of Eq. 3.59 into Eq. 3.58 results in the following explicit expression of the grid nodal internal force vector for the x component

$$\begin{aligned} f_{xI}^{\mathrm{int}} = {} & N_I(\mathbf{x}_1) \left(\frac{y_{42}}{2}\sigma_{xx} + \frac{x_{24}}{2}\sigma_{xy} \right) + N_I(\mathbf{x}_2) \left(\frac{y_{13}}{2}\sigma_{xx} + \frac{x_{31}}{2}\sigma_{xy} \right) \\ & + N_I(\mathbf{x}_3) \left(\frac{y_{24}}{2}\sigma_{xx} + \frac{x_{42}}{2}\sigma_{xy} \right) + N_I(\mathbf{x}_4) \left(\frac{y_{31}}{2}\sigma_{xx} + \frac{x_{13}}{2}\sigma_{xy} \right) \end{aligned} \tag{3.60}$$

We are going to prove that we can also get Eq. 3.60 by using GPIC. That is, we compute the internal force at the FE nodes analytically and map them to the Eulerian grid node I, and we get exactly Eq. 3.60.

The internal force vector at node J, $J = 1, 2, 3, 4$, of the particle element is given by

$$\mathbf{f}_J^{\text{int}} = -\int_{\Omega} \mathbf{B}_J^{\mathrm{T}} \boldsymbol{\sigma} \, \mathrm{d}\Omega = -\sum_{g=1} \mathbf{B}_{Jg}^{\mathrm{T}} \boldsymbol{\sigma}_g w_g |J|_g \tag{3.61}$$

where g denotes the integration points, w_g is the integration weight and $|J|$ is the determinant of the transformation. Assuming that the stress field within the particle element is uniform to be consistent with the CPDI, the internal force can be analytically evaluated using one single quadrature point positioned at the center of the element. The results are (only $x-$component of these forces)

$$\begin{aligned} f_{x1}^{\text{int}} &= \frac{y_{42}}{2}\sigma_{xx} + \frac{x_{24}}{2}\sigma_{xy} \\ f_{x2}^{\text{int}} &= \frac{y_{13}}{2}\sigma_{xx} + \frac{x_{31}}{2}\sigma_{xy} \\ f_{x3}^{\text{int}} &= \frac{y_{24}}{2}\sigma_{xx} + \frac{x_{42}}{2}\sigma_{xy} \\ f_{x4}^{\text{int}} &= \frac{y_{31}}{2}\sigma_{xx} + \frac{x_{13}}{2}\sigma_{xy} \end{aligned} \tag{3.62}$$

Now these internal forces at the nodes of the particle element are mapped to the background grid. At node I, we obtain

$$f_{xI}^{\text{int}} = N_I(\mathbf{x}_1) f_{x1}^{\text{int}} + N_I(\mathbf{x}_2) f_{x2}^{\text{int}} + N_I(\mathbf{x}_3) f_{x3}^{\text{int}} + N_I(\mathbf{x}_4) f_{x4}^{\text{int}} \tag{3.63}$$

which results in an internal force identical to the CPDI one given in Eq. 3.60.

So, what we have just demonstrated is that, for Eulerian grid with hat functions, CPDI-Q4 is equivalent to GPIC or vice versa. In the former, some parts are done analytically, in GPIC, everything is done numerically. While it is impossible to derive CPDI functions for other elements such as eight-node hexahedral elements and other quadratic elements, GPIC does not have this problem. Actually, one can use any elements (even isogeometric elements) on any background grid (structured grids with cubic B-splines or even unstructured grids).

3.7.5 Axi-Symmetric GPIC

Similar to the axi-symmetric formulation for the MPM, we compute the nodal mass (on the FE mesh) per radian:

$$m_J = \int_{\Omega_0} R\rho\phi_J^{\text{FE}} \, \mathrm{d}\Omega = \sum_g \rho R(\boldsymbol{\xi}_g)\phi_J^{\text{FE}}(\boldsymbol{\xi}_g) w_g \tag{3.64}$$

where $R(\boldsymbol{\xi}_g)$ is given by

$$R(\boldsymbol{\xi}_g) = \sum_J \phi_J^{\mathrm{FE}}(\boldsymbol{\xi}_g) R_J \tag{3.65}$$

and the node coordinates are denoted by (R_J, Z_J) in the reference configuration.

The nodal internal force vector of the FE mesh is written as

$$\begin{aligned} f_{rJ}^{\mathrm{int}} &= \sum_{g=1} R(\boldsymbol{\xi}_g) w_g \left[(P_{rr})_p \frac{\partial \phi_J^{\mathrm{FE}}}{\partial R}(\boldsymbol{\xi}_g) + (P_{rz})_p \frac{\partial \phi_J^{\mathrm{FE}}}{\partial Z}(\boldsymbol{\xi}_g) + (P_{\theta\theta})_p \frac{\phi_J^{\mathrm{FE}}(\boldsymbol{\xi}_g)}{R(\boldsymbol{\xi}_g)} \right] \\ f_{zJ}^{\mathrm{int}} &= \sum_{g=1} R(\boldsymbol{\xi}_g) w_g \left[(P_{rz})_p \frac{\partial \phi_J^{\mathrm{FE}}}{\partial R}(\boldsymbol{\xi}_g) + (P_{zz})_p \frac{\partial \phi_J^{\mathrm{FE}}}{\partial Z}(\boldsymbol{\xi}_g) \right] \end{aligned} \tag{3.66}$$

The rate of the deformation tensor $\dot{\mathbf{F}}$, now 3×3 matrix, is given by

$$\dot{\mathbf{F}} = \begin{bmatrix} \dot{F}_{rr} & \dot{F}_{rz} & 0 \\ \dot{F}_{zr} & \dot{F}_{zz} & 0 \\ 0 & 0 & \dot{F}_{\theta\theta} \end{bmatrix}, \quad \dot{F}_{\theta\theta} = \sum_J \frac{\phi_J^{\mathrm{FE}}(\boldsymbol{\xi}_g)}{R(\boldsymbol{\xi}_g)} du_{rJ} \tag{3.67}$$

And one does the same thing for $\mathbf{F}$ but $F_{\theta\theta}$ uses the total displacement not the displacement increment.

References

Alonso, E.E., Zabala, F.: Progressive failure of Aznalcóllar dam using the material point method. Géotechnique **61**(9), 795–808 (2011)

Andersen, S., Andersen, L.: Analysis of spatial interpolation in the material-point method. Comput. Struct. **88**(7–8), 506–518 (2010)

Andersen, S.M.: Material-point analysis of large-strain problems: modelling of landslides. Ph.D. thesis, Aalborg University (2009)

Aurenhammer, F.: Voronoi diagrams-a survey of a fundamental geometric data structure. ACM Comput. Surv. **23**(3), 345–405 (1991)

Bardenhagen, S.G., Kober, E.M.: The generalized interpolation material point method. Comput. Model. Eng. Sci. **5**(6), 477–495 (2004)

Belytschko, T., Liu, W.K., Moran, B.: Nonlinear Finite Elements for Continua and Structures. Wiley, Chichester, England (2000)

Beuth, L., Wieckowski, Z., Vermeer, P.A.: Solution of quasi-static large-strain problems by the material point method. Int. J. Numer. Anal. Meth. Geomech. **35**(13), 1451–1465 (2011)

Brackbill, J.U., Ruppel, H.M.: FLIP: a method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. J. Comput. Phys. **65**(2), 314–343 (1986)

de Vaucorbeil, A., Nguyen, V.P., Hutchinson, C.R.: A total-lagrangian material point method for solid mechanics problems involving large deformations. Comput. Methods Appl. Mech. Eng. **360**, 112783 (2020). https://doi.org/10.1016/j.cma.2019.112783

de Vaucorbeil, A., Nguyen, V.P., Sinaie, S., Wu, J.Y.: Chapter two—material point method after 25 years: theory, implementation, and applications. In: Advances in Applied Mechanics, vol. 53, pp. 185–398. Elsevier (2020)

Du, Q., Faber, V., Gunzburger, M.: Centroidal voronoi tessellations: applications and algorithms. SIAM Rev. **41**(4), 637–676 (1999)

Guilkey, James E., Hoying, James B., Weiss, Jeffrey A.: Computational modeling of multicellular constructs with the material point method. J. Biomech. **39**(11), 2074–2086 (2006)

Hamad, F., Stolle, D., Vermeer, P.: Modelling of membranes in the material point method with applications. Int. J. Numer. Anal. Meth. Geomech. **39**(8), 833–853 (2015)

Homel, M.A., Brannon, R.M., Guilkey, J.: Controlling the onset of numerical fracture in parallelized implementations of the material point method (MPM) with convective particle domain interpolation (CPDI) domain scaling. Int. J. Numer. Meth. Eng. **107**(1), 31–48 (2016)

Hughes, T.J.R.: The finite element method: linear static and dynamic finite element analysis. Dover Publications Inc., New York (2000). ISBN 0-486-41181-8. Corrected reprint of the 1987 original [Prentice-Hall Inc., Englewood Cliffs, N.J.]

Hughes, T.J.R., Cottrell, J.A., Bazilevs, Y.: Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. Comput. Methods Appl. Mech. Eng. **194**(39–41), 4135–4195 (2005)

Jassim, I., Stolle, D., Vermeer, P.: Two-phase dynamic analysis by material point method. Int. J. Numer. Anal. Meth. Geomech. **37**(15), 2502–2522 (2013)

Leavy, R.B., Guilkey, J.E., Phung, B.R., Spear, A.D., Brannon, R.M.: A convected-particle tetrahedron interpolation technique in the material-point method for the mesoscale modeling of ceramics. Comput. Mech. 1–21 (2019)

Lian, Y.P., Zhang, X., Zhou, X., Ma, Z.T.: A FEMP method and its application in modeling dynamic response of reinforced concrete subjected to impact loading. Comput. Methods Appl. Mech. Eng. **200**(17–20), 1659–1670 (2011)

Nairn, J.A., Guilkey, J.E.: Axisymmetric form of the generalized interpolation material point method. Int. J. Numer. Meth. Eng. **101**(2), 127–147 (2015)

Nguyen, V.P., de Vaucorbeil, A., Nguyen-Thanh, C., Mandal, T.K.: A generalized particle in cell method for explicit solid dynamics. Comput. Methods Appl. Mech. Eng. **360**, 112783 (2021). https://doi.org/10.1016/j.cma.2019.112783

Nguyen, V.P., Nguyen, C.T., Rabczuk, T., Natarajan, S.: On a family of convected particle domain interpolations in the material point method. Finite Elem. Anal. Des. **126**, 50–64 (2017)

Piegl, L.A., Tiller, W.: The NURBS Book. Springer (1996). ISBN 3540615458

Sadeghirad, A., Brannon, R.M., Burghardt, J.: A convected particle domain interpolation technique to extend applicability of the material point method for problems involving massive deformations. Int. J. Numer. Meth. Eng. **86**(12), 1435–1456 (2011)

Sadeghirad, A., Brannon, R.M., Guilkey, J.E.: Second-order convected particle domain interpolation (CPDI2) with enrichment for weak discontinuities at material interfaces. Int. J. Numer. Meth. Eng. **95**(11), 928–952 (2013)

Sinaie, S., Ngo, T.D., Nguyen, V.P., Rabczuk, T.: Validation of the material point method for the simulation of thin-walled tubes under lateral compression. Thin-Walled Struct. **130**, 32–46 (2018)

Steffen, M., Kirby, R.M., Berzins, M.: Analysis and reduction of quadrature errors in the material point method (MPM). Int. J. Numer. Meth. Eng. **76**(6), 922–948 (2008a)

Steffen, M., Wallstedt, P.C., Guilkey, J.E., Kirby, R.M., Berzins, M.: Examination and analysis of implementation choices within the material point method (MPM). Comput. Model. Eng. Sci. **31**(2), 107–127 (2008b)

Steffen, M.: Analysis-guided improvements of the Material Point Method (MPM). Ph.D. thesis, University of Utah (2009)

Stomakhin, A., Schroeder, C., Chai, L., Teran, J., Selle, A.: A material point method for snow simulation. ACM Trans. Graph. **32**(4), 1 (2013)

Sulsky, D., ong, M.: Improving the material-point method. In: Innovative Numerical Approaches for Multi-field and Multi-scale Problems, pp. 217–240. Springer, Berlin (2016)

Wang, L., Coombs, W.M., Augarde, C.E., Cortis, M., Charlton, T.J., Brown, M.J., Knappett, J., Brennan, A., Davidson, C., Richards, D., et al.: On the use of domain-based material point methods for problems involving large distortion. Comput. Methods Appl. Mech. Eng. **355**, 1003–1025 (2019)

Yuanming, Hu., Fang, Yu., Ge, Ziheng, Ziyin, Qu., Zhu, Yixin, Pradhana, Andre, Jiang, Chenfanfu: A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. ACM Trans. Graph. (TOG) **37**(4), 150 (2018)

Zhang, D.Z., Ma, X., Giguere, P.T.: Material point method enhanced by modified gradient of shape function. J. Comput. Phys. **230**(16), 6379–6398 (2011)

Chapter 4
Constitutive Models

This chapter presents some commonly used material models for solids. A model for isotropic linear elastic materials is given in Sect. 4.1. A nonlinear elastic model suitable for solids undergoing large elastic deformation such as rubbers and polymers is considered in Sect. 4.2. Section 4.3 presents the widely used Johnson-Cook elasto-plastic model in conjunction with the Mie-Grüneisen equation of state. We choose not to give a full exposition of the model and its surrounding theory but, instead, direct the reader to appropriate references where further details can be found. Rather, we provide stress update algorithms .

4.1 Linear Elastic Isotropic Material

For isotropic linear elastic materials, the stress tensor is given by

$$\sigma_{ij} = \lambda \epsilon_{kk} \delta_{ij} + 2\mu \epsilon_{ij}, \boldsymbol{\sigma} = (\lambda \text{tr} \boldsymbol{\epsilon}) \mathbf{I} + 2\mu \boldsymbol{\epsilon} \tag{4.1}$$

where λ and μ are the Lamé's constants. The stress tensor is also often written in terms of a hydrostatic and a deviatoric part

$$\boldsymbol{\sigma} = (K \text{tr} \boldsymbol{\epsilon}) \mathbf{I} + 2G \boldsymbol{\epsilon}' \tag{4.2}$$

where $\boldsymbol{\epsilon}'$ denotes the deviatoric strain tensor, and the bulk modulus K and shear modulus G are given by

$$K := \frac{3\lambda + 2\mu}{3}, \quad G := \mu \tag{4.3}$$

V. P. Nguyen et al., *The Material Point Method*, Scientific Computation,
https://doi.org/10.1007/978-3-031-24070-6_4

which are related to the Young's modulus E and Poisson's ratio ν by the following equation

$$E = \frac{9KG}{3K+G}, \quad \nu = \frac{3K-2G}{2(3K+G)} \tag{4.4}$$

The stress update is written as

$$\boldsymbol{\sigma}_p^{t+\Delta t} = \boldsymbol{\sigma}_p^t + (\lambda \mathrm{tr}\Delta\boldsymbol{\epsilon}_p)\mathbf{I} + 2\mu\Delta\boldsymbol{\epsilon}_p, \quad \Delta\boldsymbol{\epsilon}_p = \Delta t \mathbf{D}_p^{t+\Delta t}, \ \mathbf{D}_p^{t+\Delta t} = \frac{1}{2}(\mathbf{L}_p^{t+\Delta t} + (\mathbf{L}_p^{t+\Delta t})^{\mathrm{T}}) \tag{4.5}$$

4.2 Hyperelastic Solids

The Neo-Hookean model is an isotropic hyperelastic material model which exhibits characteristics that can be identified with the familiar material parameters found in linear elastic analysis Bonet and Wood (1997). For this material model, the 1st Piola-Kirchhoff stress tensor $\mathbf{P}$ is expressed as a function of the deformation matrix $\mathbf{F}$ and the Lamé constants μ and λ as follows:

$$\mathbf{P} = \mu(\mathbf{F} - \mathbf{F}^{-\mathrm{T}}) + \lambda \ln J \mathbf{F}^{-\mathrm{T}} \tag{4.6}$$

where $J = \det \mathbf{F}$.

Such formulation is convenient to be used with the total Lagrangian scheme, but for the updated Lagrangian scheme, the use of the Cauchy stress tensor $\boldsymbol{\sigma}$ is more appropriate. Since $\boldsymbol{\sigma} = J^{-1}\mathbf{P}\mathbf{F}^{\mathrm{T}}$, Eq. (4.6) becomes:

$$\boldsymbol{\sigma} = \frac{1}{J}\left[\mu(\mathbf{F}\mathbf{F}^{\mathrm{T}} - \mathbf{I}) + \lambda \ln J \mathbf{I}\right] \tag{4.7}$$

4.3 Elasto-Plastic Materials

Material modeling can be divided into three areas: volumetric response, or resistance to compressibility (equation of state), the resistance to distortion (constitutive); and the reduction in ability to carry stress as damage accumulates (failure). This section presents a temperature dependent hypoelastic-damage-plastic material model. The model is applicable to large strain and large rotation problems and suitable for problems when the elastic deformation is negligible compared with the plastic one.

To deal with the non-invariance of the stress rate under rigid body rotation Bonet and Wood (1997), the rigid body motion is eliminated from the strain rate, and thus from the stress rate. This is achieved by first performing a polar decomposing of $\mathbf{F}$ in rotation $\mathbf{R}$ and stretch $\mathbf{U}$ parts, *i.e.* $\mathbf{F} = \mathbf{R}\mathbf{U}$, using the singular value decomposition.

Then, the rigid body rotations are subtracted from the strain rate tensor $\mathbf{D}$ to obtain the un-rotated strain rate tensor $\mathbf{d} = \mathbf{R}^{\mathrm{T}}\mathbf{D}\mathbf{R}$. Once the un-rotated stress rate is integrated into the un-rotated stress $\boldsymbol{\sigma}'$ using Algorithm 8, the later is rotated back to the current configuration: $\boldsymbol{\sigma} = \mathbf{R}\boldsymbol{\sigma}'\mathbf{R}^{\mathrm{T}}$. Note that this is an alternative to objective stress rates presented in Eq. (2.15).

The Cauchy stress tensor $\boldsymbol{\sigma}$ is expressed as the sum of its isotropic part, i.e., the hydrostatic pressure ($\hat{p}$), and the traceless symmetric deviatoric stress $\boldsymbol{\sigma}^d$. An equation of state (EOS) is used to determine the hydrostatic pressure, see Sect. 4.3.1. The deviatoric response is determined using a plastic flow rule in combination with a yield condition. The von Mises yield condition is adopted here. Moreover, when fracture is taken into account, it is modeled using the classic continuum damage mechanics approach. That is, the stress tensor scales linearly with a damage variable D (Lemaitre 1985). In summary, the model equations are

$$
\begin{aligned}
&\boldsymbol{\sigma} = -\hat{p}\mathbf{I} + \boldsymbol{\sigma}^d && \text{(stress decomposition)}\\
&\hat{p} = \mathrm{EOS}(\rho, e, D, \cdots) && \text{(equation of state)}\\
&\mathbf{d}^d = \mathbf{d}^{d,e} + \mathbf{d}^{d,p} && \text{(strain rate decomposition)}\\
&\dot{\boldsymbol{\sigma}}^d = (1-D)2G(\mathbf{d}^d - \mathbf{d}^{d,p}) && \text{(isotropic hypoelastic)}\\
&f := \sigma_{\mathrm{eq}} - \sigma_f \le 0 && \text{(von Mises yield condition)}\\
&\mathbf{d}^{d,p} = \dot{\lambda}\tfrac{\partial f}{\partial \boldsymbol{\sigma}^d} && \text{(associated plastic flow)}\\
&f \le 0, \quad \dot{\lambda} \ge 0, \quad \dot{\lambda} f = 0 && \text{(Karush-Kuhn-Tucker conditions)}
\end{aligned}
\tag{4.8}
$$

where G is the shear modulus; λ is the plastic multiplier, $\mathbf{d}^d$ is the un-rotated deviatoric strain rate with $\mathbf{d}^{d,e}$ and $\mathbf{d}^{d,p}$ are the elastic and plastic parts, respectively; σ_f is the flow stress to be discussed in Sect. 4.3.2, and $\sigma_{\mathrm{eq}} = \sqrt{\frac{3}{2}\boldsymbol{\sigma}^d : \boldsymbol{\sigma}^d}$ is the equivalent von Mises stress.

4.3.1 Equation of State

The hydrostatic pressure is determined using the Mie-Grüneisen EOS modified to account for damage (Wilkins 1999):

$$
\begin{cases}
\hat{p} = \dfrac{\rho_0(1-D)c_0^2(\eta-1)\left[\eta - \dfrac{\Gamma_0}{2}(\eta-1)\right]}{[\eta - S_\alpha(\eta-1)]^2} + \Gamma_0 e; \quad \eta = \dfrac{\rho(1-D)}{\rho_0} \text{ if } \hat{p} > 0\\
\hat{p} = \dfrac{\rho_0 c_0^2(\eta-1)[\eta - \frac{\Gamma_0}{2}(\eta-1)]}{[\eta - S_\alpha(\eta-1)]^2} + \Gamma_0 e; \quad \eta = \dfrac{\rho}{\rho_0} \text{ otherwise}
\end{cases}
\tag{4.9}
$$

where c_0 is the bulk speed of sound, Γ_0 the Grüneisen Gamma in the reference state. S_α is the linear Hugoniot slope coefficient. Note that positive pressure is compression.

The internal energy e is written as

$$e = C_v \rho_0 (T - T_r) \tag{4.10}$$

where C_v denotes the specific heat at constant volume.

Linear equation of state which was modified to account for damage, designated by D, (Wilkins 1999):

$$\hat{p} = -K(1 - \det \mathbf{F})(1 - D) \tag{4.11}$$

where K is the bulk modulus.

4.3.2 Johnson-Cook Flow Model

According to the Johnson-Cook's flow stress model scaled with damage (Johnson and Cook 1985), the equivalent von Mises flow stress is written as

$$\sigma_f(\varepsilon_p, \dot{\varepsilon}_p, T) = \left[A + B\left(\varepsilon_p\right)^n\right]\left[1 + C \ln \dot{\varepsilon}_p^*\right]\left[1 - (T^*)^m\right](1 - D) \tag{4.12}$$

where ε_p is the equivalent plastic strain, ε_p^* is the normalized plastic strain rate, A the yield stress, B and n the strain hardening parameters, C the strain rate parameter, and m a temperature coefficient. This model has five experimentally determined parameters that describe quite well the response of a number of metals.

The normalized plastic strain rate and the homologous temperature T^* are given by:

$$\dot{\varepsilon}_p^* = \dot{\varepsilon}_p / \dot{\varepsilon}_0, \quad T^* = \frac{T - T_r}{T_m - T_r} \tag{4.13}$$

where $\dot{\varepsilon}_p$ and $\dot{\varepsilon}_0$ are the plastic strain rate, and the user-defined reference plastic strain rate, respectively; T_r denotes the reference temperature and T_m is the reference melting temperature. Unless otherwise stated, $\dot{\varepsilon}_0 = 1.0\ \mathrm{s}^{-1}$.

Remark 32 As can be seen from Eq. (4.12), the effect of plastic strain, its rate, temperature and damage are coupled by being multiplied by each other. In case that damage is not interested, its bracket is simply omitted. Similarly, if thermal softening is not needed, the corresponding bracket should be skipped. The temperature T can be computed solving a heat diffusion equation (see Sect. 10.3.2 for details) or it can be simply obtained from the plastic work. For high strain rate deformation, there is not sufficient time for heat conduction, and thus adiabatic condition prevails, the temperature increase can be computed as follows

$$\Delta T = \frac{\chi}{\rho C_p} \sigma_f \Delta \varepsilon_p \tag{4.14}$$

where $0 < \chi \leq 1$ is the Taylor-Quinney coefficient that determines how much the plastic work is converted into heat. For metals, $\chi = 0.9$ is often used.

4.3.3 Damage

The amount of damage (D) in each particle is determined using the Johnson-Cook damage model, widely used for engineering applications (Johnson and Cook 1985). It is a strain rate dependent phenomenological model based on the local accumulation of the plastic strain. According to this model, damage initiates when the accumulated equivalent plastic strain reaches the equivalent strain at failure ε_f (see Fig. 4.1):

$$D_{\text{init}} := \sum \frac{\Delta \varepsilon_p}{\varepsilon_f} = 1 \tag{4.15}$$

where $\Delta \varepsilon_p$ is the equivalent plastic strain increment. The equivalent strain at failure ε_f is given by Johnson-Cook's empirical equation:

$$\varepsilon_f = \left[D_1 + D_2 \exp(D_3 \sigma^*)\right] [1 + D_4 \ln(\dot{\varepsilon}_p^*)][1 + D_5 T^*] \tag{4.16}$$

and where D_1, …, D_5 are five material constants, $\sigma^* = -\hat{p}/\sigma_{\text{eq}}$ is the stress triaxiality.

As this model only describes damage initiation, in order to have a complete model of the fracture phenomenon, a damage evolution model is required. Here, it was assumed that the damage variable is given by:

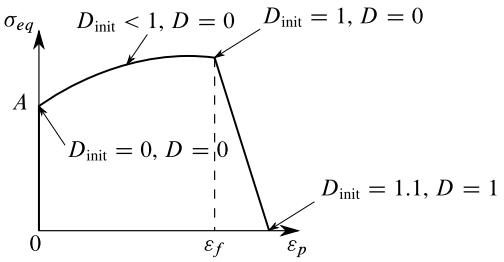


Fig. 4.1 Schematic of a typical equivalent stress-plastic strain curve showing the evolution of both the damage initiation variable D_{init} and the damage variable D. Three points are highlighted: the yield stress A, the point at which damage initiates $D_{\text{init}} = 1$, and the point of total failure $D = 1$ (de Vaucorbeil et al. 2020)

$$D = \begin{cases} 0 & \text{when } 0 \leq D_{\text{init}} < 1 \\ 10\,(D_{\text{init}} - 1) & \text{when } D_{\text{init}} \geq 1 \end{cases} \tag{4.17}$$

Even if this assumption affects the details of the damage propagation, it does not change the fundamentals of the implementation. Other forms for the damage evolution can be used.

It should be noted that this is a local damage model, which most often does not work well for damage modeling–the results are sensitive to the grid resolution. A nonlocal version of this is given later in Sect. 8.5.4.

4.3.4 Algorithm

The complete stress update algorithm is given in Algorithm 7. The equivalent plastic strain and the the deviatoric stress tensor are calculated jointly and incrementally according to the stress return algorithm developed by Leroch et al. (2016) and shown in Algorithm 8. The damage is updated using Algorithm 9. As can be seen, the damage is updated after the deviatoric stress update. This is mainly for efficiency and easy implementation. Furthermore, the flow stress is assumed to be constant during the stress update process.

Algorithm 7 Stress update algorithm.

1: Inputs: ε_p^t (equivalent plastic strain), $\boldsymbol{\sigma}_t'^{\,d}$ (un-rotated deviatoric stress), $\mathbf{L}$, $\mathbf{F}$ and damage D^t
2: Outputs: $\varepsilon_p^{t+\Delta t}$, $\boldsymbol{\sigma}_{t+\Delta t}$, new damage $D^{t+\Delta t}$
3: $\mathbf{D} = 0.5(\mathbf{L} + \mathbf{L}^{\mathrm{T}})$ ▷ strain rate
4: Polar decomposition for $\mathbf{F}$ to get $\mathbf{R}$ and $\mathbf{U}$
5: $\mathbf{d} = \mathbf{R}^{\mathrm{T}}\mathbf{D}\mathbf{R}$ ▷ un-rotated strain rate
6: $\mathbf{d}^d = \mathbf{d} - (1/3)\mathrm{tr}(\mathbf{d})\mathbf{I}$ ▷ un-rotated deviatoric strain rate
7: Compute $\boldsymbol{\sigma}'^{\,d}_{t+\Delta t}(\varepsilon_p^t, \mathbf{d}^d, D^t)$ ▷ Using Algorithm 11
8: Compute pressure $\hat{p}_{t+\Delta t}(D^t)$ ▷ using an EOS
9: Compute $\boldsymbol{\sigma}'_{t+\Delta t} = \boldsymbol{\sigma}'^{\,d}_{t+\Delta t} + p_{t+\Delta t}\mathbf{I}$ ▷ un-rotated stress
10: Compute $\boldsymbol{\sigma}_{t+\Delta t} = \mathbf{R}\boldsymbol{\sigma}'_{t+\Delta t}\mathbf{R}^{\mathrm{T}}$ ▷ final stress
11: Compute damage $D^{t+\Delta t}$ ▷ using Algorithm 12

For the TLMPM, this $\boldsymbol{\sigma}_{t+\Delta t}$ is converted to get $\mathbf{P}_{t+\Delta t}$, and to be used for the internal force computation in the next step. So, there is one time step lag between the stress and damage. However, this is not an issue due to the small time steps being used in explicit MPM.

Algorithm 8 Plasticity algorithm proposed by Leroch *et al.*

Inputs: ε_p^t (equivalent plastic strain), $\boldsymbol{\sigma}_t'^{\,d}$ (un-rotated deviatoric stress), $\mathbf{d}^d$, damage D^t
Outputs: $\varepsilon_p^{t+\Delta t}$ (equivalent plastic strain), $\boldsymbol{\sigma}_{t+\Delta t}'^{\,d}$
Compute $G' = (1 - D^t)G$
$\boldsymbol{\sigma}_{\text{trial}}'^{\,d} = \boldsymbol{\sigma}_t'^{\,d} + 2G'\Delta t\mathbf{d}^d$ ▷ purely elastic stress deviator update
$\sigma_{\text{trial}}'^{\,\text{eq}} = \sqrt{\frac{3}{2}\boldsymbol{\sigma}_{\text{trial}}'^{\,d} : \boldsymbol{\sigma}_{\text{trial}}'^{\,d}}$ ▷ equivalent von Mises trial stress
$\sigma_f = \left[A + B\left(\varepsilon_p^t\right)^n\right]\left[1 + C\ln\dot{\varepsilon}_p^*\right]\left[1 - (T^*)^m\right](1 - D^t)$ ▷ JC flow stress
if $\sigma_{\text{trial}}'^{\,\text{eq}} < \sigma_f$ **then** ▷ yielding did not occur, purely elastic step
    $\boldsymbol{\sigma}_{n+1}'^{\,d} = \boldsymbol{\sigma}_{\text{trial}}'^{\,d}$ ▷ keep trial deviatoric stress
else ▷ yielding has occurred
    $\Delta\varepsilon_p = (\sigma_{\text{trial}}'^{\,\text{eq}} - \sigma_f)/(3G')$ ▷ compute the equivalent plastic strain increment
    $\varepsilon_p^{t+\Delta t} = \varepsilon_p^t + \Delta\varepsilon_p$ ▷ update the undamaged matrix plastic strain
    $\boldsymbol{\sigma}_{t+\Delta t}'^{\,d} = \frac{\sigma_f}{\sigma_{\text{trial}}'^{\,\text{eq}}}\boldsymbol{\sigma}_{\text{trial}}'^{\,d}$ ▷ scale deviatoric stress back to yield surface
end if

Algorithm 9 Damage algorithm.

Inputs: $\Delta\varepsilon_p^{t+\Delta t}$ (incremental equivalent plastic strain), $\boldsymbol{\sigma}_{t+\Delta t}$, D_{init}^t (damage initiation variable)
Outputs: $D_{\text{init}}^{t+\Delta t}$ (updated damage initiation variable), $D^{t+\Delta t}$ (updated damage)
$\sigma^* = -hatp/\sigma_{\text{eq}}$ ▷ Compute stress triaxiality
$\varepsilon_f = \left[D_1 + D_2\exp(D_3\sigma^*)\right][1 + D_4\ln(\dot{\varepsilon}_p^*)][1 + D_5T^*]$ ▷ Strain at failure
$D_{\text{init}}^{t+\Delta t} = D_{\text{init}}^t + \Delta\varepsilon_p^{t+\Delta t}/\varepsilon_f$
if $D_{\text{init}}^{t+\Delta t} \geq 1$ **then** ▷ Damage has initiated
    $D_{t+\Delta t} = \left(D_{\text{init}}^{t+\Delta t} - 1\right)$
else ▷ Damage has not initiated
    $D_{t+\Delta t} = 0$
end if

References

Bonet, J., Wood, R.D.: Nonlinear Continuum Mechanics for Finite Element Analysis. Cambridge University Press (1997)

de Vaucorbeil, A., Nguyen, V.P., Hutchinson, C.R.: A Total-Lagrangian Material Point Method for solid mechanics problems involving large deformations. Comput. Methods Appl. Mech. Eng. **360**, 112783 (2020). https://doi.org/10.1016/j.cma.2019.112783

Johnson, G.R., Cook, W.H.: Fracture characteristics of three metals subjected to various strains, strain rates, temperatures and pressures. Eng. Fract. Mech. **21**(1), 31–48 (1985)

Lemaitre, J.: A continuous damage mechanics model for ductile fracture. J. Eng. Mater. Technol. **107**(1), 83–89 (1985). https://doi.org/10.1115/1.3225775

Leroch, S., Varga, M., Eder, S.J., Vernes, A., Rodriguez Ripoll, M., Ganzenmüller, G.: Smooth particle hydrodynamics simulation of damage induced by a spherical indenter scratching a viscoplastic material. Int. J. Solids Struct **81** (Supplement C), 188–202 (2016)

Mark, L.: Wilkins. Computer Simulation of Dynamic Phenomena. Springer, Berlin, Heidelberg (1999)

Chapter 5
Implementation

The MPM algorithms presented in Chap. 2 as well as the various weighting functions treated in Chap. 3 put us in a position nearly ready for coding. There are, nonetheless, some implementation details need to be discussed. First, particle generation is discussed in Sect. 5.1. Second, application of initial and boundary conditions are given in Sect. 5.2. Third, as CPDI's implementation is slightly different from other MPM variants, we provide implementation details of CPDI in Sect. 5.3. Fourth, material point methods adopting an unstructured grid, popular in the geo-technical engineering field, are briefly considered in Sect. 5.4. Finally, post-processing of the results of MPM simulations is discussed in Sect. 5.5.

5.1 Initial Particle Distribution

The MPM requires a grid and material points. As a Cartesian grid is easy to construct in any dimensions, we focus only on particle generation. In the FEM, the solid must be discretized into finite elements using a mesh generator. This meshing step is taking 80% of the total simulation time for complex geometries (Hughes et al. 2005). In the MPM, the solid is represented by a cloud of material points thus eliminating this time-consuming meshing step. However, the solid boundary has a zig-zag form except for CPDI or GPIC.

There are numerous ways to obtain the initial particle distribution which depends on the geometry of the object and/or the available tools. This section presents some algorithms to generate particles for solids of simple and complex geometries.

V. P. Nguyen et al., *The Material Point Method*, Scientific Computation,
https://doi.org/10.1007/978-3-031-24070-6_5

5.1.1 Regular Particle Distribution

One can distribute the particles in a regular pattern as shown in Fig. 3.20. The idea is to use a constant number of material points per grid cell and discard the points that fall outside the boundary of the initial material domain. For simple geometries, checking whether a point is outside a domain or not can be done analytically. For complex geometries, the level set method (Sethian 1999) can be used. This method of particle generation is the most commonly adopted one in MPM simulations. What constitutes a suitable number of particles is something of an open question, but it is typically advisable to use at least two particles in each cell in each direction, i.e. 4 particles per cell (PPC) in 2D and 8 PPC in 3D.

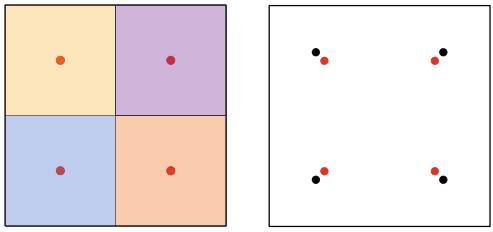
What should be the initial position of the particles relative to the grid cells? For the TLMPM, it is reasonable to place particles at the locations of the Gauss points that are optimal for numerical integration. For the ULMPM, there are different options. One can place the particles at Gauss points (this is coded in `Karamelo`), see Fig. 5.1b or regularly within the cells, see Fig. 5.1a. As the particles will, anyway, move out of their original locations, we do not see any difference between the two when a sufficient number of particles is used for each cell.

A fast particle generation algorithm. We present here some techniques to have a fast generation of particles for MPM simulations. For the sake of simplicity, only 2D is considered but the principles are general enough to be easily extended to 3D. If there are n grid cells and m geometrical objects, a naive algorithm by sweeping over the cells and for each cell loop over all the objects would result in an algorithm of order $\mathcal{O}(n \times m)$. This is inefficient if both n and m are large. First an integer cell coordinate i, j is introduced as shown in Fig. 5.2a. Note that the index is numbered from one. Let's assume that one needs to generate the particles for a triangle given in Fig. 5.2b. Based on the bounding box of the triangle and the cell indices, we can determine the cells surrounding the triangle. Finally, one loops only over those cells and distribute particles. A simple check whether a particle is within a polygon is used to discard particles outside the triangle.

The index of element with coordinates (i, j) is given by

$$e = i + \texttt{numx} \times (j - 1) \tag{5.1}$$

where `numx` denotes the number of cells along the x direction.



(a) Regular MPs (b) Gauss points

Fig. 5.1 Initial positions of material points. The black dots are particles placed at Gauss points (de Vaucorbeil et al. 2020)

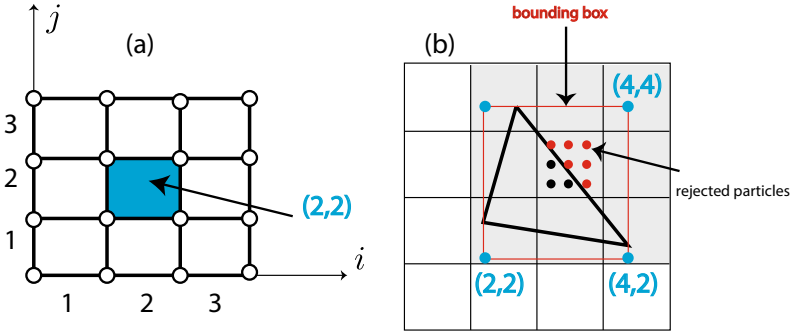


Fig. 5.2 Grid cells are indexed by (i, j) in the integer index space (**a**), and this is used to quickly identity cells which are closest to a given geometry object based on the bounding box concept (**b**). Instead of distributing particles in all cells (16) one only does this for 9 cells (de Vaucorbeil et al. 2020)

5.1.2 Irregular Particle Distribution

For solids of complex geometries, one can use a FE mesh generator to build a mesh and take the centers of the elements as particles. Actually, this is the technique often used with the uMPM (see Sect. 5.4). For example, in order to generate particles for a circular 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. 5.3. Alternatively, integration points of the triangular elements are mapped to the global coordinate system (using Sect. 2.25) 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. This kind of particle distribution is referred to as *irregular particle distribution*.

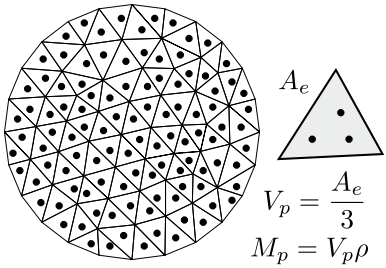


Fig. 5.3 Irregular initial particle distribution: obtained using the available FE mesh generators

5.1.3 *Particle Distribution from CAD*

This section presents a simple algorithm to create particiles directly from a CAD file. As an illustration, we confine to the STL format. This file format is supported by many other software packages; it is widely used for rapid prototyping, 3D printing and computer-aided manufacturing. An STL file describes a raw, unstructured triangulated surface by the unit normal and vertices (ordered by the right-hand rule) of the triangles using a three-dimensional Cartesian coordinate system.

The algorithm is quite simple. For each cell we distribute a certain number of particles; the problem is we have to classify each particle as being inside or outside the region specified by the STL file. To do so, we use a raytracing/winding number. Essentially, for a given particle, we just make a random ray from that particle in a random direction to infinity, and count the number of triangles it intersects with. If the number of intersections is odd, then the point is inside the region (or solid); if the number of intersections is even, then the point is outside the region (or solid).

The algorim is simple, but there is a problem: intersecting every ray with every single triangle takes forever, so to accelerate the ray tracing we use an octree.[1] Basically, we split the domain (which is one big cube) into eight cubes (split in half in each of the three directions), and then as necessary each cube is split into 8, etc. cursively. And at the leaf cubes, we record which triangles are contained within the cube. Then when we want to find all the triangles that intersect with a given ray, we intersect the ray with these nested bounding boxes first, recursively. If the ray does not intersect a bounding box, then we can instantly know that none of the triangles inside that bounding box intersect the ray. For instance, referring to Fig. 5.4, all the purple triangles no longer need to be tested for intersection with the ray.

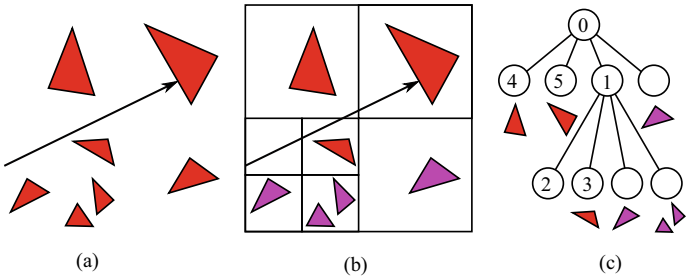
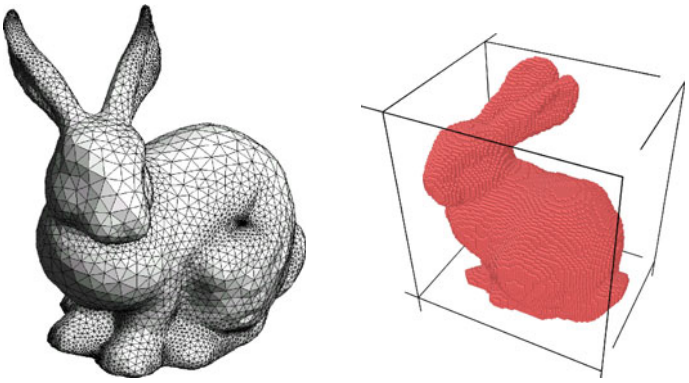


Fig. 5.4 The acceleration of raytracing is a common application of octrees: with a naive approach all triangles need to be tested for intersection with one ray (**a**). By the use of an octree, only the red triangles need to be tested for intersection (**b**). The numbers show the order of visited octree nodes in the hierarchy and the corresponding triangles (**c**). Adapted from Patzold (2016)

[1] An octree is a tree data structure in which each internal node has exactly eight children. Octrees are most often used to partition a three-dimensional space by recursively subdividing it into eight octants.



(a) STL triangles (b) 64 ppc

Fig. 5.5 Particle generation for the Stanford bunny obtained from a STL file. Visualized with `Ovito`

This algoritm was implemented in `Karamelo`–to be presented in Chap. 7–by Edward Buckland, a graduate from University of Melbourne. To test the algorithm, we consider the Stanford bunny and the resulting particle distributions are shown in Fig. 5.5.

5.1.4 Particle Distribution from Images

Geometries are often available as digital images (e.g. CT scan of the microstructure of the materials). In this case, the MPM is more suitable than the FEM for it allows a rather straightforward pre-processing step from the images to the numerical spatial discretization (Bardenhagen et al. 2005; Guilkey et al. 2006; Nairn 2007a). The basic idea is to convert each pixel to a material point locating at the center of the pixel and depending on the intensity of the pixel the particle is tagged to a specific material. The quality of this process depends heavily on the contrast of the image. Figure 5.6 depicts an example of converting an image which is a fiber-reinforced composite material to particles.

The Matlab code used to get Fig. 5.6 is given in Listing 5.1. The intensity of different phases can be easily obtained using the Matlab image processing toolbox **imtool**. Extension to three dimensional images is straightforward and thus not discussed here.

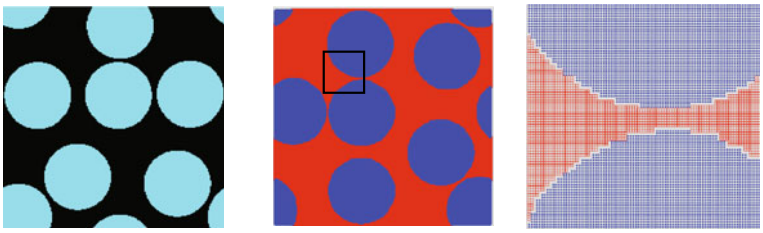


Fig. 5.6 Initial particle distribution: converted from an image. From left to right: RGB image, particles and zoom in (de Vaucorbeil et al. 2020)

Listing 5.1 From image to material points (2D two-phase images).

```
function pts=image2particles(file,lx,ly);
A       = imread(file);
grayIm = rgb2gray(A);
[noPixelX,noPixelY] = size(grayIm);
[dx,dy] = [lx/noPixelX ly/noPixelY];
pts1 = [];
pts2 = [];
for j=1:noPixelY
    for i=1:noPixelX
        intensity = grayIm(i,j);
        x  = (i-1)*dx + dx/2;
        y  = (j-1)*dy + dy/2;
        if      (intensity==0)
            pts1 = [pts1;x y];
        elseif (intensity==178)
            pts2 = [pts2;x y];
        end
    end
end
```

Remark 33 It should not be misunderstood that MPM should be the method of choice when the solid geometry is defined as images. There exists excellent tools to convert images into finite element meshes such as `Simpleware` (commercially available at http://www.simpleware.com) or `OOF` (freely available at http://www.ctcms.nist.gov/oof/oof2/). Nonetheless, for highly complex geometries, the automation of the mesh generation process is notoriously difficult and a significant portion of the analysis time is spent simply on mesh generation.

To demonstrate image-based simulations using the MPM, we use the MPM code `Uintah` to do a densification of foamed materials. Foamed materials find application in engineering systems on account of their unique structural properties. These properties include effective packaging, and energy absorption. Applications generally involve large material deformations. Foam mechanical properties are the result of the material's microstructure which is a complex three-dimensional network of

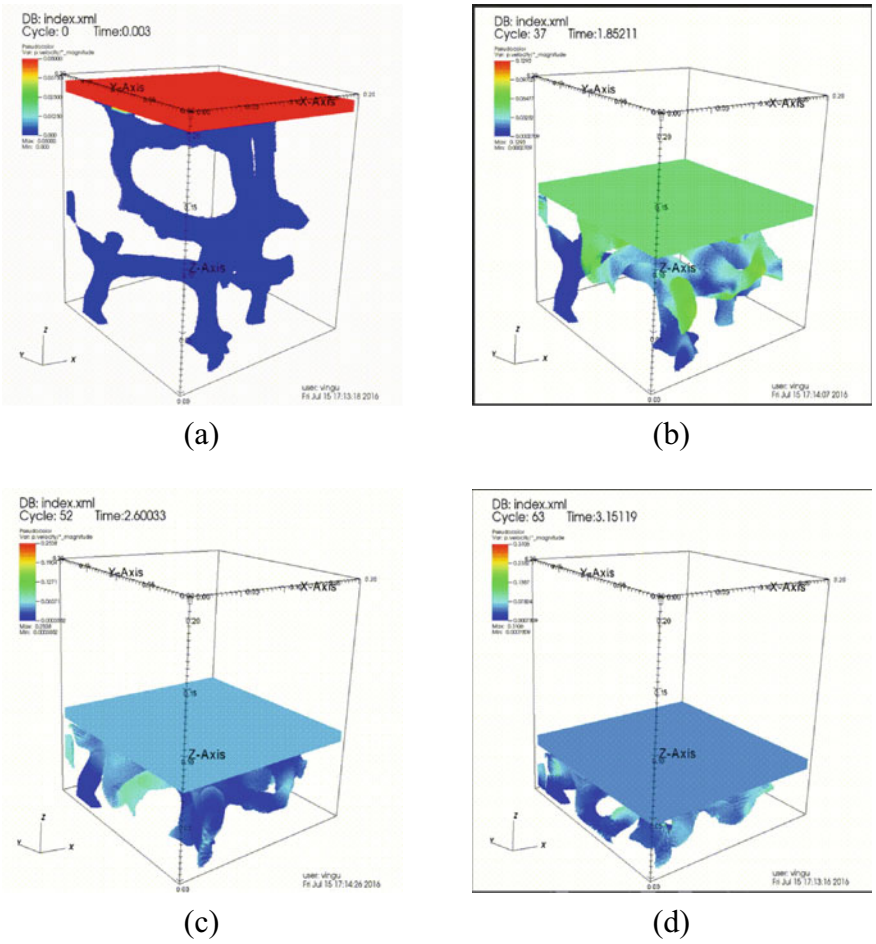


Fig. 5.7 Image-based simulation using the MPM: densification of foam using the MPM code `Uintah`. There many self contact events but they are only no-slip contacts. The foamed material is compressed by a rigid plate moving down with a constant velocity

struts, which undergo large deformations and self contact during deformation. Foam densification refers to the phase when the network is collapsed onto itself and contact between network elements results in a dramatic stiffening of the material. Computational modeling of the foam densification is challenging because we have to deal with (1) complex geometries and (2) many self contacts. The MPM provides a framework suitable for this problem as demonstrated in Bardenhagen et al. (2005). We reproduce some of their simulations in Fig. 5.7.

5.2 Initial and Boundary Conditions

Initial conditions such as initial velocities, temperatures and stresses (residual stresses or equilibrium stresses obtained in a static analysis prior to a dynamic simulation) are imposed on the particles, prior to the time loop starts.

On the other hand, it is quite hard to enforce Dirichlet and Neumann boundary conditions (BCs) in the MPM. Confining the discussion to mechanical problems, there exist Dirichlet BCs of the type $v_{iI} = \bar{v}$ on Γ_u–the so-called Dirichlet boundary, and Neumann BCs of the type $\mathbf{t} := \boldsymbol{\sigma} \cdot \mathbf{n} = \bar{\mathbf{t}}$ on the Neumann boundary Γ_t. In the case that $\bar{\mathbf{t}} = \mathbf{0}$, that is all boundaries are traction-free, one does not have to do anything related to Neumann BCs. Note that if no Dirichlet BC is applied to the boundary nodes, then particles are able to freely move out of the computational domain.

Enforcement of Dirichlet BCs is presented in Sect. 5.2.1 and of Neumann BCs in in Sect. 5.2.3. A technique tailored to CPDIs for enforcing Neumann BCs is given in Sect. 5.2.4. Finally, how Dirichlet and Neumann BCs are handled in GPIC is presented in Sect. 5.2.5.

5.2.1 Dirichlet Boundary Conditions

The Dirichlet boundary Γ_u is most of the case stationary. If Γ_u aligns with the grid, see Fig. 5.8a, it is straightforward to enforce BCs since the weighting functions satisfy the Kronecker delta property, at least at the boundaries (e.g. B-splines, GIMP). For explicit MPMs, basically one overwrites the calculated grid velocities (v_{iI}^t, $\tilde{v}_{iI}^{t+\Delta t}$ and $v_{iI}^{t+\Delta t}$) with the prescribed values ($\bar{v}$), for nodes I on Γ_u (solid black nodes in Fig. 5.8). For implicit MPMs, methods used in the FEM can be directly used (Hughes 2000).

When the Dirichlet boundary is inclined, see Fig. 5.8b, different options exist. The easiest option is to adopt an unstructured grid that conforms to the Dirichlet boundary as done by geo-technical engineers, see e.g. Wieçkowski (2004). If a Cartesian grid is being used, and for dynamics problems, the BCs can be enforced using rigid particles

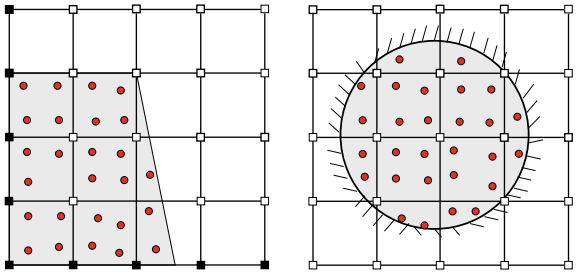


Fig. 5.8 Dirichlet boundary condition treatment in MPM: boundary aligns with the grid (left) and boundary not aligned with the grid (right). Black solid squares are the nodes where the Dirichlet BCs are enforced

as discussed in Sect. 8.1.6. For implicit MPM (implicit dynamics and quasi-static), it is much harder and there are some solutions (Remmerswaal 2017; Cortis et al. 2018; Bing et al. 2019; Liu and Sun 2019). Bing et al. (2019) presented a B-spline representation of 2D boundaries in the MPM.

Remark 34 Note that there exists problems where Γ_u is in motion. For example, in the field of geo-technical engineering, a zero pore pressure condition is applied on a moving soil surface. For free surface flows, one also needs to apply a pressure boundary condition (i.e., fluid pressure equals air pressure). It is, therefore, necessary to locate accurately a moving boundary. Remmerswaal (2017) studied many techniques commonly used in fluid mechanics such as VOF (Volume of Fluid), SMM (Surface Marker Method) and LSM (Level Set Method).

5.2.2 *Symmetric Boundary Conditions*

Symmetry BCs are used to represent a plane of symmetry, which allows the use of a reduced computational domain. Thanks to the background grid, it is straightforward to impose symmetry BCs. They are achieved by simply applying a zero velocity Dirichlet boundary condition on the component of velocity normal to a boundary, while allowing the other components to remain at their computed values.

In treating symmetric boundaries in GIMP or MPM with C^1 functions, special care must be taken for the ghost nodes. In particular, the normal component of velocity for these nodes is no longer set to zero, but rather should be set to the negative of the value of the nodes opposite the boundary, see Fig. 5.9.

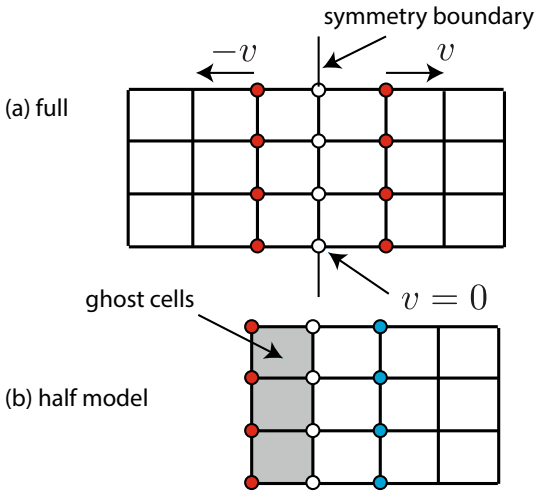


Fig. 5.9 Symmetric boundary conditions for ghost nodes in GIMP and in high order MPM (do not apply for B-splines with open knots): nodes on the symmetry boundary are fixed on the component of velocity normal to the boundary and the velocities of ghost nodes are set to negative of the velocities of the nodes opposite the boundary (blue nodes)

5.2.3 Neumann Boundary Conditions

Let us recall how the external force due to a traction is computed in the FEM. For simplicity, let's consider a 2D case. The Neumann boundary Γ_t is discretized by a set of 1D elements (they are actually the edges of the solid elements). The nodal force vector is then given by

$$\mathbf{f}_I^{\text{ext}} = \int_{\Gamma_t} N_I \bar{\mathbf{t}} \mathrm{d}\Gamma = \int_{-1}^{1} N_I(\xi) \bar{\mathbf{t}} J \mathrm{d}\xi \tag{5.2}$$

As an explicit representation of Γ_t is lacking, and there is no nodes on it as well, the MPM way of computing the force is as follows

$$\mathbf{f}_I^{\text{ext}} = \sum_{p=1}^{n_p} m_p \phi_I(\mathbf{x}_p) \bar{\mathbf{t}}^s(\mathbf{x}_p) h^{-1} = \sum_{p=1}^{n_p} A_p \phi_I(\mathbf{x}_p) \bar{\mathbf{t}}(\mathbf{x}_p) \tag{5.3}$$

where A_p represents the area of particle p. And the sum is over only boundary particles, see Fig. 5.10. The particle area can be updated using Nanson's formula, see e.g. Belytschko et al. (2000).

5.2.4 Neumann Boundary Conditions with CPDI

In this section computation of surface tractions is presented using procedures extensively used in the FEM. We illustrate the procedure with an example of a cylinder subjected to an inner pressure in Fig. 5.11. For simplicity the discussion is confined to two dimensions.

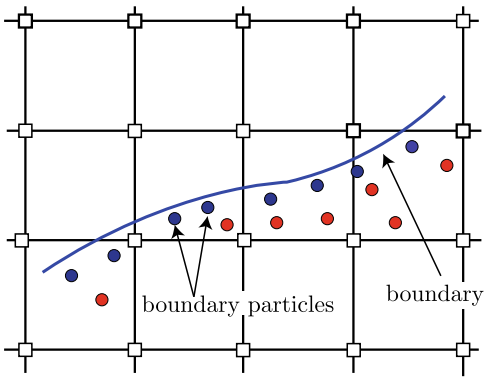


Fig. 5.10 Computation of external force due to a surface traction in the MPM. The traction is applied to the so-called boundary particles

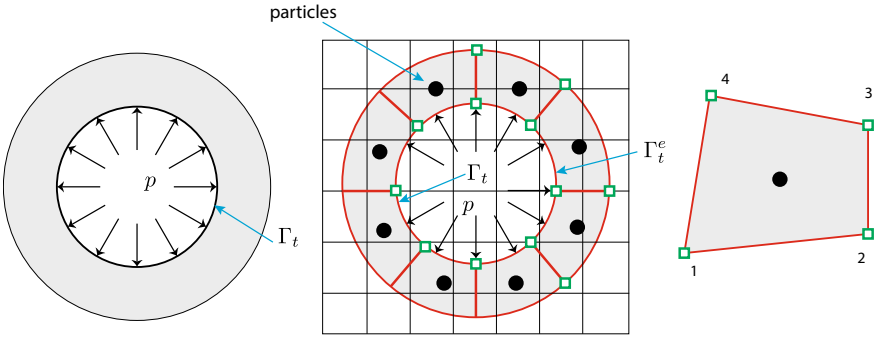


Fig. 5.11 Computation of surface traction in CPDI-MPM. A cylinder subjects to inner pressure (left) and CPDI-MPM (right)

Let us consider the case in which a portion of the traction is applied on edge 1 of a particle domain—edge connecting nodes 1 and 2—as shown in Fig. 5.11. The external force is thus given by

$$\begin{aligned}\mathbf{f}_I^{\text{ext}} &= \int_{\Gamma_t} N_I \bar{\mathbf{t}} \mathrm{d}\Gamma \\ &= \int_{\Gamma_t} \left(\sum_{c=1}^{4} M_c(\xi, -1) N_I(\mathbf{x}_c) \right) \bar{\mathbf{t}} \mathrm{d}\Gamma = \int_{-1}^{1} \left(\sum_{c=1}^{4} M_c(\xi, -1) N_I(\mathbf{x}_c) \right) \bar{\mathbf{t}} J \mathrm{d}\xi \end{aligned} \tag{5.4}$$

where in the second equality the idea of CPDI of approximating N_I was used; J is the Jacobian of the transformation that reads

$$J = \left|\left|\mathbf{x}'\right|\right| = \sqrt{x_{,\xi}^2 + y_\eta^2} = \frac{1}{2}\sqrt{x_{21}^2 + y_{21}^2} = 0.5l \tag{5.5}$$

which is constant and l is the length of edge 1. Furthermore, we assume that the traction is uniform over the particle edges. Thus Eq. (5.4) becomes

$$\mathbf{f}_I^{\text{ext}} = \sum_{c=1}^{4} \left(\int_{-1}^{1} M_c(\xi, -1) \mathrm{d}\xi \right) N_I(\mathbf{x}_c) \bar{\mathbf{t}} J = N_1(\mathbf{x}_1) \bar{\mathbf{t}} J + N_2(\mathbf{x}_2) \bar{\mathbf{t}} J \tag{5.6}$$

5.2.5 *Boundary Conditions in GPIC*

It is notoriously difficult to enforce Dirichlet and Neumann boundary conditions in a particle-based method. Even though there exists some options to enforce Dirichlet

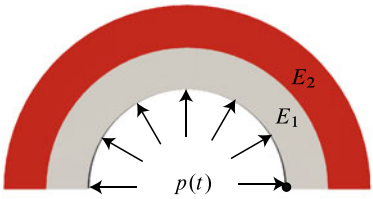


Fig. 5.12 A bi-material cylinder under internal pressure: the inner radius is 80 mm, the outer radius is 150 mm and the thickness is 10 mm. The pressure is $p(t) = p_0 \exp(-t/t_0)$ with $p_0 = 400$ MPa and $t_0 = 100$ μs. Only half of the cylinder is considered so that Dirichlet boundary conditions have to be enforced. The displacements of the marked node are monitored (Nguyen et al. 2021)

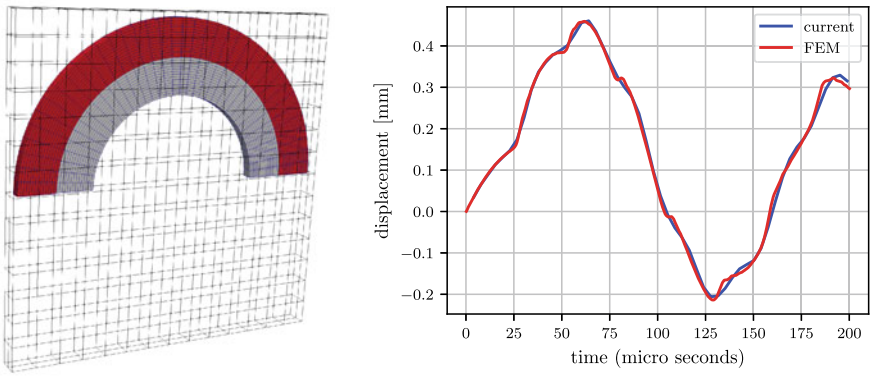
conditions, see e.g. Cortis et al. (2018), there is no good solution to Neumann conditions in the MPM, at least to the best of the authors' knowledge. Herein, we consider the problem of a bi-material (small strain) elastic cylinder subjected to an internal pressure. This simple problem demonstrates that GPIC can handle straightforwardly the enforcement of Dirichlet and Neumann boundary conditions, in the same manner of the FEM. Furthermore, it can model material interfaces without special treatment. Sadeghirad et al. (2013) showed that the MPM cannot capture weak discontinuities at material interfaces[2] with coarse discretization, and they presented an enrichment for the CPDI to model material interfaces.

We intentionally model half of the cylinder (Fig. 5.12) so as there are Dirichlet conditions to be applied (on the symmetrical surfaces). These Dirchilet BCs must be enforced on the solid mesh (not on the background grid); see line 20 in Algorithm 5. This is so because the internal forces are updated based on the information at the nodes of the solid mesh. The cylinder is meshed with eight-node hexahedral elements. The pressure is applied on the inner surface of the cylinder and the external forces due to this pressure are computed using the mesh of the inner surface (which consists of four-node quadrilateral elements) and a full quadrature. As this is entirely a FE business, we do not bother readers with details.

A coarse GPIC model is given in Fig. 5.13a. First, we consider the case where $E_1 = E_2 = 210$ GPa, $\rho = 7850$ kg/m^3 and $\nu = 0.3$. We have verified GPIC with an FEM solution (Figs. 5.13b, and 5.14). By considering $E_1 = E_2$ we can focus on the treatment of internal pressure in case errors occur. We move now to the case where $E_2 = 10E_1 = 2100$ GPa and the treatment of Neumann conditions and material interface of GPIC is verified (Fig. 5.15).

To conclude, for GPIC the treatment of both Dirichlet and Neumann boundary conditions is as easy as it is done in FEM.

[2] Weak discontinuities across material interfaces refer to a jump of the stresses/strains across a material interface.



(a) GPIC model (b) Verification with FEM

Fig. 5.13 Cylinder under internal pressure with $E_1 = E_2 = 210$ GPa: a coarse GPIC set-up (**a**) and quantitative verification with the FEM (**b**) (Nguyen et al. 2021)

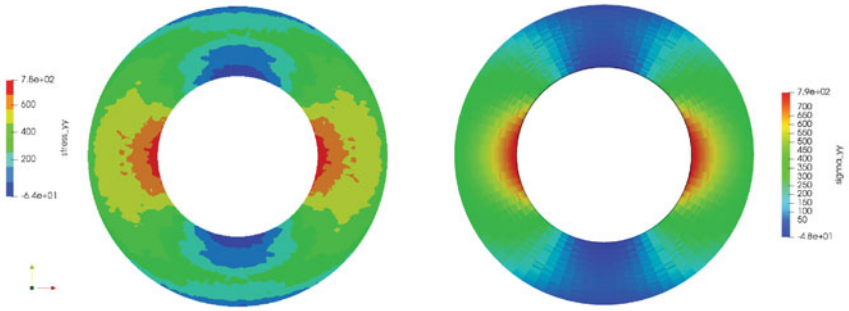
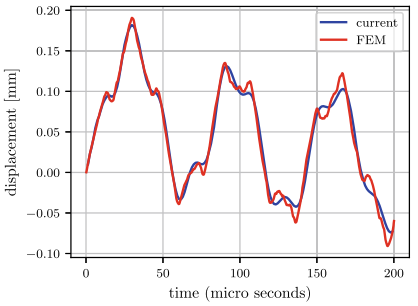


Fig. 5.14 Cylinder under internal pressure with $E_1 = E_2 = 210$ GPa: comparison of σ_{yy} obtained with the FEM (7700 four-node tetrahedral elements) and GPIC (2128 eight-node hexahedral elements for the cylinder and 12675 nodes for the Eulerian grid) (Nguyen et al. 2021)

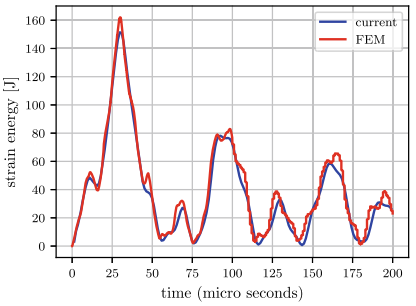
5.2.6 Rigid Bodies

Rigid bodies are used to model moving displacement boundary conditions as shown in Fig. 5.16. Rigid bodies are also represented by particles as deformable solids. However, there are some simplifications. Indeed, material properties are not required since these regions will not deform; their purpose is only to impose boundary conditions on deformable regions.

The positions and velocities of the rigid particles are simply given by (performed after updating the deformable particles)



(a) Displacement profile of the marked node



(b) Strain energy profile

Fig. 5.15 Cylinder under internal pressure with $E_2 = 10E_1 = 2100$ GPa: verification of GPIC with the FEM for displacement and strain energy profiles. The FEM model (Abaqus): about 33 thousands eight-node hexahedral elements. GPIC: similar to the case $E_1 = E_2$ (Nguyen et al. 2021)

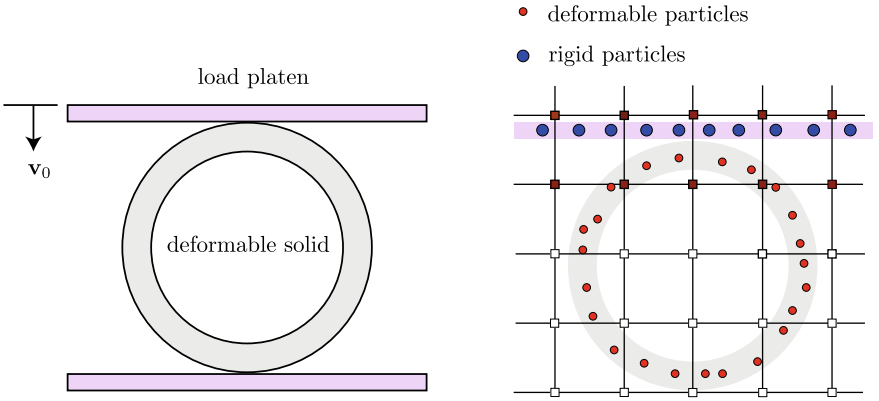


Fig. 5.16 Moving displacement boundary condition: a ring being smashed by a rigid platen (**a**) and rigid body to model the platen (**b**). Dark red squares represent nodes which are assigned the moving velocities (if the hat functions are used) (de Vaucorbeil et al. 2020)

$$\begin{aligned} \mathbf{v}_p^{\text{rigid},t+\Delta t} &= \mathbf{v}_0 \\ \mathbf{x}_p^{\text{rigid},t+\Delta t} &= \mathbf{x}_p^{\text{rigid},t} + \Delta t \mathbf{v}_0 \end{aligned} \tag{5.7}$$

The velocities of the rigid particles are transferred to the normal particles as follows. First, the elements which contain the rigid particles are determined. Second, all the nodes of these elements are marked (Fig. 5.16). These nodes are assigned the velocities of the rigid particles in case of a no slip contact between the rigid body and the deformable one. For frictional contacts, see Sect. 8.1.6. For implementation, we present the MUSL algorithm when rigid particles are present in Algorithm 10. It is worthy noting that this algorithm is simply a special case of the multi-material contact algorithm of Bardenhagen et al. (2000).

Algorithm 10 Solution procedure of explicit MPM using MUSL with rigid bodies.

while $t < t_f$ **do**
  Reset grid quantities: $m_I^t = 0$, $(m\mathbf{v})_I^t = \mathbf{0}$, $\mathbf{f}_I^{\text{ext},t} = \mathbf{0}$, $\mathbf{f}_I^{\text{int},t} = \mathbf{0}$
  Mapping from deformable particles to nodes (P2G)
  end
  Update the momenta $(m\tilde{\mathbf{v}})_I^{t+\Delta t} = (m\mathbf{v})_I^t + \mathbf{f}_I^t \Delta t$
  Fix Dirichlet nodes I: I e.g. $(m\mathbf{v})_I^t = \mathbf{0}$ and $(m\tilde{\mathbf{v}})_I^{t+\Delta t} = \mathbf{0}$
  Fix Dirichlet nodes II: J e.g. $\mathbf{v}_J^t = \mathbf{v}_0$ and $\tilde{\mathbf{v}}_J^{t+\Delta t} = \mathbf{v}_0$
  Update deformable particle velocities and grid velocities (double mapping)
    Get nodal velocities $\tilde{\mathbf{v}}_I^{t+\Delta t} = (m\tilde{\mathbf{v}})_I^{t+\Delta t}/m_I^t$
    Update particle positions $\mathbf{x}_p^{t+\Delta t} = \mathbf{x}_p^t + \Delta t \sum_I \phi_I(\mathbf{x}_p^t)\tilde{\mathbf{v}}_I^{t+\Delta t}$
    Update particle velocities $\mathbf{v}_p^{t+\Delta t} = \alpha\left(\mathbf{v}_p^t + \sum_I \phi_I(\mathbf{x}_p^t)\left[\tilde{\mathbf{v}}_I^{t+\Delta t} - \mathbf{v}_I^t\right]\right) + (1-\alpha)\sum_I \phi_I(\mathbf{x}_p^t)\tilde{\mathbf{v}}_I^{t+\Delta t}$
    Update grid velocities $(m\mathbf{v}_I)^{t+\Delta t} = \sum_p \phi_I(\mathbf{x}_p^t)(m\mathbf{v})_p^{t+\Delta t}$
    Fix Dirichlet nodes I: $(m\mathbf{v})_I^{t+\Delta t} = \mathbf{0}$
    Fix Dirichlet nodes II: $\mathbf{v}_J^{t+\Delta t} = \mathbf{v}_0$
  end
  Update deformable particles (G2P)
  end
  Update rigid particles (G2P)
    $\mathbf{x}_p^{\text{rigid},t+\Delta t} = \mathbf{x}_p^{\text{rigid},t} + \Delta t \mathbf{v}_0$
  end
end while

5.3 Implementation of CPDI

We therein only present the implementation of only CPDI-Q4 as once this is understood, the implementation of the other variants is straightforward. The algorithm to compute the CPDI shape functions and derivatives for a given particle is given in Algorithm 11 and Fig. 5.17. One extra step, compared to all other MPM variants, is to update the particle domain vectors (for CPDI-R4) or particle corners (for CPDI-Q4). This is done at the end of every time step.

Algorithm 11 Algorithm to evaluate CPDI shape functions/derivatives.

1: Compute the four function weights w_c^f;
2: Compute the four gradient weights $\mathbf{w}_c^g$;
3: For each corner, determine which element contains it;
4: Get the nodes of the four elements, I, that contain the four corners;
5: For each node of those, loop over the corner and compute $N_I(\mathbf{x}_c)$;
6: Use Eqs. (3.37) and (3.38) to compute ϕ_I and $\nabla\phi_I$.

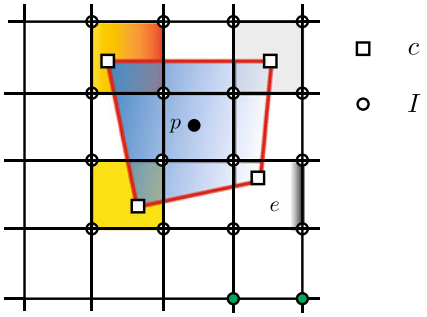


Fig. 5.17 Particle and node interaction in CPDI-Q4: white solid circles are those nodes with non-zero shape functions at the particle i.e., $\phi_{Ip} \neq 0$, green solid circles are those do not interact with particle p. Corners of the particle domain are denoted by squares. Shaded elements are those contain the particle corners

5.4 MPM Using an Unstructured Grid

Although computationally intensive, MPM formulations adopting an unstructured background grid is popular in the geo-technical engineering community (Wieçkowski et al. 1999; Wieçkowski 2004; Beuth et al. 2011; Jassim et al. 2013). One advantage is the ease with which boundary conditions can be enforced. Note, however, that this is only the case for fixed boundaries. Another motivation of adopting unstructured grids is to handle complex geometries (e.g. multiple soil layers). Herein we discuss aspects that are specific to MPM using unstructured meshes. First, shape functions of the MPM using unstructured grids, dubbed uMPM, are discussed in Sect. 5.4.1. Second, the problem of particle registration to the grid is treated in Sect. 5.4.2. Finally, mixed integration, as a means to reduce cell-crossing error in the quasi-static uMPM, is given in Sect. 5.4.3. A more efficient uMPM with C^1 shape functions is given in Sect. 5.4.4.

5.4.1 Shape Functions

As it is difficult to develop GIMP (and CPDI) and B-splines for an unstructured mesh, uMPM simply employs the well-known isoparametric finite elements in which the shape functions are written in terms of the so-called natural coordinates. Thus, there is a need to convert the particle position (in global coordinates) to the natural coordinates. For illustration, consider a grid made of four-node quadrilateral (Q4) elements. By employing the isoparametric concept one writes

$$x = \sum_{I=1}^{4} N_I(\xi, \eta) x_I^e, \quad y = \sum_{I=1}^{4} N_I(\xi, \eta) y_I^e \tag{5.8}$$

where (x_I^e, y_I^e) denote the nodal coordinates of element e that contains the particle (x, y) under consideration; $N_I(\xi, \eta)$ are the Q4 shape functions (see Fig. B.1 for the formula).

One solves Eq. (5.8) by using the iterative Newton-Raphson method:

$$\begin{bmatrix} \frac{\partial N_I}{\partial \xi}\Big|_{\xi_0} x_I^e & \frac{\partial N_I}{\partial \eta}\Big|_{\eta_0} x_I^e \\ \frac{\partial N_I}{\partial \xi}\Big|_{\xi_0} y_I^e & \frac{\partial N_I}{\partial \eta}\Big|_{\eta_0} y_I^e \end{bmatrix} \begin{bmatrix} \Delta\xi \\ \Delta\eta \end{bmatrix} = \begin{bmatrix} x - N_I(\xi_0, \eta_0) x_I^e \\ y - N_I(\xi_0, \eta_0) y_I^e \end{bmatrix} \tag{5.9}$$

which provides $\Delta\xi, \Delta\eta$ and the solution is updated $\xi = \xi_0 + \Delta\xi$, $\eta = \eta_0 + \Delta\eta$. The iteration continues until convergence is attained. The initial value for (ξ_0, η_0) are usually $(0, 0)$ i.e., the iterative procedure starts from the center of the element.

5.4.2 Particle Registration

Particle registration is required to perform the particle-to-node and node-to-particle mapping. For example, to calculate the nodal mass $m_I^t = \sum_p \phi_I(\mathbf{x}_p^t) m_p$, one needs to know which particle p contributes to which nodes I. While this step is fast and efficient when a Cartesian grid is used, the registration of particles to the grid in uMPM is inefficient as the grid is unstructured. The problem is how to quickly locate an element that contains a given particle. Just a few works discussed this problem. In Wang et al. (2005) a ray-crossing algorithm is employed to determine whether the material points are inside or outside of arbitrary quadrilateral cells. An in-depth analysis of this problem was given in Pruijn's master thesis (Pruijn 2017).

5.4.3 Mixed Integration

In the uMPM, the standard particle-based quadrature of the MPM is adopted for dynamics problems but each element is filled with many elements e.g. 10 material points per one single linear tetrahedral (Al-Kafaji 2013). For quasi-static problems, a mixed integration is adopted (Beuth et al. 2011). In this mixed integration scheme, Gaussian integration is applied to all elements that are fully filled with material, whereas material point based integration is only adopted for partially filled elements. Elements (cells) in the interior of a body are assumed to be fully filled. Conversely, partially filled elements are assumed to occur only along the boundary of a body. Precisely, an element located on the boundary of the body is considered to be partially filled when the volume sum of all particles inside this element is less than a prescribed percentage of the element volume.

5.4.4 uMPM with C^1 Shape Functions

We think that the mixed integration is not an elegant solution to cell-crossing issue. Recently, de Koster et al. (2019) developed C^1 basis functions over unstructured grids using Powell-Sabin functions (Powell and Sabin 1977; May et al. 2016). While this constitutes a C^1 uMPM formulation which is free of cell-crossing problem similar to BSMPM (MPM with a Cartesian grid and B-splines as weighting functions), evaluation of the Powell-Sabin functions seems complex and to the best knowledge of the authors, no work on 3D extension has been reported.

To close this discussion on the uMPM, we anticipate that a total Lagrangian uMPM would be the most efficient uMPM as the shape functions and its derivatives and the particle registration need only be calculated once for all.

5.5 Visualization

In the FEM, visualization is typically performed on the mesh. Information stored at the integration points are extrapolated to the nodes to this end. Additionally some averaging are carried out to obtain smooth fields as each node is connected to a number of elements. In MPM, the computational grid is fixed and thus not suitable for visualization. There are at least three approaches to visualizing MPM results (Childs et al. 2012)

- Particle visualization: the particles are visualized directly as spheres/circles (3D/2D). Particle data (position, stresses etc.) are written to files and can be processed by `Paraview` or `VisIt` or `Ovito`.[3]
- 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. In this way, available visualization technologies developed for the FEM can be reused. Note that this method is expensive since the particles positions are evolving in time. And then visualization can be done with `Paraview` or `VisIt`.
- Voxel-based visualization: each particle is assigned with a domain so that the particles are alike to pixels/voxels of digital images (Andersen and Andersen 2010b; Choudhury et al. 2010). This visualization technique is best suited for CPDI formulation.

Figure 5.18 illustrates particle based and mesh-based visualization. For particle-based visualization, `Ovito` (Stukowski 2009) is an excellent tool as it is a scientific visualization and analysis software for atomistic and particle simulation data. From our experiences, for MPM simulations, `Ovito` is faster and easier to use than both `Paraview` and `ViSit`.

[3] http://www.paraview.org, https://visit-dav.github.io/visit-website/, https://www.ovito.org.

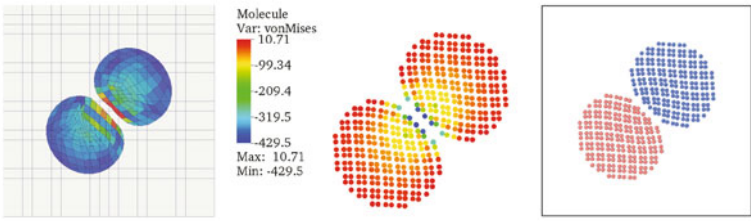


Fig. 5.18 CPDI and MPM/GIMP visualization of MPM data (de Vaucorbeil et al. 2020)

We refer to Sect. 6.12 for details on how to write particle data to a file supported by `Paraview/ViSit` and Sect. 7.3 for a presentation on the file format supported by `Ovito`.

Andersen and Andersen (2010a), Dunatunga and Kamrin (2015) showed that although the stress field at the individual material points may show a noisy variation, a smooth physically realistic stress field can be extracted by a mass-weighted mapping via the computational grid:

$$\begin{aligned} \boldsymbol{\sigma}_I &= \frac{1}{m_I}\left(\sum_p \phi_I(\mathbf{x}_p)\boldsymbol{\sigma}_p m_p\right) \\ \boldsymbol{\sigma}_p^{\text{smooth}} &= \sum_I \phi_I(\mathbf{x}_p)\boldsymbol{\sigma}_I \end{aligned} \tag{5.10}$$

We refer to Sect. 10.3.1 for illustrations of noisy particle results. It is interesting to note that Andersen and Andersen (2010a) realized that attempt to utilize these more realistic stresses $\boldsymbol{\sigma}_p^{\text{smooth}}$ directly in the computational MPM scheme has been unsuccessful. But it might be the motivation for Zhang et al. (2011) to have proposed the dual domain MPM.

References

Al-Kafaji, I.K.J.: Formulation of a Dynamic Material Point Method (MPM) for Geomechanical Problems. Ph.D. thesis, University of Stuttgart (2013)

Andersen, S., Andersen, L.: Modelling of landslides with the material-point method. Comput. Geosci. **14**(1), 137–147 (2010)

Andersen, S., Andersen, L.: Analysis of spatial interpolation in the material-point method. Comput. Struct. **88**(7–8), 506–518 (2010)

Bardenhagen, S.G., Brackbill, J.U., Sulsky, D.: The material-point method for granular materials. Comput. Methods Appl. Mech. Eng. **187**(3–4), 529–541 (2000)

Bardenhagen, S.G., Brydon, A.D., Guilkey, J.E.: Insight into the physics of foam densification via numerical simulation. J. Mech. Phys. Solids **53**(3), 597–617 (2005)

Belytschko, T., Liu, W.K., Moran, B.: Nonlinear Finite Elements for Continua and Structures. Wiley, Chichester, England (2000)

Beuth, L., Wieckowski, Z., Vermeer, P.A.: Solution of quasi-static large-strain problems by the material point method. Int. J. Numer. Anal. Methods Geomech. **35**(13), 1451–1465 (2011)
Bing, Y., Cortis, M., Charlton, T.J., Coombs, W.M., Augarde, C.E.: B-spline based boundary conditions in the material point method. Comput. Struct. **212**, 257–274 (2019)
Childs, H., Brugger, E., Whitlock, B., Meredith, J., Ahern, S., Pugmire, D., Biagas, K., Miller, M., Harrison, C., Weber, G.H., Krishnan, H., Fogal, T., Sanderson, A., Garth, C., Bethel, E.W., Camp, D., Rübel, O., Durant, M., Favre, J.M., Navrátil, P.: VisIt: an end-user tool for visualizing and analyzing very large data. In: High Performance Visualization–Enabling Extreme-Scale Scientific Insight, pp. 357–372 (2012)
Choudhury, A., Steffen, M., Guilkey, J., Parker, S.: Enchanced understanding of particle simulations through deformation-based visualization. Comput. Methods Eng. Sci. **63**, 117–136 (2010)
Cortis, Michael, Coombs, William, Augarde, Charles, Brown, Michael, Brennan, Andrew, Robinson, Scott: Imposition of essential boundary conditions in the material point method. Int. J. Numer. Methods Eng. **113**(1), 130–152 (2018)
de Vaucorbeil, A., Nguyen, V.P., Sinaie, S., Wu, J.Y.: Chapter two—material point method after 25 years: theory, implementation, and applications. Advances in Applied Mechanics, vol. 53, pp. 185–398. Elsevier (2020)
de Koster, P., Tielen, R., Wobbes, E., Moller, M.: Extension of B-spline material point method for unstructured triangular grids using powell-sabin splines. Comput. Mech. (2019)
Dunatunga, S., Kamrin, K.: Continuum modeling and simulation of granular flows through their many phases. J. Fluids Mech. (2015)
Guilkey, James E., Hoying, James B., Weiss, Jeffrey A.: Computational modeling of multicellular constructs with the material point method. J. Biomech. **39**(11), 2074–2086 (2006)
Hughes, T.J.R.: The Finite Element Method: Linear Static and Dynamic Finite Element Analysis. Dover Publications Inc., New York (2000). ISBN 0-486-41181-8. Corrected reprint of the 1987 original [Prentice-Hall Inc., Englewood Cliffs, NJ]
Hughes, T.J.R., Cottrell, J.A., Bazilevs, Y.: Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. Comput. Methods Appl. Mech. Eng. **194**(39–41), 4135–4195 (2005)
Jassim, I., Stolle, D., Vermeer, P.: Two-phase dynamic analysis by material point method. Int. J. Numer. Anal. Methods Geomech. **37**(15), 2502–2522 (2013)
Liu, C., Sun, W.: Shift boundary material point method: an image-to-simulation workflow for solids of complex geometries undergoing large deformation. Comput. Part. Mech. 1–18 (2019)
May, Stefan, Vignollet, Julien, de Borst, René: Powell-sabin b-splines and unstructured standard t-splines for the solution of the kirchhoff-love plate theory exploiting bézier extraction. Int. J. Numer. Methods Eng. **107**(3), 205–233 (2016)
Nairn, J.A.: Material point method simulations of transverse fracture in wood with realistic morphologies. Holzforschung **61**(4), 375–381 (2007)
Nguyen, V.P., de Vaucorbeil, A., Nguyen-Thanh, C., Mandal, T.K.: A generalized particle in cell method for explicit solid dynamics. Comput. Methods Appl. Mech. Eng. **360**, 112783 (2021). https://doi.org/10.1016/j.cma.2019.112783
Patzold, M.: The improvement of the material point method by increasing effciency and accuracy. Master's thesis, Universitaat Siegen (2016)
Powell, M.J.D., Sabin, M.A.: Piecewise quadratic approximations on triangles. ACM Trans. Math. Softw. (TOMS) **3**(4), 316–325 (1977)
Pruijn, N.S.: Graphical models and simulation for thz-imaging. Master's thesis, Delft University of Technology (2017)
Remmerswaal, G.: Development and implementation of moving boundary conditions in the material point method. Master's thesis, TU Delft (2017)
Sadeghirad, A., Brannon, R.M., Guilkey, J.E.: Second-order convected particle domain interpolation (CPDI2) with enrichment for weak discontinuities at material interfaces. Int. J. Numer. Methods Eng. **95**(11), 928–952 (2013)

Sethian, J.A.: Level set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision and Materials Science. Cambridge University Press, Cambridge, UK (1999)
Stukowski, A.: Visualization and analysis of atomistic simulation data with ovito-the open visualization tool. Modell. Simul. Mater. Sci. Eng. **18**(1), 015012 (2009)
Wang, B., Karuppiah, V., Lu, H., Komanduri, R., Roy, S.: Two-Dimensional mixed mode crack simulation using the material point method. Mech. Adv. Mater. Struct. **12**(6), 471–484 (2005)
Wieçkowski, Z., Sung-kie, Y., Jeoung-Heum, Y.: A particle-in-cell solution to the silo discharging problem. Int. J. Numer. Methods Eng. **45**, 1203–1225 (1999)
Wieçkowski, Z.: The material point method in large strain engineering problems. Comput. Methods Appl. Mech. Eng. **193**(39–41), 4417–4438 (2004)
Zhang, D.Z., Ma, X., Giguere, P.T.: Material point method enhanced by modified gradient of shape function. J. Comput. Phys. **230**(16), 6379–6398 (2011)

Chapter 6
MPMat: A MPM Matlab Code

This chapter presents a tutorial MPM code written in Matlab. The code and examples presented in this chapter are available at https://github.com/vinhphunguyen/mpm. We start from an implementation of the MPM for a single-material-point analysis and proceed to one dimensional continua consisting of multiple particles. Two dimensional MPM implementation is then presented followed by a 3D implementation. It should be emphasized that the code was written to be readable as much as possible and thus efficiency was not considered. We give enough details for users interested in modifying the code or write their own MPM code from scratch.

We have decided to follow a FEM implementation by considering particles as integration points. Therefore, all the summations either $\sum_p$ (particles to nodes operation) or $\sum_I$ (nodes to particles operation) are carried out by looping over grid cells and the particles contained in each cell. We call this implementation *element-based* which is familiar for researchers from the FEM community. There also exists a *particle-based* implementation which is also presented later in the manuscript.

The code organization is first presented in Sect. 6.1. We discuss Cartesian background grids in Sect. 6.2: how to generate them and grid data structure. Particle data (e.g. position, gradient deformation, stress etc.) are described in Sect. 6.3. We present some ways to generate particles for solids of simple geometries and complex geometries in Sect. 6.4. The solution phase i.e., the time stepping is treated in Sect. 6.5. One good thing about the MPM is its simplicity, and we show in Sect. 6.6 that 3D implementation is stringkingly identical to 2D one. We then discuss the implementation of GIMP in Sect. 6.7. Details regarding BSMPM (MPM with B-splines) are given in Sect. 6.8. The next three sections are devoted to the implementation of CPDI: CPDI-R4 in Sect. 6.9, CPDI-Q4, CPDI-T3 in Sect. 6.10 and polygonal CPDI in Sect. 6.11. Post-processing using Paraview is discussed in Sect. 6.12. Some improvements of the code are provided in Sects. 6.13 and 6.14. Finally, some examples are given to verify the implementation (Sect. 6.15). These examples are just simple simulations involving the collision of two dimensional elastic solids and high velocity impact of plastic solids. The idea is simply to illustrate the code and showcase how it is simple

V. P. Nguyen et al., *The Material Point Method*, Scientific Computation,
https://doi.org/10.1007/978-3-031-24070-6_6

to do impact/contact simulations using the MPM (involving only no-slip contacts). More challenging simulations signature of the MPM will be provided in subsequent chapters.

Note that this code only implements the ULMPM, TLMPM and GPIC are implemented in different codes (written in C++ and `Julia`) for efficiency. We refer to Chap. 7 and Appendix F for details on these C++/`Julia` codes.

6.1 Code Structure

The Matlab code is organized as follows

- **grid**: functions to generate 1D, 2D and 3D Cartesian grids;
- **basis**: functions to evaluate weighting functions and gradients;
- **particleGen**: functions to generate particles for simple solids;
- **postProcessing**: functions for VTK outputs;
- **fem**: implementation of ULFEM and TLFEM for hyperelastic solids;
- **example1D**: one dimensional examples;
- **example2D**: two dimensional examples;
- **example3D**: three dimensional examples.

In folder **example2D**, there are three sub-folders: **example2D/mpm** for MPM simulations, **example2D/cpdi** for CPDI examples and **example2D/gimp** for GIMP. The code implements only explicit MPMs.

As this is a tutorial code for learning the MPM, the code is written in the way that, to solve for a problem, the user has to write codes e.g. to build the grid, do the particle to node mapping etc. This is different from ready to use packages such as `Uintah`, http://www.uintah.utah.edu, or `Karamelo` described in Chap. 7, where users just need to prepare an input file.

6.2 Background Grid

In the 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. 6.1 for 2D cases. Note that the nodes are numbered starting from one to be compatible with one-based array indexing used in the Matlab language. The node coordinates are stored in a matrix `nodes` of dimension $n \times 2$ where n denotes the number of nodes and the elements are put in the matrix `element` of dimension $nel \times 4$, where nel is the number of elements

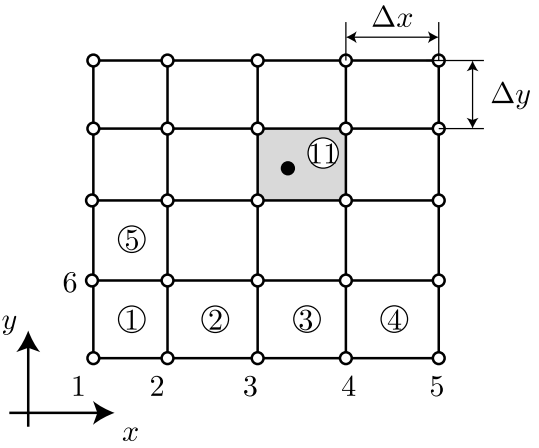


Fig. 6.1 A two dimensional structured grid

$$\texttt{nodes} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix}, \quad \texttt{elements} = \begin{bmatrix} n_1^{(1)} & n_2^{(1)} & n_3^{(1)} & n_4^{(1)} \\ n_1^{(2)} & n_2^{(2)} & n_3^{(2)} & n_4^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ n_1^{(nel)} & n_2^{(nel)} & n_3^{(nel)} & n_4^{(nel)} \end{bmatrix} \tag{6.1}$$

Each row of the `elements` matrix gives the nodes of an element.

Other grid nodal quantities include nodal mass, nodal momenta and nodal forces. For example, the nodal mass and momenta are stored using the following 1D and 2D arrays

$$\texttt{nmass} = \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_n \end{bmatrix}, \quad \texttt{nmomentum} = \begin{bmatrix} (mv)_{x1} & (mv)_{y1} \\ (mv)_{x2} & (mv)_{y2} \\ \vdots & \vdots \\ (mv)_{xn} & (mv)_{yn} \end{bmatrix} \tag{6.2}$$

To keep the implementation similar to FEM as much as possible, at every time step, the elements must know which particles they are storing. For element e, `mpoints{e}` returns the indices of the particles located inside e. The idea is based on the fact that, for a particle p with coordinates (x_p, y_p), it is straightforward to track which element it belongs to using the following equation

$$e = [\text{floor}((x_p - x_{\min})/\Delta x) + 1] + \texttt{numx}[\text{floor}((y_p - y_{\min})/\Delta y)] \tag{6.3}$$

where the floor function is the function that takes as input a real number x and gives as output the greatest integer less than or equal to x, `numx` denotes the number of elements along the x direction and $(x_{\min}, y_{\min})$ are the minimum node coordinates and Δx, Δy are the nodal spacing in the x and y directions, respectively. The complete algorithm is shown in Listing 6.1.

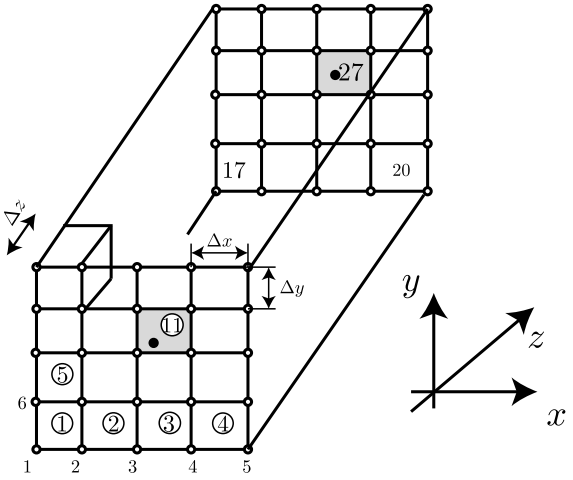


Fig. 6.2 A three dimensional structured grid. The nodes are numbered from left to right in the x direction, then from bottom to up in the y direction for the plane $z = z_{\min}$. Then, for the next plane up to $z = z_{\max}$

Listing 6.1 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
```

Extension to 3D is straightforward and if the elements are numbered as shown in Fig. 6.2, then the particle $\mathbf{x}_p = (x_p, y_p, z_p)$ belongs to element with index determined using the following equation

$$e = [\text{floor}((x_p - x_{\min})/\Delta x) + 1] + \text{numx}[\text{floor}((y_p - y_{\min})/\Delta y)] + \text{numx} \cdot \text{numy}[\text{floor}((z_p - z_{\min})/\Delta z)] \tag{6.4}$$

Remark 35 It should be emphasized that for structured meshes it is also possible to find directly the nodes to which a particle will interpolate its information rather than via finding the element as previously presented. The element based implementation is, however, general and can thus be applied for unstructured meshes.

Listing 6.2 shows the code to generate the grid and initialize the nodal quantities. Since creating a 2D Cartesian grid is standard, we do not present further explanations here. The readers are referred to the source code for details.

Listing 6.2 Two dimensional MPM: grid creation and initialization of nodal quantities.

```
% build the structured grid:
% mesh.node -> node coordinates, mesh.element -> element connectivity
% grid generation functions in folder mpm/grid/
[mesh]     = buildGrid2D (Lx,Ly,noX0,noY0, ghostCell); % ghostCell = 1: CPDI, GIMP
nodeCount = mesh.nodeCount;
elemCount = mesh.elemCount;
% 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
```

6.3 Particle Data

Standard particle quantities include position, velocity, mass, volume, stresses, strains and gradient deformation. Scalar particle quantities (such as mass and volume) are stored as column vectors and vectorial quantities (such as velocity, stress and strains) as matrices in which the columns are for the different components (Listing 6.3). Other particle data such as temperature, pressure or plastic strains can be added if needed.

Listing 6.3 Two dimensional MPM: particle data initialization.

```
pCount = numelem;                % # of particles
Mp  = ones(pCount,1);            % mass
Vp  = ones(pCount,1);            % updated volume
Fp  = ones(pCount,4);            % gradient deformation
s   = zeros(pCount,3);           % stress, Voigt notation
eps = zeros(pCount,3);           % strain
vp  = zeros(pCount,2);           % velocity
xp  = zeros(pCount,2);           % position
Vp0 = Vp;                        % old volume
```

In our code, we store second-order symmetric tensors such as the Cauchy stress and the strain tensor as 6×1 (column) vectors using the Voigt notation. The Cauchy stress vector and strain vector are stored as the following column vectors

$$\boldsymbol{\sigma} = \left[\sigma_{xx}\ \sigma_{yy}\ \sigma_{zz}\ \sigma_{yz}\ \sigma_{xz}\ \sigma_{xy}\right]^{\mathrm{T}}, \quad \boldsymbol{\epsilon} = \left[\epsilon_{xx}\ \epsilon_{yy}\ \epsilon_{zz}\ 2\epsilon_{yz}\ 2\epsilon_{xz}\ 2\epsilon_{xy}\right]^{\mathrm{T}} \tag{6.5}$$

where the off diagonal strain components are doubled to ensure that $\boldsymbol{\sigma} : \boldsymbol{\epsilon} = \boldsymbol{\sigma}^{\mathrm{T}}\boldsymbol{\epsilon}$. Note that storing symmetric tensors as column vectors is not a requirement. Actually, later on we present a C++ code named `Karamelo` in which symmetric strain and stress tensors are stored as 3×3 matrices. This is to make easy for matrix operations such as the polar decomposition of the deformation gradient tensor.

The internal force vector is written explicitly as follows

$$
\begin{aligned}
f_{xI}^{\text{int}} &= -\sum_{p=1}^{n_p} V_p \left[(\sigma_{xx})_p \frac{\partial N_I}{\partial x}(\mathbf{x}_p) + (\sigma_{xy})_p \frac{\partial N_I}{\partial y}(\mathbf{x}_p) + (\sigma_{xz})_p \frac{\partial N_I}{\partial z}(\mathbf{x}_p) \right] \\
f_{yI}^{\text{int}} &= -\sum_{p=1}^{n_p} V_p \left[(\sigma_{xy})_p \frac{\partial N_I}{\partial x}(\mathbf{x}_p) + (\sigma_{yy})_p \frac{\partial N_I}{\partial y}(\mathbf{x}_p) + (\sigma_{yz})_p \frac{\partial N_I}{\partial z}(\mathbf{x}_p) \right] \\
f_{zI}^{\text{int}} &= -\sum_{p=1}^{n_p} V_p \left[(\sigma_{zx})_p \frac{\partial N_I}{\partial x}(\mathbf{x}_p) + (\sigma_{zy})_p \frac{\partial N_I}{\partial y}(\mathbf{x}_p) + (\sigma_{zz})_p \frac{\partial N_I}{\partial z}(\mathbf{x}_p) \right]
\end{aligned}
\tag{6.6}
$$

And the corresponding simplification for 2D is straightforward.

6.4 Particle Generation

6.4.1 *Particle Generation Using a Mesh*

Particles can be generated using a FE meshing program. For example, the code snippet given in Listing 6.4 creates particles as centers of the elements of a FE mesh (defined by two data structures–`node1` and `element1` for the node coordinates and the element connectivity, respectively).

Listing 6.4 Two dimensional MPM: particle initialization from FE mesh.

```
% particle data initialisation: see Listing 6.3
% initialise particle position, mass ,volume, velocity
for e = 1:numelem
    coord = node1(element1(e,:),:);
    a     = det([coord,[1;1;1]])/2;
    Vp(e) = a;
    Mp(e) = a*rho;
    xp(e,:) = mean(coord);
    vp(e,:) = [v v];
    Fp(e,:) = [1 0 0 1];
end
Vp0 = Vp;
```

6.4.2 *Particle Generation for Simple Geometries*

For simple geometries such as rectangles, circles, cylinders, etc., one can generate the particles directly. Herein, we present a technique to efficiently generate particles for MPM simulations. For the sake of simplicity, only 2D is considered but the principles are general enough.

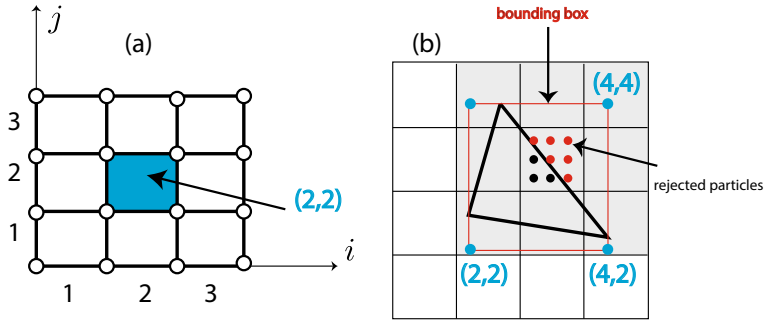


Fig. 6.3 Grid cells are indexed by (i, j) in the integer index space (**a**), and this is used to quickly identity cells which are closest to a given geometry object based on the bounding box concept (**b**). Instead of distributing particles in all cells (16) one only does this for 9 cells

If there are n grid cells and m geometrical objects a naive algorithm by sweeping over the cells and for each cell loop over all the objects would result in an algorithm of order $\mathcal{O}(n \times m)$. This is inefficient if both n and m are large. A better algorithm considers only those cells intersecting with a given geometrical object. This can be achieved using the bounding box concept and integer cell coordinates (Fig. 6.3).

First an integer cell coordinate (i, j) is introduced as shown in Fig. 6.3a. Note that the index is numbered from one since it was coded in Matlab. Let assume that one needs to generate the particles for a triangle given in Fig. 6.3b. We can determine the cells intersecting with this triangle (shaded cells in the referred figure) based on the bounding box of the triangle and the cell indices. Finally one loops only over those cells and distribute particles. A simple check whether a particle is within a polygon is used to discard particles outside the triangle. Listing 6.5 gives a code snippet implementing the aforementioned particle generation technique.

The index of element with coordinates (i, j) is given by

$$e = i + \texttt{numx} \times (j - 1) \tag{6.7}$$

Some functions used to generate particlels for simple 2D geometries are collected in Listing 6.6.

Listing 6.5 Matlab script to generate particles for a polygon. `mesh` is a structure storing the background mesh, `xmin`, `xmax`, `ymin`, `ymax` define the bounding box of the polygon.

```
% find element index (i,j) of the lower left and upper right corners of
% the bounding box of the polygon
minIndex = point2ElemIndexIJ([xmin ymin],mesh);
maxIndex = point2ElemIndexIJ([xmax ymax],mesh);

iMin     = minIndex.i; iMax     = maxIndex.i;
jMin     = minIndex.j; jMax     = maxIndex.j;
% number of cells intersecting with the polygon
noElems  = (iMax-iMin+1) * (jMax-jMin+1);
res.elems = zeros(noElems,1);
dx       = mesh.deltax/(ppc); dy  = dx;
ii       = 1;
for i = iMin:iMax              % first 2 loops over cells closest to polygon
  for j = jMin:jMax
    id  = i + mesh.numx* ( j - 1);
    res.elems(ii) = id; ii  = ii + 1;
    sctr = mesh.element(id,:); % element scatter vector
    pts  = mesh.node(sctr,:);
    x1   = pts(1,:);           % first corner of the cell
    for ip = 1:ppc             % distribute particles in x and y dirs.
      for jp = 1:ppc
        x(1) = x1(1) + dx*0.5 + (jp-1)*dx;
        x(2) = x1(2) + dy*0.5 + (ip-1)*dy;
      end
    end
  end
end
```

Listing 6.6 Some Matlab functions for particle generation.

```
function [res] = generateMPForCircle       (geo,ppc,mesh)
function [res] = generateMPForQuad         (geo,ppc,mesh)
function [res] = generateMPQuadDiffCircles(quad,circle1,circle2,ppc,mesh)
```

6.5 Solution Algorithm

The functions to evaluate the hat functions and first derivatives are given in Listing 6.7. Usage of these functions are illustrated in Listing 6.9.

Listing 6.7 Some Matlab functions for basis functions and derivatives (source is in folder **/basis**).

```
function [phi,dphi]=getMPM(x,h)
function [phi,dphi]=getMPM2D(x,hx,hy)      % x = [x1,x2]
function [phi,dphi]=getMPM3D(x,hx,hy,hz)   % x = [x1,x2,x3]
```

Solution phase is implemented in Listing 6.8 with details given in Listings 6.9 and 6.10. At specified time intervals VTK outputs are written (lines 14–18 in Listing 6.8). The implementation was for a linear elastic material in which **C** denotes the constitutive matrix (lines 23–25 in Listing 6.10). Modification for other constitutive models is straightforward.

Listing 6.8 Two dimensional MPM: solution phase (explicit).

```
1  while ( t < time )
2      % reset grid data
3      nmass(:)      = 0;
4      nmomentum(:) = 0;
5      niforce(:)    = 0;
6      % particle to grid nodes
7      See Listing 6.9
8      % update nodal momenta
9      nmomentum = nmomentum + niforce*dtime;
10     % nodes to particles
11     See Listing 6.10
12     % update the element particle list, see Listing 6.1
13     % VTK output
14     if ( mod(istep,interval) == 0 )
15         xp = pos{istep};
16         vtkFile = sprintf('../results/%s%d',vtkFileName,istep);
17         VTKParticles(xp,vtkFile,s);
18     end
19     % advance to the next time step
20     t     = t + dtime; istep = istep + 1;
21 end
```

Listing 6.9 Two dimensional MPM: particles to nodes.

```
1  for e=1:elemCount                % loop over elements
2     esctr = element(e,:);         % element connectivity
3     enode = node(esctr,:);        % element node coords
4     mpts  = mpoints{e};           % particles inside element e
5     for p=1:length(mpts)          % loop over particles
6         pid    = mpts(p);         % particle ID
7         stress = s(pid,:);
8         for i=1:length(esctr)  % loop over nodes of cell 'e'
9            id             = esctr(i);      % node ID
10           x              = xp(pid,:) - node(id,:);
11           [N,dNdx]       = getMPM2D(x,mesh.deltax,mesh.deltay);
12           dNIdx          = dNdx(i,1);
13           dNIdy          = dNdx(i,2);
14           nmass(id)          += N(i)*Mp(pid);
15           nmomentum(id, :) += N(i)*Mp(pid)*vp(pid,:);
16           niforce(id,1)     -= Vp(pid)*(stress(1)*dNIdx + stress(3)*dNIdy);
17           niforce(id,2)     -= Vp(pid)*(stress(3)*dNIdx + stress(2)*dNIdy);
18        end
19    end
20 end
```

Listing 6.10 Two dimensional MPM: nodes to particles (USL and cutoff).

```
for e=1:elemCount                % loop over elements
    esctr = element(e,:);
    enode = node(esctr,:);
    mpts  = mpoints{e};
    for p=1:length(mpts)         % loop over particles
        pid  = mpts(p);
        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 velocity gradient
        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
```

6.6 Three Dimensions

The implementation of 3D MPM follows exactly the 2D procedure and therefore not discussed here in great details. We refer to the source code, e.g. M file **MPM3DTwoDisksMUSL.m** for details. Listing 6.11 shows some minor differences of a 3D MPM code compared to a 2D MPM. As Matlab is slow, for 3D simulations, we prefer to use a C++ code (Chap. 7 presents such a code) or a Julia code (refer to Appendix F.3), both are much more efficient than the Matlab code.

Listing 6.11 Three dimensional MPM: minor differences compared to 2D MPM.

```
% find which elements contain which particles
x = xp(ip,1); y = xp(ip,2); z = xp(ip,3);
e = floor(x/deltax) + 1 + numx*floor(y/deltay) + numx*numy*floor(z/deltaz) ;
elems(ip) = e;
% compute N_I and derivatives at particle xp
x      = xp - node(id,:);
[N,dNdx]=getMPM3D(x,deltax,deltay,deltaz);
% nodal internal force fint_I
niforce(id,1)   = niforce(id,1) - Vp*(stress(1)*dNIdx + stress(6)*dNIdy
                + stress(5)*dNIdz);
niforce(id,2)   = niforce(id,2) - Vp*(stress(6)*dNIdx
                + stress(2)*dNIdy + stress(4)*dNIdz);
niforce(id,3)   = niforce(id,3) - Vp*(stress(5)*dNIdx + stress(4)*dNIdy
                + stress(3)*dNIdz);
```

6.7 Implementation of (u/cp)GIMP

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 in which they locate but also to the nodes of neighboring elements (Fig. 6.4). As there are 16 non-zero basis functions within a cell (2D), for a particle p one needs to compute 16 ϕ_{Ip}. Note that out of 16 these functions only 9 are non-zero. The cpGIMP is more involved as one has to determine the element connectivity at every time steps since the particle domain expands in time. However, this step is relatively fast for structured grid.

Listing 6.12 is used to build the element connectivity for uGIMP in 2D. The function **getNeighbors** returns the eight neighbor elements and the element itself for a given element. Note that the connectivity array has different dimensions for elements on the boundary compared to interior elements. That is why a *cell* data structure was utilized. In order to illustrate the minor modifications to the MPM code, Listing 6.13 presents the code for particle to node mapping. Only lines 2 and 11 are different from a standard MPM code.

Listing 6.12 Two dimensional uGIMP: grid connectivity construction.

```
1  gimpElement = cell(elemCount,1);
2
3  for e=1:elemCount
4      neighbors       = getNeighbors(e, numx2, numy2);
5      neighborNodes   = element(neighbors,:);
6      gimpElement{e} = unique(neighborNodes);
7  end
```

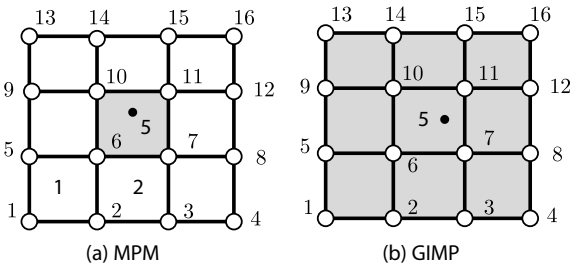


Fig. 6.4 Larger element connectivity in GIMP (**b**) than in MPM (**a**). The connectivity of an element in GIMP can be determined by getting firstly its neighboring elements (a cheap task for structured grids used in MPM/GIMP) and then the nodes of these elements

Listing 6.13 Two dimensional uGIMP: particles to nodes mapping

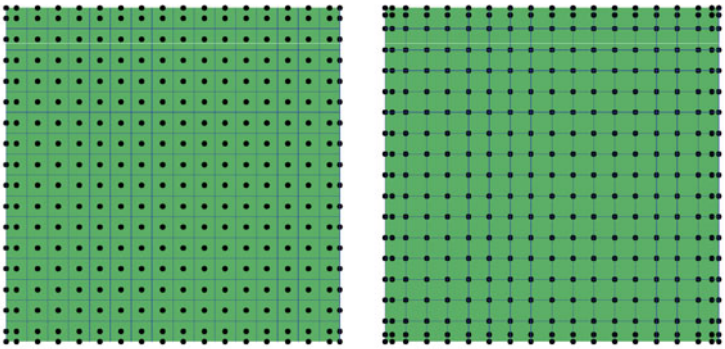
```
for e=1:elemCount
      esctr = gimpElement{e};     % GIMP (extended) element connectivity
      enode = node(esctr,:);      % element node coords
      mpts  = mpoints{e};         % particles inside element e
      for p=1:length(mpts)        % loop over particles
          pid  = mpts(p);
          % particle mass and momentum to node
          for i=1:length(esctr)
              id    = esctr(i);
              x     = xp(pid,:) - node(id,:);
              [N,dNdx]=getGIMP2D(x,deltax,deltay,lpx,lpy);
              dNIdx = dNdx(1); dNIdy = dNdx(2);
              nmass(id)       = nmass(id)       + N*Mp(pid);
          end
      end
end
```

6.8 B-splines MPM

We present the implementation of the BSMPM using the recursive formula (Sect. 6.8.1) and the Bézier operators (Sect. 6.8.2). The latter simplifies the implementation and speeds up the computation of B-splines functions. Compared to the boundary modified B-splines presented in Sect. 3.4, this implementation is general as it can implement B-splines of any order. We refer to Appendix F for reader interested in how the modified B-splines are implemented.

6.8.1 Recursive B-splines MPM

In this section we present the implementation of MPM using B-splines described in Sect. 3.4. Listing 6.14 shows the code that builds a linearly parametrized bi-quadratic B-spline surface for a square of unit side. The code uses the NURBS toolbox of de Falco et al. (2011). Line 12 performs a k-refinement where the last two input parameters control the h-refinement. Only a simple uniform refinement where knot spans are halved is implemented. Line 13 builds a FE mesh (notably the element connectivity matrix) from the B-spline surface. B-spline meshes produced by this code are given in Fig. 6.5. Listing 6.15 presents some codes illustrating the use of a B-spline mesh in MPM. Note that function **BSPLINE2DBasisDers** is implemented in a C-MEX file to speed up the computation of B-spline basis functions which are recursively defined, cf. Eq. (3.17). The functions on B-splines geometries and corresponding mesh are discussed in our isogeometric finite elements review article (Nguyen et al. 2015). We refer to the file **example2D/mpm2DTwoDisksBsplines.m** for details.



(a) bi-quadratic (b) bi-cubic

Fig. 6.5 B-spline meshes of a square (16×16 elements): **a** bi-quadratic and **b** bi-cubic. The control points or nodes are represented by black dots

Listing 6.14 Building a 2D B-spline mesh.

```
L = 1; w = 1;
controlPts          = zeros(4,2,2);
controlPts(1:2,1,1) = [0;0];
controlPts(1:2,2,1) = [L;0];
controlPts(1:2,1,2) = [0;w];
controlPts(1:2,2,2) = [L;w];
controlPts(4,:,:)   = 1;  % B-splines weights=1
uKnot = [0 0 1 1]; vKnot = [0 0 1 1];
pnew    = 2; % new order basis in x dir
qnew    = 2; % new order basis in y dir
surf    = nrbmak(controlPts,{uKnot vKnot});   % Bspline surface
surf    = doKRefinementSurface(surf,pnew,qnew,3,3);% k-refinement,8x8 mesh
igaMesh = buildIGA2DMesh(surf);                % mesh
```

Listing 6.15 B-spline basis functions in action.

```
for e=1:elemCount
    esctr  = igaMesh.globElems (e,:);      % element connectivity
    pts    = igaMesh.controlPts(esctr,:); % element nodal coords
    mpts   = mpoints{e};                   % particles inside element e
    for p=1:length(mpts)                   % loop over particles
       pid  = mpts(p);
       x    = xp(pid,1);
       y    = xp(pid,2);
       xi   = (x-xMin)/L;                   % parameter coords.
       et   = (y-yMin)/w;
       [N, dNdxi, dNdeta] = BSPLINE2DBasisDers([xi;et],igaMesh.p,...
              igaMesh.q, igaMesh.uKnot,igaMesh.vKnot);
       jacob   = pts' * [dNdxi' dNdeta'];
       dNdx    = [dNdxi' dNdeta'] * inv(jacob);
    end
end
```

6.8.2 *Bézier Extraction B-splines MPM*

Although the B-splines as presented in the previous section are sufficient for use in the MPM, evaluation of a recursive function is slow and should be avoided especially in the setting of the MPM. In this section, advancements done in IGA are borrowed to have a fast implementation of B-splines functions: the so-called Bézier extraction. Firstly, we recall the univariate Bernstein basis functions of order p that are defined over the biunit interval $[-1, 1]$ as

$$B_{i,p}(\xi) = \frac{1}{2^p}\binom{p}{i-1}(1-\xi)^{p-(i-1)}(1+\xi)^{i-1}, \tag{6.8}$$

where $\binom{p}{i-1}$ is the binomial coefficient $\binom{p}{i-1} = \frac{p!}{(i-1)!(p+1-i)!}$, $1 \le i \le p+1$. We emphasize that, in CAD Bernstein polynomials are defined on interval $[0, 1]$. However, in a finite element setting, the bi-unit interval $[-1, 1]$, where the Gauss quadrature is defined, is preferable. The nice thing about the Bernstein basis is that they can be hard coded for any p.

The first derivative of the Bernstein basis is defined in terms of low order basis as follows

$$\frac{\partial B_{i,p}(\xi)}{\partial \xi} = \frac{1}{2}p\left[B_{i-1,p-1}(\xi) - B_{i,p-1}(\xi)\right]. \tag{6.9}$$

The shape functions and first derivatives for element e read

$$\mathbf{N}^e(\boldsymbol{\xi}) = \mathbf{C}^e\mathbf{B}^e(\boldsymbol{\xi}), \quad \mathbf{N}^e_{,\xi}(\boldsymbol{\xi}) = \mathbf{C}^e\mathbf{B}^e_{,\xi} \tag{6.10}$$

where $\mathbf{C}^e$ is the elemental Bézier extraction operator and $\mathbf{B}^e$ are the Bernstein polynomials. Index space, knot vectors are all embedded in the Bézier extractors which are computed in a pre-processing step (line 2 in Listing 6.16). Note that if in the above equation $\mathbf{C}^e$ is omitted then one obtains high order C^0 grid functions that are quite similar to high order Lagrange functions except that the Bernstein functions are always positive. We refer to the file **example2D/mpm2DTwoDisksBezier.m** for details.

Listing 6.16 B-spline basis functions using the Bézier extraction.

```
1 % compute Bezier extraction operators
2 [C,Cxi,Cet] = bezierExtraction2D(uKnot,vKnot,p,q);
3 % compute Bernstein basis and derivatives at (xi,eta)
4 % at element 'e'
5 [B dB] = getShapeGradBernstein2D(p, q, xi, eta);
6 % the grid shape function
7 N      = C(:,:,e) * B;
```

6.9 Implementation of CPDI-R4

Contrary to the standard MPM where particles are simply points, those in the CPDI have extent and thus require a special implementation. In what follows, we present a simple data structure for the finite-extent particles (Sect. 6.9.1) and how to evaluate the basis functions (Sect. 6.9.2) in MPMat. We also made a slight modification in the code by switching to a particle based implementation i.e., there is no longer a loop over the grid cells. Instead, we directly sweep over the particles. Although the CPDI-R4 is a special case of CPDI2s (CPDI-Q4, CPDI-T3), we have decided to still present the implementation for the CPDI-R4. Next section will be devoted to CPDI2s.

6.9.1 Data Structure for Particles

In addition to the standard particle data, in the CPDI-R4, one needs to store the particle domain vectors $\mathbf{r}_1^0$, $\mathbf{r}_2^0$, $\mathbf{r}_1$ and $\mathbf{r}_2$. Listing 6.17 is a Matlab code used to compute the initial particle domain vectors for the case they are rectangles.

Listing 6.17 Data structure to store particle domain vectors.

```
lpx      = mesh.deltax/noParticleX;
lpy      = mesh.deltay/noParticleY;
dvec1    = zeros(pCount,2);              % domain vector 1, r1
dvec2    = zeros(pCount,2);              % domain vector 2, r2
for p=1:pCount
    dvec1(p,1) = 0.5*lpx;
    dvec1(p,2) = 0;
    dvec2(p,1) = 0;
    dvec2(p,2) = 0.5*lpy;
end
dvec10 = dvec1;                          % r10
dvec20 = dvec2;
```

6.9.2 Evaluation of ϕ_{Ip} and $\nabla\phi_{Ip}$

The algorithm to compute the CPDI-R4 weighting functions and derivatives for a given particle is given in Algorithm 12. The corresponding Matlab implementation (details were skipped to save space) is shown in Listing 6.18.

Algorithm 12 Algorithm to evaluate CPDI-R4 shape functions/derivatives.

1: Determine the four corners of the particle domain, Eq. (3.35);
2: Compute the four gradient weights w_c^g;
3: For each corner, determine which element contains it;
4: Get the nodes of the four elements that contain the four corners;
5: For each node of those, loop over the corner $\mathbf{x}_c$, compute $N_I(\mathbf{x}_c)$ and use Eq. (3.33).

Listing 6.18 Matlab function to evaluate CPDI-R4 shape function/derivatives.

```
function data = cpdi2D(xp,r1p,r2p,mesh)
% xp: particle position; r1p: particle domain vector
% mesh: background grid
% four corners of the particle domain
x1 = xp - r1p - r2p;
x2 = xp + r1p - r2p;
% gradient weights
w    = zeros(4,2);
w(1,:) = [r1p(2)-r2p(2) r2p(1)-r1p(1)];
w(2,:) = [r1p(2)+r2p(2) -r1p(1)-r2p(1)];
% find elements contain the corners
% compute phi_I(xp) and first derivatives, see Listing 28
data.phi  = phi;
data.dphi = dphi;
data.node = nodes; %indices of nodes to which xp contributes
```

6.9.3 Time Advance

Compared to the previous MPM implementation, in CPDI there is no loop over the grid cells. In stead, one directly sweeps over the particles and for a given particle find the nodes to which it contributes.[1] Listing 6.19 shows the code snippet to compute nodal mass and momentum from the particle data. One extra step, compared to MPM and uGIMP, at the end of a time step is to update the particle domain vectors cf. Eq. (3.28).

[1] Obviously one can implement MPM without looping over the cells. However we decided to keep that implementation which is similar to FEM procedure so that the transition from FEM to MPM is rather smooth.

Listing 6.19 Particles to nodes mapping using CPDI.

```
for p=1:pCount  % loop over particles
  sig     = stress(p,:);
  xp      = coord(p,:);
  r1p     = dvec1(p,:);
  r2p     = dvec2(p,:);
  % particle mass and momentum to node
  data  = cpdi2D(xp,r1p,r2p,mesh);
  esctr = data.node;
  for i=1:length(esctr) % loop over nodes to which xp contributes
      id              = esctr(i);
      nmass(id)       = nmass(id)       + data.phi(i)*mass(p);
      nmomentum(id,:) = nmomentum(id,:) + data.phi(i)*mass(p)*velo(p,:);
  end
end
```

6.10 Implementation of CPDI2s (CPDI-Q4, CPDI-T3)

In CPDI2s one needs to track the corners of the particle domains not the particle positions. Therefore, it is convenient to store the particle domains as a FE mesh which is called *the particle mesh*—`particles.node` and `particles.elem`—that consists of the nodes which are the corners and the element connectivity array. The latter is used to quickly retrieve the corners of a given particle (one can consider a particle an element), cf. Listing 6.20. This particle mesh can be a structured mesh or an unstructured mesh obtained from a mesh generator such as `Gmsh`.

Listing 6.20 Particle mesh data structure.

```
% pmesh.node:     nnode x 2 matrix where node(i,:)->[xi yi]
% pmesh.element:  nelem x 4 matrix for Q4 elements
% store the particle mesh into a structure for convenience
particles.node    = pmesh.node;
particles.elem    = pmesh.element;
```

Before discussing our implementation of the CPDI-Q4 weighting function and gradient, we recall that they, cf. Eqs. (3.37) and (3.38), can be written in the following compact form

$$\phi_{Ip} = \sum_{c=1}^{4} w_c^f N_I(\mathbf{x}_c) \tag{6.11}$$

$$\nabla\phi_{Ip} = \sum_{c=1}^{4} \mathbf{w}_c^g N_I(\mathbf{x}_c) \tag{6.12}$$

In our code, for a given particple p, we proceed in two steps. First, we determine the four weighting coefficients w_c^f and the four gradient coefficients $\mathbf{w}_c^g$. They can be computed using the geometry of the particle domain. We also determine all the

nodes I where ϕ_{Ip} might be non-zero. Listing 6.21 implements this step. In the second step, we use Eq. (6.12) with the data just computed in the first step, to compute the weighting function and the gradient. Listing 6.22 implements this step.

Listing 6.21 Calculation of CPDI-Q4 particle data.

```
function data = getCPDIQuadData(pid,particle,mesh)
% Inputs:
% pid:       particle index
% particle:  particle mesh
% mesh:      background mesh/grid
% Output:
% data.wf:     function weights
% data.wf:     gradient weights
% data.nodes: indices of nodes influencing particle "pid"

% four corners of the particle domain
nodeIds = particle.elem(pid,:);
corners = particle.node(nodeIds,:);
% particle domain area
Vp      = 0.5*   ( corners(1,1)*corners(2,2)  - corners(2,1)*corners(1,2) ...
                 + corners(2,1)*corners(3,2)  - corners(3,1)*corners(2,2) ...
                 + corners(3,1)*corners(4,2)  - corners(4,1)*corners(3,2) ...
                 + corners(4,1)*corners(1,2)  - corners(1,1)*corners(4,2) );
% function and gradient weights
c1    = (corners(2,1)-corners(1,1))*(corners(4,2)-corners(1,2)) - ...
        (corners(2,2)-corners(1,2))*(corners(4,1)-corners(1,1));
c2    = (corners(2,1)-corners(1,1))*(corners(3,2)-corners(2,2)) - ...
        (corners(2,2)-corners(1,2))*(corners(3,1)-corners(2,1));
c3    = (corners(3,1)-corners(4,1))*(corners(4,2)-corners(1,2)) - ...
        (corners(3,2)-corners(4,2))*(corners(4,1)-corners(1,1));
c4    = (corners(3,1)-corners(4,1))*(corners(3,2)-corners(2,2)) - ...
        (corners(3,2)-corners(4,2))*(corners(3,1)-corners(2,1));

wf     = (1/(36*Vp))*[4*c1+2*c2+2*c3+c4    2*c1+4*c2+c3+2*c4
                      c1+2*c2+2*c3+4*c4  2*c1+c2+4*c3+2*c4];

wg(1,:) = [corners(2,2)-corners(4,2) corners(4,1)-corners(2,1)];
wg(2,:) = [corners(3,2)-corners(1,2) corners(1,1)-corners(3,1)];
wg(3,:) = -wg(1,:);
wg(4,:) = -wg(2,:);

wg(:,:) = (1/(2*Vp))*wg(:,:);
% find elements contain the corners
elems   = zeros(4,1); % indices of elements of 4 corners
for c=1:4
    xc         = corners(c,:);
    elems(c)   = point2ElemIndex(xc,mesh);
end
% nodes I where phi_I(xp) are non-zero
nodes = unique(mesh.element(elems,:));
data.nodes = nodes;
data.wf    = wf;
data.wg    = wg;
data.Vp    = Vp;
```

Listing 6.22 Matlab function to evaluate CPDI-Q4 shape function/derivatives.

```
function data = getCPDIQuadBasis(pid,input,particle,mesh)
% Inputs:
% pid:       particle index
% particle:  particle mesh
% mesh:      background mesh/grid
% four corners of the particle domain
nodeIds = particle.elem(pid,:);
corners = particle.node(nodeIds,:);

Vp    = input.Vp;
nodes = input.nodes;
wf    = input.wf;
wg    = input.wg;
% compute phi_I(xp) and first derivatives
nodeCount = length(nodes);
phi       = zeros(nodeCount,1);
dphi      = zeros(nodeCount,2);

for i=1:nodeCount
    xI = mesh.node(nodes(i),:);
    for c=1:4
        x          = corners(c,:) - xI;
        [N,dNdx] = getMPM2D(x,mesh.deltax,mesh.deltay);
        phi(i)   = phi(i)    + wf(c)  *N;
        dphi(i,:)= dphi(i,:) + wg(c,:)*N;
    end
end
data.phi  = phi;
data.dphi = dphi;
data.node = nodes;
```

Listing 6.23 Particles to nodes mapping using CPDI-Q4.

```
for p=1:pCount
    sig    = stress(p,:);
    input  = getCPDIQuadData (p,particle,mesh)
    data   = getCPDIQuadBasis(p,input,particles,mesh);
    esctr  = data.node;
    for i=1:length(esctr)
        id              = esctr(i);
        nmass(id)       = nmass(id)       + data.phi(i)*mass(p);
        nmomentum(id,:) = nmomentum(id,:) + data.phi(i)*mass(p)*velo(p,:);
        niforce(id,1)   = niforce(id,1) - volume(p)*(sig(1)*data.dphi(i,1)
         + sig(3)*data.dphi(i,2));
    end
end
```

Listing 6.23 presents the code used to map particle data to grid nodes. After updating the particle velocities and stresses (not positions), one needs to update the position of the corners of the particle domains or equivalently the nodes of the particle mesh, cf. Eq. (3.39). Listing 6.24 is the code used for this purpose. Note that as neighboring particles share nodes (nodes of the particle mesh or particle corners) we do not loop over the particles and update its corners.

Listing 6.24 Updating corners of the particle domains.

```
for c=1:size(particles.node,1)          % loop over all particle nodes
    xc    = particles.node(c,:);        % coords of node 'c'
    ec    = point2ElemIndex(xc,mesh);   % element id contains 'c'
    esctr = element(ec,:);              % nodes of element 'ec'
    for i=1:length(esctr)               % loop over nodes of 'ec'
        id      = esctr(i);
        x       = xc - node(id,:);
        [N,]    = getMPM2D(x,mesh.deltax,mesh.deltay);
        if nmass(id) > tol
            xc = xc + dtime*N*nmomentum(id,:)/nmass(id);
        end
    end
    particles.node(c,:) = xc;           % update pos. of particle node 'c'
end
```

6.11 Implementation of CPDI-Poly

If the material domain is discretized by Voronoi cells then the number of nodes per cell varies from cell to cell. Therefore, one has to use a `cell` data structure to store the elements of the particle mesh. We use `PolyMesher` package developed by Talischi et al. (2012) to create meshes of polygonal elements as shown in Listing 6.25. Similarly, the CPDI data (nodes, function weights and gradient weights) for all particles are stored using a data structure `cell` as given in Listing 6.26. CPDI data are computed using the function given in Listing 6.27 which follows closely Eq. (3.46).

Listing 6.25 `PolyMesher` package Talischi et al. (2012) used to generate the polygonal particle mesh.

```
%%   particle distribution from a mesh
% generate Voronoi mesh where CircleDomain.m defines the geometry
% which is in this case a circle
NElem = 20;
[Node,Element,Supp,Load,P]=PolyMesher(@CircleDomain,NElem,100);
% store the particle mesh into a structure for convenience
particles.node      = Node;
particles.elem      = Element;
% particles.elem{p} -> nodes or vertices of particle 'p'
```

Listing 6.26 Matlab data structures to store particle CPDI-poly data.

```
nodeid  = cell(pCount,1);                % nodes affect particle 'p'
funcW   = cell(pCount,1);                % function weights of 'p'
gradW   = cell(pCount,2);                % gradient weights of 'p',
```

Listing 6.27 Matlab function to determine particle CPDI data.

```
function data = getCPDIPolygonData(pid,particle,mesh)
% Inputs:
% pid:       particle index
% particle:  particle mesh
% mesh:      background mesh/grid
% Output:
% data.wf:     function weights
% data.wf:     gradient weights
% data.nodes: indices of nodes influencing particle "pid"
nodeIds   = particle.elem{pid};       % corners of the particle domain
corners   = particle.node(nodeIds,:); % coords of corners
nodeCount = length(nodeIds);
xp        = mean(corners);
wf        = zeros(nodeCount+1,1);     % function weights
wg        = zeros(nodeCount+1,2);     % gradient weights
elems     = zeros(nodeCount+1,1);     % elements of particle nodes
A = 0;
for i = 1:nodeCount
  j       = rem(i,nodeCount) + 1;
  x1      = corners(i,1); y1   = corners(i,2);
  x2      = corners(j,1); y2   = corners(j,2);
  x3      = xp(1);        y3   = xp(2);
  area    = 0.5*( x21*y31 - y21*x31 );  % area of sub-triangle
  A       = A + area; area3 = area/3;
  wf(i)   = wf(i)   + area3;
  wf(j)   = wf(j)   + area3;
  wf(end) = wf(end) + area3;
  wg(i,1) = wg(i,1) + (0.5)*(y2-y3); % x-derivative
  wg(i,2) = wg(i,2) + (0.5)*(x3-x2); % y-derivative
  wg(j,1) = wg(j,1) + (0.5)*(y3-y1);
  wg(j,2) = wg(j,2) + (0.5)*(x1-x3);
  wg(end,1) = wg(end,1) + (0.5)*(y1-y2);
  wg(end,2) = wg(end,2) + (0.5)*(x2-x1);
  elems(i)  = point2ElemIndex([x1 y1],mesh);
end
elems(end) = point2ElemIndex(xp,mesh);
data.nodes = unique(mesh.element(elems,:));
data.wf    = wf/A;
data.wg    = wg/A;
```

6.12 Visualization Toolkit (VTK)

In this section we will show how to use Paraview or VisIt to visualize MPM simulation results. For each time step, a VTP file, of which an example is given in Listing 6.28, is created. The number of particles is specified in Line 3. Lines 5–12 are for the particles coordinates. Stresses and pressure or any other particle fields are specified in Lines 13–29. Here, three fields (two scalar fields and one vector field) are demonstrated. This file was created using the function `VTKParticles`. Interested readers are

referred to the source code for details. Usually this function writes the following particle data to file, per time step: position, stresses, plastic strain, velocity. Other quantities can be added if needed.

For Paraview, a PVD file which collects all the VTP files is needed. Listing 6.29 is an example. For VisIt, one has to number the VTP files properly e.g. *mpm2DTwoDisk100.vtp*, *mpm2DTwoDisk200.vtp*, *mpm2DTwoDisk300.vtp* and so on. Additionally the background grid is also exported to a VTU file. Both Paraview and VisIt have user friendly GUIs we therefore refer to their websites for documentation.

Listing 6.28 Particle information stored in a VTP file.

```
<VTKFile type="PolyData" version="0.1" >
<PolyData>
<Piece NumberOfPoints="950" NumberOfVerts="0" NumberOfLines="0"
NumberOfStrips=" 0" Number OfPolys=" 0">
<Points>
<DataArray type="Float64" NumberOfComponents="3" format="ascii" >
0.274196 0.098072 0.000000
0.140125 0.369835 0.000000
0.449611 0.247533 0.000000
...
</DataArray>
</Points>
<PointData Scalars="pressure" Vectors="sigma">
<DataArray type="Float64" Name="pressure" format="ascii">
2.0
1.0
...
</DataArray>
<DataArray type="Float64" Name="temparature" format="ascii">
0.0
5.0
...
</DataArray>
<DataArray type="Float64" Name="str" NumberOfComponents="3" format="ascii">
0.0 0.0 0.0
0.0 0.0 0.0
...
</DataArray>
</PointData>
</Piece>
</PolyData>
</VTKFile>
```

Listing 6.29 VTP files are collected in a PVD file.

```
VTKFile byte_order="LittleEndian" type="Collection" version="0.1">
Collection>
DataSet file='mpm2DTwoDisks100.vtp' groups='' part='0' timestep='100'/>
DataSet file='mpm2DTwoDisks200.vtp' groups='' part='0' timestep='200'/>
..
DataSet file='mpm2DTwoDisks370.vtp' groups='' part='0' timestep='370'/>
/Collection>
/VTKFile>
```

6.13 Some Efficiency Improvements

As MPM calculations are slower than FE ones for moderate deformation problems, many attempts have been done to improve the efficiency of the MPM. In this section, we present some ideas such as active elements/nodes, coupling the MPM with FEM and use of dynamically allocated memory etc.

In the MPM the computational grid must be chosen to be sufficiently large to cover the trajectory of the moving particles. Therefore, at any instance, there are elements (or cells) which do not contain any particles. It is more efficient not to loop over these *inactive elements*.[2] Associated to the inactive elements are *inactive nodes* which are not needed to be included in the momenta equation. We refer to Fig. 6.6 for an illustration of these concepts. Listing 6.30 shows some Matlab codes to build the active nodes and elements.

Listing 6.30 Active nodes and elements.

```
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;
end
% active elements
activeElems = unique(pElems);
% active nodes
activeNodes = unique(element(activeElements,:));

% loop over elements is now loop over activeElements
% update nodal momenta for only active nodes
nmomentum(activeNodes,:) += niforce(activeNodes,:)*dtime;
```

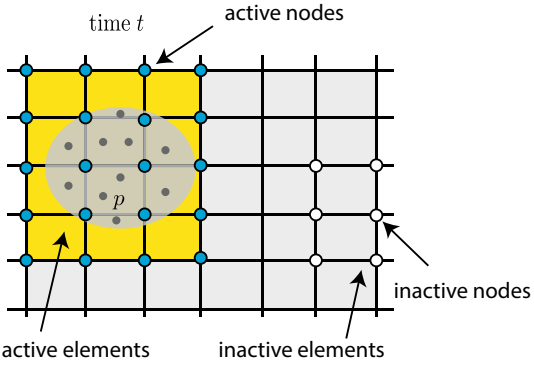


Fig. 6.6 Active and inactive elements and nodes in MPM. Note that the status of nodes and elements is changing in time as the material deforms i.e., material points move across the mesh

[2] Elements are inactive for only the time step under consideration as in the next step particles can move to these inactive elements and they become active.

Our implementation (and most of MPM implementations) described so far can be referred to as using a *static mesh* as the memory was allocated for all the nodes and cells (active and inactive). In Shin et al. (2010), the concept of *dynamic mesh* was proposed to reduce the high memory storage in the conventional MPM. However, using a dynamic mesh can make the parallelization of a MPM code more difficult.

Another attempt to reduce the extra cost of MPM (due to mapping between grid nodes and particles) is to use a combined MPM-FEM strategy in which the MPM is used for domains of intensive deformation and FEM is reserved for regions of benign deformation (Zhang et al. 2006). These researchers showed that the MPM-FEM is three times faster than FEM. Another improvement was made by Buzzi et al. (2008) by computing the shape functions and derivatives once and store them. This is possible because during a Lagrangian phase of one time step, the relative positions of the particles with respect to the grid nodes do not change and thus the shape functions and derivatives are constant. However, this trick requires more storages.

6.14 More Improvements Using MEX Files

MEX—stands for Matlab Executable—functions provide a simple means to speed up Matlab codes by allowing ones to call Fortran and/or C/C++ functions inside Matlab. In this section we present how one can use MEX files (written in C) to significantly improve the performance of the MPM Matlab code. Another advantage is that large pre-existing C and Fortran programs can be called from MATLAB without having to be rewritten as M-files.

We created a folder **mex** within the parent folder containing the MPM source code and all the MEX files are located in this folder. Before using them, one has to compile them, by in the Matlab Command Window running the compile script **compile**. Note that our MEX files are manually created while one could use the Matlab Coder, http://www.mathworks.co.uk/products/matlab-coder/, to automatically generate MEX files from Matlab codes. A good tutorial to the subject can be found at http://www.nr.com/nr3_matlab.html.

MEX-functions are best suited to substitute some bottleneck M-functions in an application. If you replace all functions in an application with MEX, you might port the application entirely to C which is a much harder task to accomplish. For this reason, the pre- and post- processing are still performed using Matlab codes while the major parts of the processing is realized via MEX functions. Listing 6.31 shows such an implementation where MEX functions are used in lines 3 and 14. For simplicity, treatment of boundary conditions was omitted. We refer to file **mpm2DTwoDisksNew.m** for the entire source code. In this implementation the particles are grouped into *bodies* to facilitate handling of multiple materials and/or contact, see Chap. 10 for details. One should remember that in MEX files, a two dimensional $m \times n$ Matlab matrix $\mathbf{A}$ is stored as an one dimensional array stored in a column-major format. That is $A(i+1, j+1)$ in Matlab becomes $A[i + j * m]$ in MEX files where $0 \leq i \leq m-1$ and $0 \leq j \leq n-1$.

Listing 6.31 Processing step of a hybrid MPM code using MEX functions.

```
while ( t < time )
  % particles to nodes, MEX function implemented in mex/ParticlesToNodes.c
  [nmass,nmomentum,niforce] = ParticlesToNodes(bodies,mesh);
  % update nodal momenta (using Matlab)
  activeNodes=[bodies{1}.nodes; bodies{2}.nodes];
  nmomentum(activeNodes,:) += niforce(activeNodes,:)*dtime;
  % compute nodal velocities and accelerations
  nvelo(activeNodes,1) = nmomentum(activeNodes,1)./nmass(activeNodes);
  nvelo(activeNodes,2) = nmomentum(activeNodes,2)./nmass(activeNodes);
  nacce(activeNodes,1) = niforce(activeNodes,1)./nmass(activeNodes);
  nacce(activeNodes,2) = niforce(activeNodes,2)./nmass(activeNodes);
  % update particle positions, velocities, stresses, volumes, deformation
  % using MEX function in mex/UpdateParticles.c
  UpdateParticles(bodies,mesh,nvelo,nacce,dtime);
  % compute kinetic and strain energy again with Matlab
  k = 0; u = 0;
  for ib=1:bodyCount
    body      = bodies{ib};
    for p=1:length(body.mass)
      vp    = body.velo(p,:);
      k = k + 0.5*(vp(1)^2+vp(2)^2)*body.mass(p);
      u = u + 0.5*body.volume(p)*body.stress(p,:)*body.strain(p,:)';
    end
  end
  % update the element particle list (Matlab)
  bodies = findActiveElemsAndNodes(bodies,mesh);
  % advance to the next time step
  t      = t + dtime; istep = istep + 1;
end
```

From our own experiences calling Matlab functions in C-MEX files (using MexCallMatlab) is inefficient. Due to that reason, stress update routines should be rewritten in C as well.

Remark 36 We have to admit using MEX files is not an elegant solution and this is the problem of using two languages (Matlab and C). The new dynamic programming language `Julia` can be a more elegant and efficient solution, see Appendix F.

6.15 Examples

In this section, we present some one and two dimensional simulations to verify the implementation and to demonstrate the performance of the MPM. They include one dimensional problems (Sect. 6.15.1), a two dimensional elastic collision problem (Sect. 6.15.2), a high velocity impact problem (Sect. 6.15.3), a compliant cantilever beam (Sect. 6.15.4) and lateral compression of thin-walled tubes (Sect. 6.15.5).

We iterate that these simulations aim to illustrate the essence of the MPM and our Matlab code, rather than to demonstrate the modeling capabilities of the MPM. The aim of the last example (lateral compression of thin-walled tubes) is to present how to verify numerical simulations against experiments. Advanced problems (e.g. frictional contact, fracture, fluid-structure interaction etc.) are presented in the coming chapters.

6.15.1 One Dimensional Examples

Single-material-point vibration. As the simplest MPM example, let us consider the vibration of a single material point as shown in Fig. 6.7. 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. This problem was studied in Bardenhagen (2002), Buzzi et al. (2008).

The exact solution is given by

$$v(t) = v_0 \cos(\omega t), \quad \omega = \frac{1}{L}\sqrt{E/\rho} \tag{6.13}$$

for the velocity and

$$x(t) = x_0 \exp\left[\frac{v_0}{Lw}\sin(\omega t)\right] \tag{6.14}$$

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 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 file **example1D/mpm1D.m**. Note that this implementation adopts the double mapping technique presented in Sect. 2.5.4. A time step of 0.001 was used.

A good agreement between the MPM and the exact solutions can be observed (Fig. 6.8). To check if energy is conserved, kinetic, strain and total energies are plotted in Fig. 6.9. The strain and kinetic energy are computed as

$$U = \frac{1}{2}\sigma_p \epsilon_p V_p, \quad K = \frac{1}{2}v_p^2 M_p \tag{6.15}$$

Figure 6.10 shows the result obtained with the standard USL formulation and it is consistent with the result given by Buzzi et al. (2008). It was based on this that Bardenhagen (2002) concluded that the USL is strictly dissipative. Here, we demonstrated that only the standard USL formulation is dissipative and the double mapping USL is not.

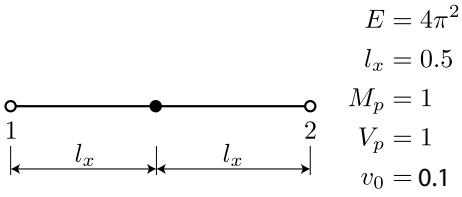


Fig. 6.7 Vibration of a single material point (solid point). Any set of consistent units suffices, see Sect. C.5

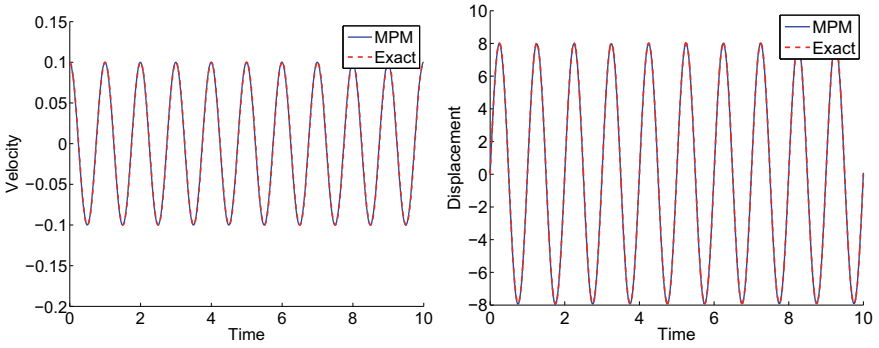


Fig. 6.8 Vibration of a single material point: USL and double mapping algorithm ($\Delta t = 0.001$ s)

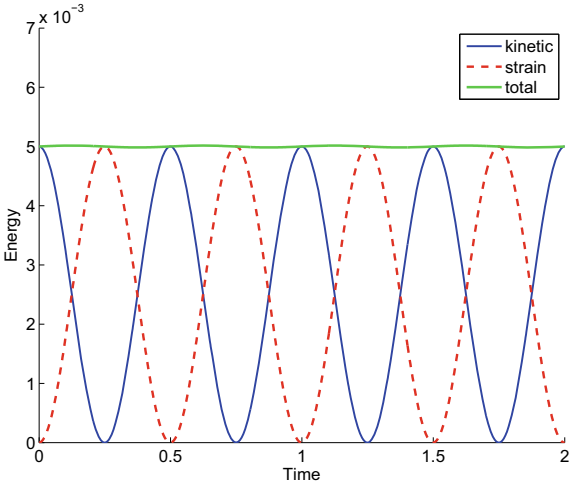


Fig. 6.9 Vibration of a single material point: kinetic, strain and total energies. Formulation: USL and double mapping algorithm ($\Delta t = 0.001$ s). The USF also gives the same result

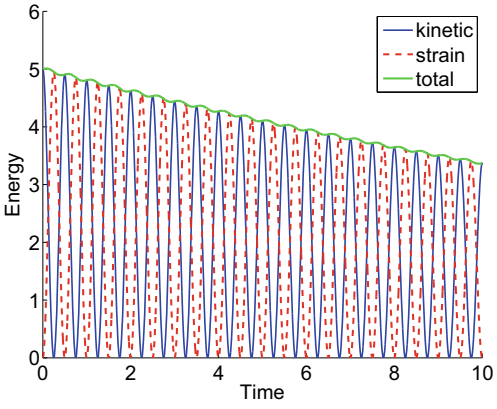


Fig. 6.10 Vibration of a single material point: kinetic, strain and total energies. Formulation: standard USL ($\Delta t = 0.001$ s)

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 $L = 25$. The case considered here is the analogy to the single-material-point problem. One end ($x = 0$) of the bar is fixed, and the other ($x = L$) is free.

Exact solutions for mode n are (refer to e.g. Bardenhagen (2002))

$$v(x, t) = v_0 \cos(\omega_n t) \sin(\beta_n x) \tag{6.16}$$

$$u(x, t) = \frac{v_0}{\omega_n} \sin(\omega_n t) \sin(\beta_n x) \tag{6.17}$$

where $\omega_n = \beta_n \sqrt{E/\rho}$ and $\beta_n = \frac{2n-1}{2} \frac{\pi}{L}$. The period of vibration is $2\pi/\omega_1 = 10$. In the computations, $v_0 = 0.1$ was used.

The initial velocity is given by

$$v(x, 0) = v_0 \sin(\beta_n x) \tag{6.18}$$

The grid consists of 13 two-noded line elements (14 grid nodes) and 13 material points (i.e., one particle per element) placed at the center of the elements are used. The Matlab M-file of this problem is **example1D/mpm1DVibrationBar.m**. The time increment is chosen as $\Delta t = 0.1 \Delta x/c$ where Δx denotes the nodal spacing. We consider two modes–mode 1 ($n = 1$) and mode 10 ($n = 10$) and for both cases, the quantity of interest used to compare the numerical and exact solution is the center of mass velocities which are given by

$$v_{\text{cm}}^{\text{exa}}(t) = \frac{v_0}{\beta_n L} \cos(\omega_n t); \quad v_{\text{cm}}^{\text{num}}(t) = \frac{\sum v_p(t) m_p}{\sum m_p} \tag{6.19}$$

for the exact solution and the MPM solution, respectively.

For mode 1 the MPM solution are in good agreement with the exact solution (Fig. 6.11) and the algorithm conserved the energy (Fig. 6.12). The same observation was made by Bardenhagen (2002). To demonstrate that PIC results in a big numerical dissipation and FLIP does not, we solved this example with two values of $\alpha \in \{0, 1\}$ (Fig. 6.13). In this chapter, $\alpha = 1$ was used unless otherwise stated. Recall that α is the PIC-FLIP blending factor used in the particle velocity update, cf. Eq. (2.54).

To further test the algorithms, the tenth mode ($n = 10$) is investigated. The center of mass velocities obtained with two mesh densities (one particle per element) are depicted in Fig. 6.14. Figure 6.15 depicts the energies for both standard/double-mapping USL formulations.

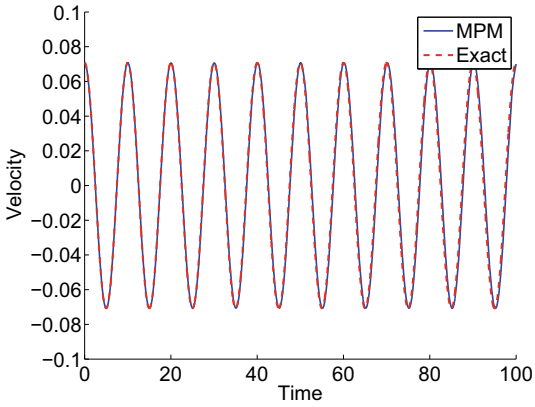


Fig. 6.11 Vibration of a continuum bar: comparison of the MPM velocity solution and the exact solution for mode 1

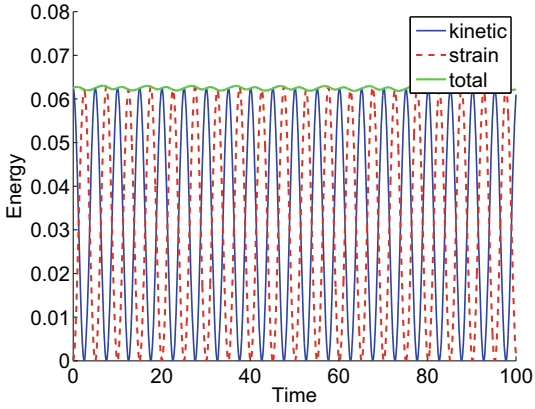


Fig. 6.12 Vibration of a continuum bar: kinetic, strain and total energies for mode 1

6.15.2 Impact of Two Elastic Disks

As the first 2D MPM simulation, we advocate the collision problem proposed by Sulsky et al. (1994). This problem involves the impact of two identical elastic disks which move in opposite direction towards each other (Fig. 6.16). This is an ideal test case for a 2D MPM code because of the following reasons. First, no boundary conditions and external force are involved. Second, no inelastic material is present. Third, before collision, the two disks are in a rigid body motion and thus the stress and strain are identically zero (if they are not, then the calculation of the shape function derivatives should be incorrect).

The computational domain is a 1×1 mm^2 square, which is discretized into 20×20 cells. A plane strain condition and no gravity are assumed. The mesh and the initial irregular particle distribution are shown in Fig. 6.17. 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. Note that other authors (e.g. Sulsky et al. 1994; Buzzi

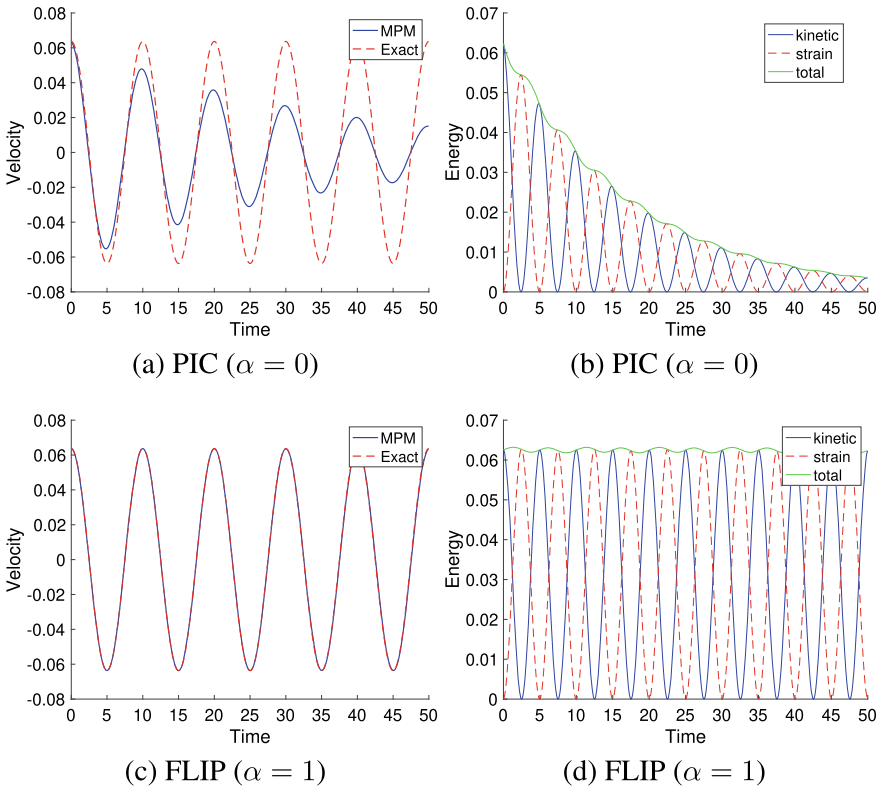


Fig. 6.13 Vibration of a bar: mode 1 case: PIC versus FLIP with USL algorithm used (16 cells, PPC = 2). For this mode, USL and MUSL perform similarly (de Vaucorbeil et al. 2020)

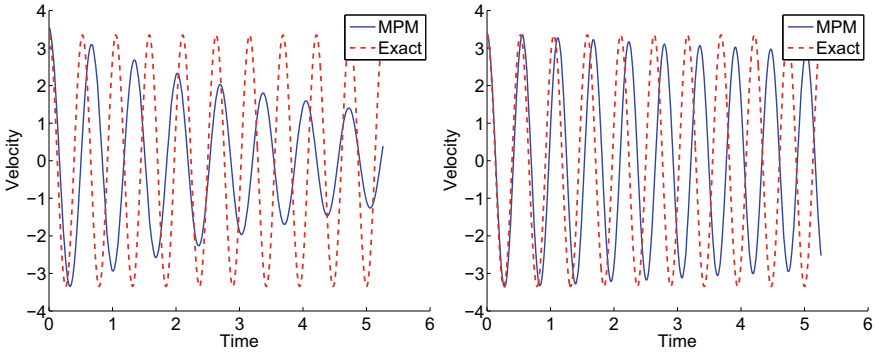
et al. 2008) used a regular particle distribution. Initial condition for this problem is the initial velocities of the particles, $\mathbf{v}_p = \mathbf{v}$ for lower-left particles and $\mathbf{v}_p = -\mathbf{v}$ for upper-right particles. There is no boundary conditions in this problem since the simulation stops before the particles move out of the computational box after impact. The M-file of this simulation is **example2D/mpm/mpm2DTwoDisks.m**.

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

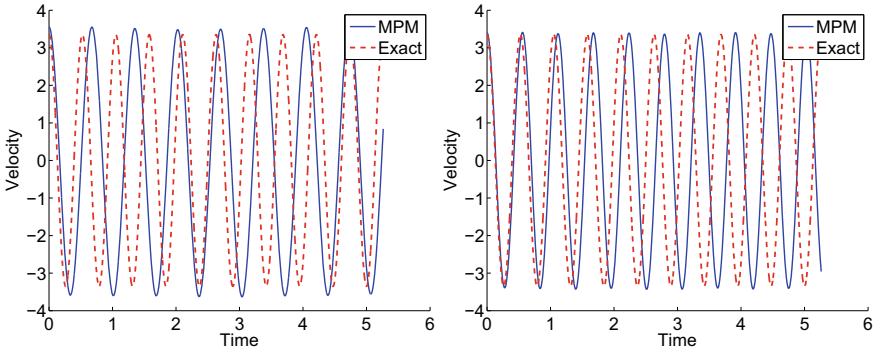
$$U = \sum_{p=1}^{n_p} u_p V_p, \quad K = \frac{1}{2} \sum_{p=1}^{n_p} \mathbf{v}_p \cdot \mathbf{v}_p M_p \tag{6.20}$$

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

$$u_p = \frac{1}{4\mu} \left[\frac{\kappa + 1}{4} (\sigma_{p,xx}^2 + \sigma_{p,yy}^2) - 2(\sigma_{p,xx}\sigma_{p,yy} - \sigma_{p,xy}^2) \right] \tag{6.21}$$

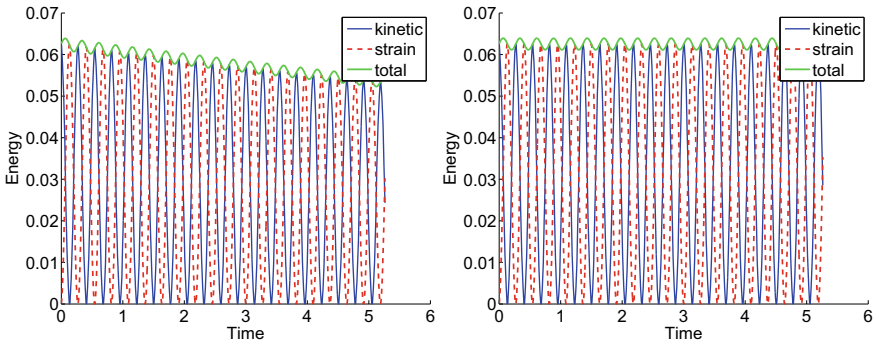


(a) USL,25 elements (b) USL,50 elements



(c) MUSL,25 elements (d) MUSL,50 elements

Fig. 6.14 Vibration of a continuum bar (mode 10): center of mass velocities (de Vaucorbeil et al. 2020)



(a) USL,cutoff (b) USL,double mapping

Fig. 6.15 Vibration of a continuum bar (mode 10): evolution of energies in time according to different USL formulations. The grid consists of 64 elements with one particle per element (de Vaucorbeil et al. 2020)

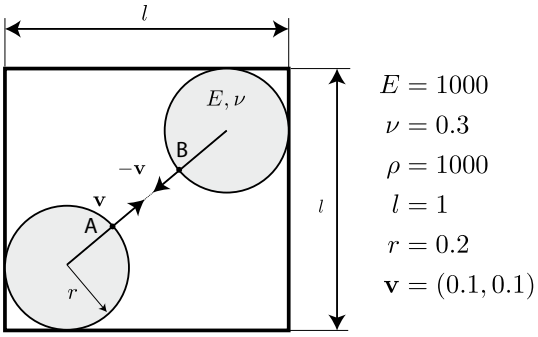


Fig. 6.16 Impact of two elastic bodies: problem statement. The computational domain is a unit square and the radius of the disks is 0.2 mm (de Vaucorbeil et al. 2020)

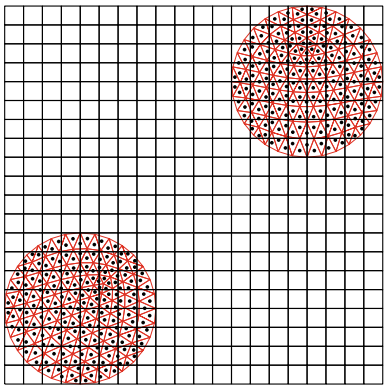


Fig. 6.17 Impact of two elastic bodies: Eulerian mesh and initial particle distribution

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

Also, only the algorithm using a cutoff value to detect small nodal masses was presented. A time step of 0.001 s was used.

The movement of two disks is given in Fig. 6.18. The collision occurs in a physically realistic fashion, although no contact law has been specified. Figure 6.19 plots the evolution of the kinetic, strain and total energy. All of the initial energy is kinetic energy. The initial kinetic energy is $K = 2 \times (0.5 \times (v^2 + v^2) \times \rho \times \pi \times r^2) = 2.513$. The kinetic energy decreases during impact and is then mostly recovered after separation. The strain energy reaches its maximum value at the point of maximum deformation during impact and then decreases to a value associated with free vibration of the disk. The result is identical to the ones reported in Sulsky et al. (1994), Buzzi et al. (2008), Coetzee (2003) which confirms the implementation. However the contact did occur earlier than it should have been. The correct contact time is $t = AB/(2\sqrt{2}v) = 1.5858$ s where $v = 0.1$. In the simulation contact happened at $t = 1.3$ s. This result is expected as the contact is resolved at the grid nodes not the particles i.e., contact is detected even when the particles of the two bodies are one cell separate. The situation is more severe if a C^{p-1} smooth basis function is used as in GIMP or B-splines MPM where the nodal support is larger. A simple mesh refinement can improve the result or a contact algorithm which is based on particle

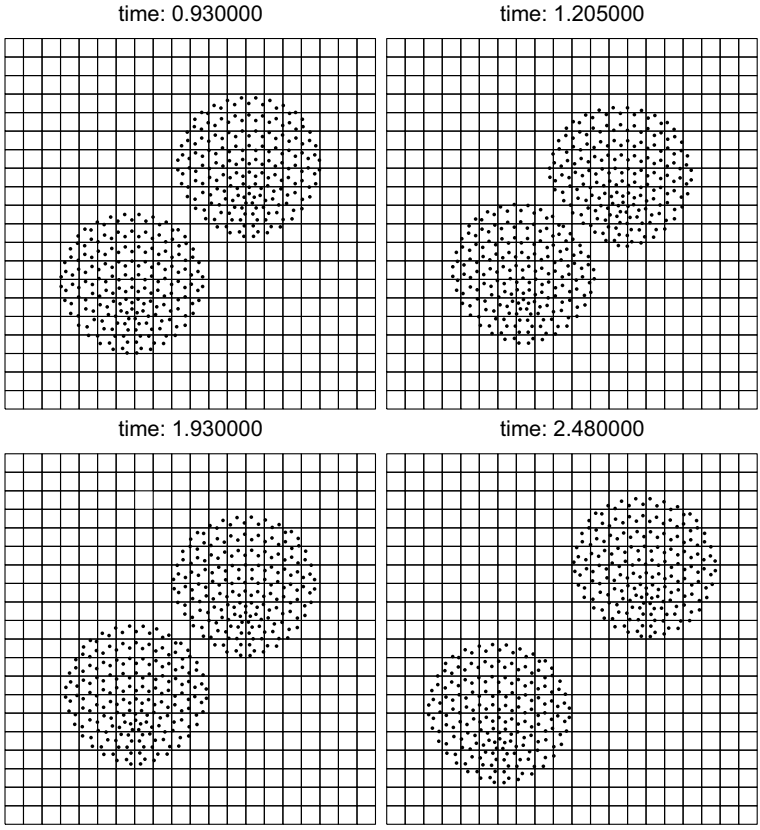


Fig. 6.18 Impact of two elastic bodies: time snapshots of the bodies. Top figures: twp bodies move towards each other and collide. Bottom figures: they bounce back and move far from each other. These figures were created in Matlab using the *scatter* command

distance (Lemiale et al. 2010) or surface stress (Bardenhagen et al. 2001) should be used. We refer to Chap. 8 for more details on contact in the MPM.

Verification of CPDI-Poly. As a verification of the CPDI-Poly, we re-consider again the Sulsky's two disk problem. The disks are discretized by 500 Voronoi cells (Fig. 6.20). Results given in Fig. 6.21 verify the formulation and its implementation. The M-file for this example is **example2D/cpdi/ cpdiPolygonTwoDisks.m**.

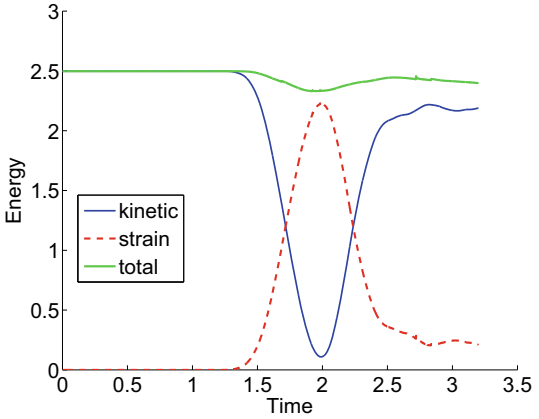


Fig. 6.19 Impact of two elastic bodies: evolution of strain, kinetic and total energies in time

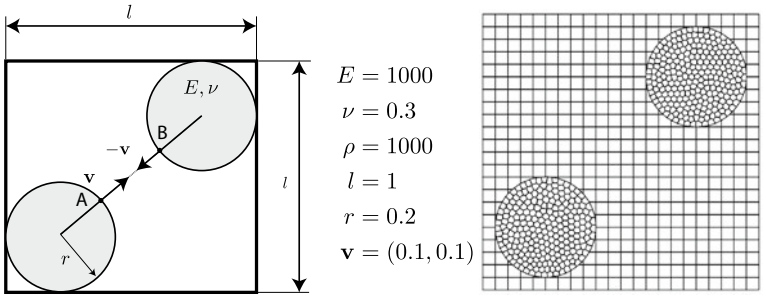


Fig. 6.20 Impact of two elastic bodies: problem description and MPM setup. There are 500 particles or 500 Voronoi cells

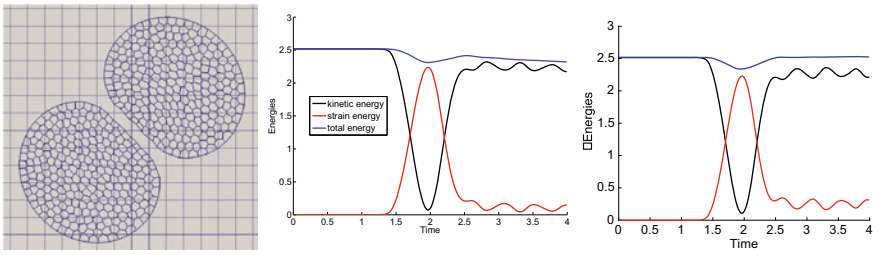


Fig. 6.21 Impact of two elastic bodies: simulation snapshot and evolution of energies. On the right, the double mapping method of Sulsky was used to have better energy conservation

6.15.3 High Velocity Impact

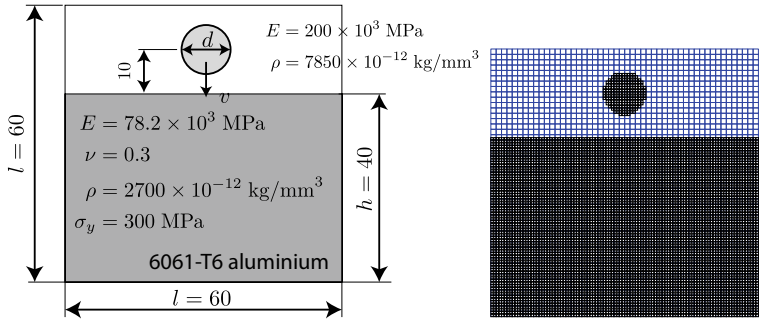
We consider a high velocity impact problem in which an AISI 52-100 chromium steel disk impacting an elastic-perfectly plastic target of 6061-T6 aluminum under plane strain conditions (Fig. 6.22a). This problem, presented in Sulsky et al. (1995a), Coetzee (2003), was inspired by the experiments carried out by Trucano and Grady (1985). The steel disk is assumed to be linear elastic and the aluminum target to be elastic-perfect plastic obeying a von Mises yield criterion. The boundary of the computational domain is fixed.

We used a time step of $\Delta t = 0.05\sqrt{E_s/\rho_s} = 1.18 \times 10^{-8}$ s where subscript s denotes the steel disk. The total time t_f is about 40 μs and thus there are about 40 000 time steps. The initial particle distribution and the grid is given in Fig. 6.22b. We used the MUSL algorithm as the USL with cut-off value for small nodal masses did not work if a proper value for the cut-off was not used; and this value is grid-dependent. The M-file is **example2D/mpm/mpm2DSteelDiskImpact.m**.

The von Mises contour plot at different time frames is depicted in Fig. 6.23. As can be seen the steel disk does not deform which is consistent with experiment carried out by Trucano and Grady (1985). The penetration depth is in good agreement with the MPM result reported in Coetzee (2003).

6.15.4 Large Deformation Vibration of a Compliant Cantilever Beam

We now consider yet another problem that the FEM can solve easily but many MPM variants struggle with. This problem is the vibration of a cantilever beam which is soft and subjected to a large gravity force (Fig. 6.24). Sadeghirad et al. (2011) presented



(a) Problem setup (b) Grid and particles

Fig. 6.22 Impact of a steel disk into an aluminum target. The disk has an initial velocity of 1160 m/s and the disk's diameter is 9.53 mm. The boundary represents the computational domain (50 × 50 elements). There are 6 700 particles for the aluminum target and 204 particles for the steel disk

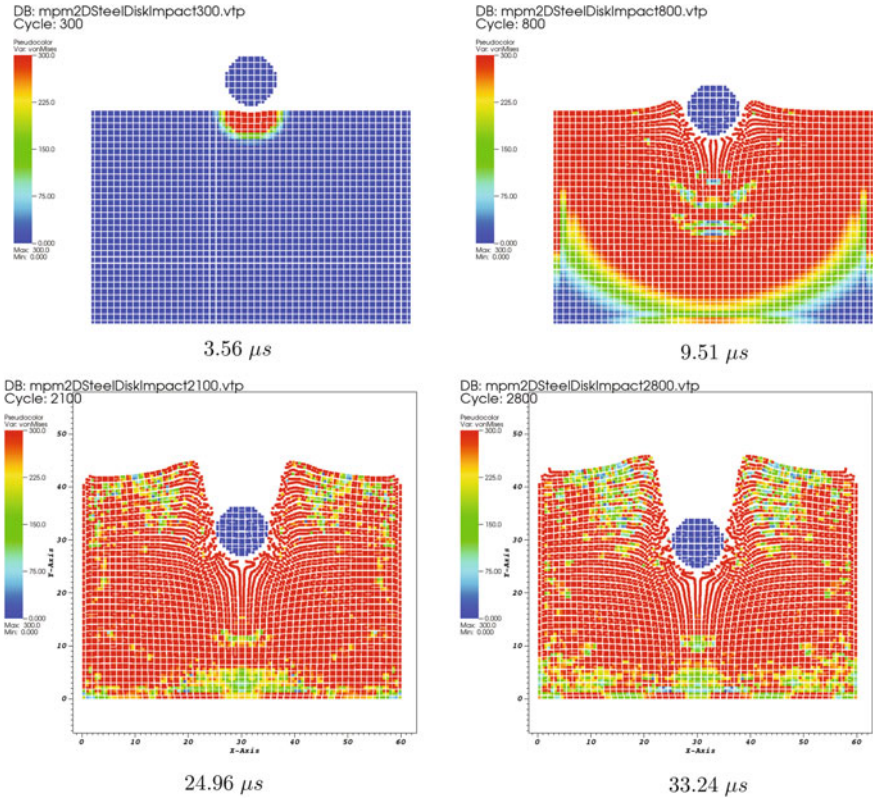


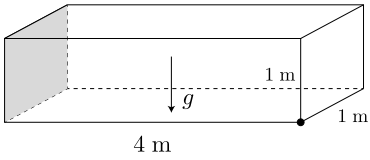
Fig. 6.23 Impact of a steel disk into an aluminum target: von Mises stress distribution at different time instances. The results are post-processed in VisIt

this example for the first time in the MPM literature but similar problems appeared earlier in the SPH literature. We refer to Sect. D.4.3 for the accurate finite element solutions obtained with a very coarse mesh.

The cantilver beam is made of a hyperelastic material, which is modeled by a Neo-Hookean constitutive model described in Sect. 4.2. At $t = 0$ s a large gravity of magnitude $g = 10\,\mathrm{m/s^2}$ is suddenly applied to the beam. And this enduces a large displacement vibration of the beam. The material data are $E = 10^6$ Pa, $\nu = 0.3$, $\rho = 1050\,\mathrm{kg/m^3}$. This example is analyzed using constant time steps of 0.002 s, which is about 0.2 h/c where h denotes the element size, and real-time simulation is $T = 3$ s.

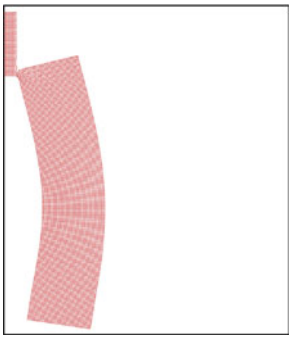
We are going to solve this problem with the ULMPM with cubic B-splines and CPDI-Q4. For that only a 2D version of the problem is considered. For the 3D version, we solve it using the CPDI-Tet4 formulation and the polyhedral CPDI.

On Fig. 6.25 we present two solutions obtained with the ULMPM: one with cubic B-splines weighting functions and one with the CPDI-Q4 weighting function. There

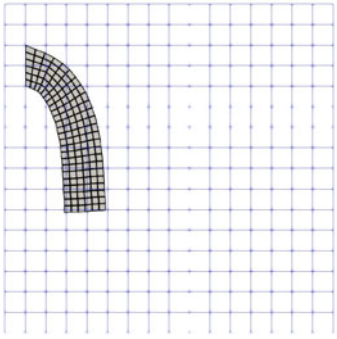


$E = 10^6$ Pa
$\nu = 0.3$
$\rho = 1050$ kg/m^3
$g = 10$ m/s^2

Fig. 6.24 Vibration of a compliant beam: the left face (shaded) is fixed. The dot denotes the point of which vertical displacement is tracked in time (Nguyen et al. 2017)



(a) ULMPM, FLIP, 3rd B-splines



(b) ULMPM, FLIP, CPDI-Q4

Fig. 6.25 Vibration of a compliant beam: many ULMPM variants are prone to numerical fracture **a** whereas ULMPM with CPDI is not (**b**). Certainly, the TLMPM does not suffer from numerical fracture and thus works well for this problem (not shown here for brevity)

we observe that due to the large tensile stress on the top surface of the beam close to the fixed support, the ULMPM in which the particle domains are not tracked exhibits numerical fracture: the beam is suddenly broken into two pieces. GIMP is also unstable for this problem, even though the problem is not as severe as the one given in Fig. 6.25a (Sadeghirad et al. 2011).

For the results given in Fig. 6.25b the background grid consists of 16×16 elements while the beam is discretized with 26×4 squares. At $t = 0$ s, each grid cell (within the solid domain) is populated with 3×3 CPDI-Q4 particles. The M-file of this simulation is **example2D/mpm/cpdiQ4VibratingBeam.m**.

CPDI-Tet4 formulation. First, we validate the CPDI-Tet4 formulation. A reference solution is constructed using the FEM with a refined mesh of about 1000 linear tetrahedra. For the MPM simulations, two background grids of $16 \times 16 \times 2$ and $64 \times 64 \times 2$ cells are considered with four particle meshes in which the coarsest contains 82 particles and the finest has 1000 particles (Fig. 6.26). The time evolution of the vertical displacement of the tracked point from various analyses, shown in Fig. 6.27, indicates convergence towards the FEM solution. With a coarse background grid more particles are needed for accurate results whereas less particles are required

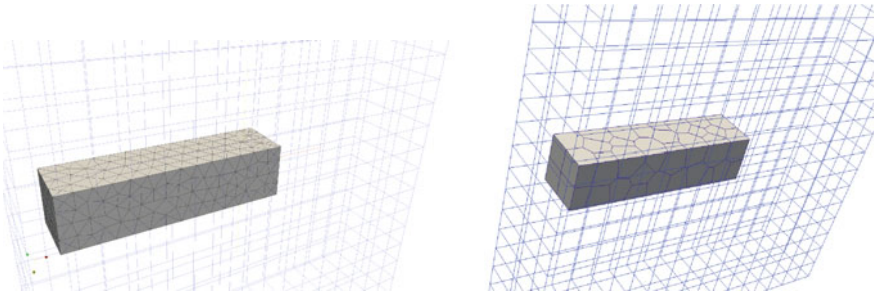


Fig. 6.26 Vibration of a compliant beam (Nguyen et al. 2017): Background grid and the material mesh. The computational domain is $8 \times 8 \times 2$ m^3 discretized by $16 \times 16 \times 2$ cells. The cantilever is meshed by 1000 tetrahedra on the left (created using Gmsh) and by 50 polyhedra on the right (using Neper and a centroidal Voronoi tessellation, CVT, Du et al. (1999))

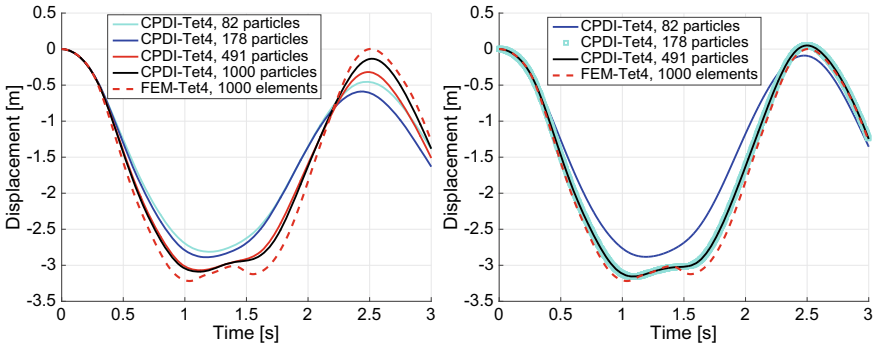


Fig. 6.27 Vibration of a compliant beam: FEM versus CPDI-Tet4. On the left, the background grid contains $16 \times 16 \times 2$ (minimum grid spacing is 0.5 m) cells whereas on the right it consists of $64 \times 64 \times 2$ cells (minimum grid spacing is 0.125 m). The smallest element size in the FEM mesh is 0.0534 m (Nguyen et al. 2017)

when a fine background grid is employed. Since the particle mesh was generated independently of the background grid, it is difficult to find an optimal relation between the background mesh and the particles as it is the case for the standard MPM or uGIMP where the rule of 3 particle (in each direction) per cell is usually encouraged (Sulsky et al. 1995a). The M-file of this simulation is **example3D/cpdi/cpdiBeamTet4.m**.

Polyhedral CPDI. Next, we turn attention to the polyhedral CPDI. To reduce the computational cost, a grid of $32 \times 32 \times 2$ was adopted and two particle meshes of 82 and 500 polyhedra, obtained using a centroidal Voronoi tessellation (Du et al. 1999), were considered. The result given in Fig. 6.28 shows the convergence of the CPDI results towards the FEM one. And Fig. 6.29 presents some simulation snapshots showing the deformed beam. Finally, we consider a regular Voronoi tessellation where the particles are not as regular as those obtained by a CVT. These

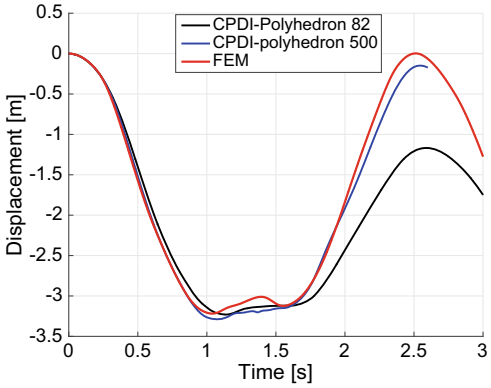


Fig. 6.28 Vibration of a compliant beam: FEM versus polyhedral CPDI. The background consists of $32 \times 32 \times 2$ cells (minimum grid spacing is 0.25 m). The smallest element size in the FEM mesh is 0.0534 m (Nguyen et al. 2017)

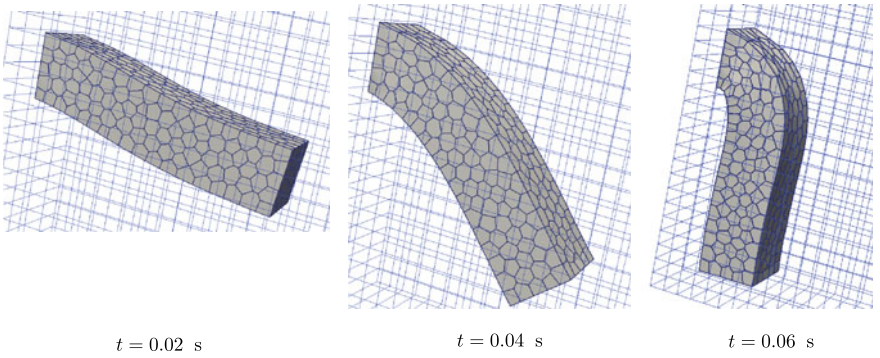


Fig. 6.29 Vibration of a compliant beam (Nguyen et al. 2017): some simulation snapshots with polyhedral CPDI (500 polyhedra)

particles can come directly from the microstructure of the material or they are the deformed CVT particles. The result shown in Fig. 6.30 indicates that the polyhedral CPDI still performs well with regular Voronoi cells. The M-file of this simulation is **example2D/mpm/cpdiBeamPolyhedra.m**.

6.15.5 Lateral Compression of Thin-Walled Tubes

To evaluate the application of the MPM for the design of energy absorption systems, Sinaie et al. (2018) carried out a series of 3D simulations on thin-walled tubes made of mild steel. Herein, we present one of their simulations to validate the MPM against experiments. The simulation is a quasi-static compression test of thin-walled steel

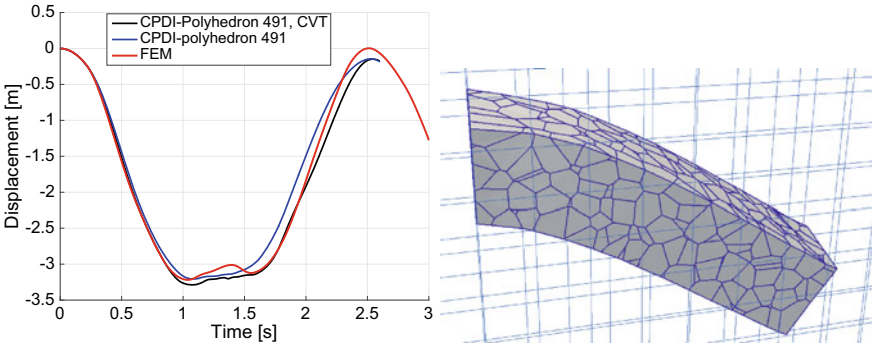


Fig. 6.30 Vibration of a compliant beam represented by a regular Voronoi tessellation where the particle domains are not as regular as those obtained by a CVT (Nguyen et al. 2017)

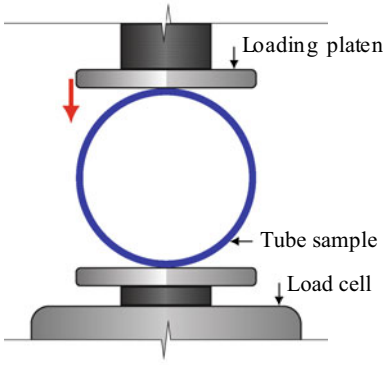


Fig. 6.31 Schematic of quasi-static compression tests by Xiang et al. (2017). Inner diameter is 47.9 mm and thickness is 1.48 mm (Sinaie et al. 2018)

tubes carried out by Xiang et al. (2017), see Fig. 6.31 for the set up. Along the way, we provide details on the procedure of validation of a computational code against experiment.

The procedure is as follows

- Select a material model that best describes the behavior of the material under consideration.
- Calibrating the parameters of the selected material model using data from the experiments.
- Carry out mesh-convergence analysis to ensure the numerical solution is indeed trust worthy. Then, record the minimum number of elements (and particles for particle methods) that is sufficient to get convergent result and use that for further simulations.

The second item demands elaboration. First, it is recommended that experimentalists provide all information required to conduct simulations. Second, material parameters

Table 6.1 Material properties of the quasi-static test

Variable	Meaning	Value
ρ	Density	7800 kg/m^3
E	Elastic modulus	210.0 GPa
ν	Poisson's ratio	0.3
A	Yield stress	310.0 MPa
b	Hardening parameter	4.0
B	Hardening parameter	150.0 MPa

calibration should be done using simple tests e.g. tensile/compressive tests and not the test that the code is trying to reproduce.

Material model calibration. Herein we use a J2 plasticity model with linear isotropic hardening i.e., $\sigma_f(\varepsilon_p) = A + B\left(\varepsilon_p\right)^n$. We need to find values for E, ν, ρ, A, B and n using the tensile test result provided by Xiang et al. (2017). Poisson's ratio ν and density ρ of mild steel are set to their typical values reported in the literature. To get other material parameters, we carry out a simple tension test with one grid cell and 9 material points (3D) using some trial values for E, A, B and n. The stress at one material point is compared with the experiment. The experimental result is usually presented in the form of a graph, which can be digitalized using softwares such as `Plot Digitizer`.[3] After a few trials, the material parameters that yielded matched results are given in Table 6.1.

Mesh convergence analysis. Figure 6.32 shows the cross section of the tube, specifically focusing on element density in terms of the number of CPDI-Tet4 particles along the circumference and across the thickness of the tube. There is only one layer of particle along the length direction and all the z-components of the nodal quantities are set to zero. All contacts between the sample and the load platens (or walls) are no-slip. And thus, the inherent contact capability of the MPM was exploited. Furthermore, the load platens are modeled as rigid bodies using rigid particles. The numerical solution is indeed convergent as shown in Fig. 6.33. Then, we can focus on the mechanics, for example the deformation and plastic hinges (Fig. 6.34).

Remark 37 We believe that CPDI-Tet4 does not perform well for bending and GPIC with eight-node hexahedral elements would outperform CPDI-Tet4. In any way, problems like this that involves moderately large deformation and few contacts can be solved efficiently using the FEM. For problems involving lots of contacts see Sect. 8.4.5 or the work of Sinaie et al. (2019).

[3] http://plotdigitizer.sourceforge.net.

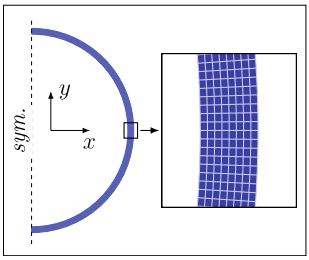


Fig. 6.32 Illustration of the simulated tube showing divisions along the circumference (360 particles) and divisions across the thickness (8 particle). Each blue box in the magnified view represents a single CPDI-Tet4 particle (Sinaie et al. 2018)

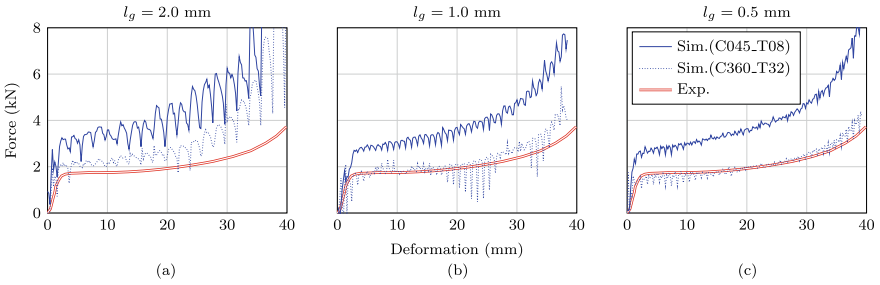


Fig. 6.33 Simulated force-deformation curves in comparison to the experimental one. Each figure corresponds with a value of l_g–the grid cell size. The number of CPDI elements along the circumference and across the thickness are the values following C and T, respectively (Sinaie et al. 2018)

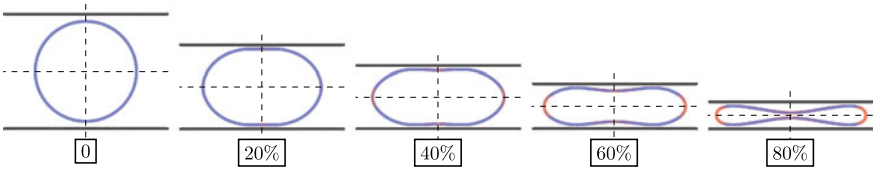


Fig. 6.34 Snapshots of the simulations carried out on thin walled tube under quasi-static compression (Sinaie et al. 2018)

We did not solve this problem using the described Matlab code as it is too slow. We used the code developed by Sinai Sinaie. But this example can be solved efficiently either using the `Julia` code discussed in Appendix F or the `C++` code `Karamelo` to be presented in the next chapter.

References

Bardenhagen, S.G.: Energy conservation error in the material point method for solid mechanics. J. Comput. Phys. **180**(1), 383–403 (2002)

Bardenhagen, S.G., Guilkey, J.E., Roessig, K.M., Brackbill, J.U., Witzel, W.M., Foster, J.C.: An improved contact algorithm for the material point method and application to stress propagation in granular material. Comput. Model. Eng. Sci. **2**(4), 509–522 (2001)

Buzzi, O., Pedroso, D.M., Giacomini, A.: Caveats on the implementation of the generalized material point method. Comput. Model. Eng. Sci. **1**(1), 1–21 (2008)

Coetzee, C.J.: The Modelling of Granular Flow Using the Particle-in-Cell Method. Ph.D. thesis, University of Stellenbosch (2003)

de Falco, C., Reali, A., Vázquez, R.: GeoPDEs: a research tool for Isogeometric analysis of PDEs. Adv. Eng. Softw. **42**(12), 1020–1034 (2011)

de Vaucorbeil, A., Nguyen, V.P., Sinaie, S., Wu, J.Y.: Chapter two—material point method after 25 years: theory, implementation, and applications. Advances in Applied Mechanics, vol. 53, pp. 185–398. Elsevier (2020)

Du, Q., Faber, V., Gunzburger, M.: Centroidal voronoi tessellations: applications and algorithms. SIAM Rev. **41**(4), 637–676 (1999)

Lemiale, V., Nairn, J., Hurmane, A.: Material point method simulation of equal channel angular pressing involving large plastic strain and contact through sharp corners. Comput. Model. Eng. Sci. **70**(1), 41–66 (2010)

Nguyen, V.P., Nguyen, C.T., Rabczuk, T., Natarajan, S.: On a family of convected particle domain interpolations in the material point method. Finite Elem. Analys. Des. **126**, 50–64 (2017)

Nguyen, V.P., Anitescu, C., Bordas, S., Rabczuk, T.: Isogeometric analysis: an overview and computer implementation aspects. Math. Comput. Simul. **117**, 89–116 (2015)

Sadeghirad, A., Brannon, R.M., Burghardt, J.: A convected particle domain interpolation technique to extend applicability of the material point method for problems involving massive deformations. Int. J. Numer. Methods Eng. **86**(12), 1435–1456 (2011)

Shin, W., Miller, G.R., Arduino, P., Mackenzie-Helnwein, P.: Dynamic meshing for mateiral point method computations. Int. J. Comput. Math. Sci. **4**(8), 379–387 (2010)

Sinaie, S., Ngo, T.D., Kashani, A., Whittaker, A.S.: Simulation of cellular structures under large deformations using the material point method. Int. J. Impact Eng. **134**, 103385 (2019)

Sinaie, S., Ngo, T.D., Nguyen, V.P., Rabczuk, T.: Validation of the material point method for the simulation of thin-walled tubes under lateral compression. Thin-Walled Struct. **130**, 32–46 (2018)

Sulsky, D., Chen, Z., Schreyer, H.L.: A particle method for history-dependent materials. Comput. Methods Appl. Mech. Eng. **5**, 179–196 (1994)

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

Talischi, C., Paulino, G.H., Pereira, A., Menezes, I.F.M.: Polymesher: a general-purpose mesh generator for polygonal elements written in Matlab. Struct. Multidiscip. Optim. **45**(3), 309–328 (2012)

Trucano, T.G., Grady, D.E.: Study of intermediate velocity penetration of steel spheres into deep aluminum targets. Technical Report, Sandia National Labs., Albuquerque, NM (USA) (1985)

Xiang, X.M., Lu, G., Li, Z.-X., Lv, Y.: Large deformation of tubes under oblique lateral crushing. Int. J. Impact Eng. (2017)

Zhang, X., Sze, K.Y., Ma, S.: An explicit material point finite element method for hyper-velocity impact. Int. J. Numer. Methods Eng. **66**(4), 689–706 (2006)

Chapter 7
Karamelo: A Multi-CPU/GPU C++ Parallel MPM Code

The Matlab code presented in the previous chapter is best suited for people new to the MPM who want to learn the method. This code is flexible, but slow. So for long term research, a better code is needed. This chapter presents the design and implementation of `Karamelo`—our open source multi-CPU/GPU parallel C++ package for the material point method. `Karamelo` has been designed to retain the flexibility of small codes while being extremely fast and powerful (specially with the recently added support of GPUs). The structure of this code is based on that of the popular molecular dynamics simulator `LAMMPS` (Plimpton 1995). From `LAMMPS` it inherits an impressive flexibility that allows users to easily add functionalities.

First, `Karamelo`'s code is outlined in Sect. 7.1. The code's particular class system inherited from `LAMMPS` is described in Sect. 7.2. Pre and post-processing is treated in Sect. 7.3. Then, the user interface is introduced in Sect. 7.4 by detailing the input file's syntax. `Karamelo` is fully parallelized using MPI (Message Passing Interface) . The way this is done is explained in Sect. 7.5. Compiling an open-source project is always a daunting operation. However, with `Karamelo`, compilation is easy as you will later see in Sect. 7.6. `Karamelo` has been created to be easy to extend for rapid prototyping of new ideas while using an efficient core. How to extend it is presented in Sect. 7.7. Support for GPU is presented in Sect. 7.8. Finally, some simulations performed with `Karamelo` involving large deformation and contacts are presented in Sect. 7.9. More interesting simulations done with `Karamelo` are presented in later chapters.

In `Karamelo`, symmetric strain and stress tensors are stored as 3×3 matrices which is different from the common Voigt notation, often presented in FEM textbooks, where they are stored as column vectors. This is to make matrix operations (such as polar decomposition of the deformation gradient tensor) easy.

V. P. Nguyen et al., *The Material Point Method*, Scientific Computation,
https://doi.org/10.1007/978-3-031-24070-6_7

7.1 `Karamelo` in a Nutshell

`Karamelo` is an explicit dynamics MPM code using a Cartesian background grid. It implements various weighting functions including hat functions, cubic B-splines, quadratic Bernstein, and CPDI. It can be used in an updated Lagrangian description or in a total Lagrangian description in either 2D or 3D. For 2D problems, plane strain and axi-symmetric conditions are supported. The simulation outputs are `LAMMPS` dump files in either binary or compressed text format. These dumps can easily be visualized using `Ovito` (Stukowski 2009). So far, the following material models are supported:

- isotropic linear elastic material;
- Neo-Hookean hyperelastic material;
- small strain J2 elasto-plastic material;
- large strain hypoelastic plastic materials;
- EOS for weakly compressible fluids and gases.

This list is in constant evolution. Please check the code's official webpage for the updated list of supported materials: www.karamelo.org.

7.2 Hierarchical Class System

`Karamelo` is a C++ code that uses a hierarchical class system directly inherited from `LAMMPS`. At its center lies the Pointers class. All the main classes (to the exception of the classes MPM and Var) are inherited from it as show in Fig. 7.1. This structure allows all these classes to access elements from all the other classes, while being independent.

When `Karamelo` is launched, it creates the class MPM which contains all the pointers to the other main classes which are (see Fig. 7.1):

- `Universe`: sets up partitions of processors so that multiple simulations can be run, each on a subset of the processors allocated for a run, e.g. by the `mpirun` command.

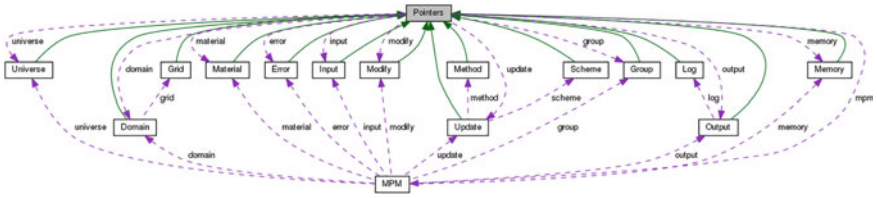


Fig. 7.1 Class hierarchy within Karamelo source code. All classes are inherited from the class Pointers. This diagram was automatically generated by `Doxygen`. Green arrows show parental links, while the dashed purple ones show pointers (de Vaucorbeil and Nguyen 2021a)

- `Domain`: stores the simulation dimensions (i.e. 2D or 3D), the simulation box geometry, the list of the user defined geometric regions and solids, as well as the background grid if the updated Lagrangian is used.
- `Grid`: stores all informations related to the background grid(s): number of cells as well as all the nodes' properties such as position, velocity, mass, etc. It also updates the Grid updating step (Fig. 1.12).
- `Material`: stores all the user defined equations of state, elasto-plastic, damage, and temperature laws as well as the different materials which are a combination of the formers.
- `Error`: prints all error and warning messages.
- `Input`: reads an input script, stores variables, and invokes stand-alone commands that are children classes of the other main classes.
- `Modify`: stores the list of `Compute` and `Fixes` classes, both of which are parent styles.
- `Update`: stores everything related to time steps as well as the `Scheme` and `Method` classes.
- `Method`: parent class of all the MPM methods supported: ULMPM, TLMPM, ULCPDI and TLCPDI, to date.
- `Scheme`: parent class of all the computational cycle schemes supported: modified update stress last to date.
- `Group`: manipulates groups that atoms are assigned to via the group command. It also computes various attributes of groups of atoms.
- `Log`: it is used to generate the screen printed outputs and the log files.
- `Output`: it is used to generate 4 kinds of output from a simulation: information printed to the screen and log file, dump file snapshots, plots, and restart files.
- `Memory` class handles allocation of all large vectors and arrays.

To alter properties of the system during timestepping "fixes" are used. They are the Karamelo mechanism (directly inherited from `LAMMPS`) for tailoring the operations of a time step for a particular simulation. Essentially everything that happens during a simulation besides the regular MPM algorithm, and output, is a 'fix'. Example of operations are: setting boundary conditions or setting up a virtual indenter.

The computation of various attributes of particles or nodes during timestepping are, however, done using "computes" rather than "fixes". Some example of computes are: calculating the total strain energy, or calculating the maximum plastic strain.

7.3 Pre and Post-processing

`Karamelo` supports two types of particle generation. For simple geometries, it can directly generate the particles. The algorithm is simple: first, a Cartesian grid is built, and the user decides how many particles will populate each cell. Then, the code generates particles for all cells and discard those that fall outside the boundary of the geometry. Supported geometries include spheres, cylinders and blocks in 3D.

The corresponding 2D geometries are disks and rectangles. For complex geometries, `Karamelo` can read a finite element mesh and generate particles as centroids of these elements. Currently it supports `Gmsh`—an open source unstructured mesh generator (Geuzaine and Remacle 2009b).

Similar to `LAMMPS`, the simulation snapshots are visualized using `Ovito` (Stukowski 2009). A typical `LAMMPS` dump file is shown in Listing 7.1.

Listing 7.1 `LAMMPS` dump file which can be processed by `Ovito`

```
1  ITEM: TIMESTEP
2  0
3  ITEM: NUMBER OF ATOMS
4  334328
5  ITEM: BOX BOUNDS sm sm sm
6  −17 50
7  −25 25
8  −25 25
9  ITEM: ATOMS id type x y z damage s11 s22 s33 s12 s13 s23
10 0 1 10.5401 −1.50408 −0.647102 0 −16.9889 −19.2388 −13.7454 11.4435 4.68828 −2.59562
11 1 1 10.5759 −1.51475 −0.215365 0 −15.1273 −15.1966 −14.8969 8.70967 1.73124 −1.20511
12 2 1 10.5759 −1.51475 0.215365 0 −15.1273 −15.1966 −14.8969 8.70967 −1.73124 1.20511 3
13 ...
```

7.4 Input Files

Interacting with `Karamelo` is performed through input files using an easy and flexible syntax. One can for example intuitively add new variables that can be constant:

```
E = 211
```

or depend on internal variables such as time—using the `time` variable—or the particle positions—using the `x`, `y` or `z` variables:

```
T     = 1
r     = sqrt(x*x+y*y)
g     = sin(PI*time/T)
```

Everything else is controlled through functions. For instance, the global dimensionality of the simulation, the domain's size, and the background grid cell size are set by the `dimension(...)` command:

```
dimension(2, xlo, xhi, ylo, yhi, cellsize)
```

A typical input file is given in Listing 7.2. Basically, it performs the following actions:

- define some constants (lines 4 to 8, 12 and 10, 14, 15, 18, 24, 30, 34 and 35);

- define the method (ULMPM or TLMPM or CPDI) together with the basis function (line 11);
- define the dimension together with the computational domain (line 16);
- define regions or geometries (lines 19 and 20). Simple geometries (blocks, cylinders, spheres) are supported;
- define materials (line 22)
- define solids using regions and materials (lines 25 and 26);
- define particle and node groups (lines 28 and 29);
- define initial conditions and boundary conditions on the particle/node groups (lines 31 and 32);
- define outputs (lines 36 and 37).
- define the time increment and the total simulation time (lines 39 and 40);

Listing 7.2 A compact input file

```
1  ######################################################
2  #                  UNITS: MPa, mm, s                 #
3  ######################################################
4  E    = 1e+3                    # Young's modulus
5  nu   = 0.3                     # Poisson's ratio
6  rho  = 1000                    # Density
7  L    = 1                       # a dimension
8  hL   = 0.5*L
9  #------------------------ SET METHOD ------------------------#
10 alpha = 1.0                    #PIC/FLIP mix (0 for full PIC, 1 for full FLIP)
11 method(ulmpm, FLIP, cubic-spline, alpha)
12 scheme(usl)
13 #------------------------ SET DIMENSION ---------------------#
14 N        = 40     # 20 cells per direction
15 cellsize = L/N    # cell size
16 dimension(2,-hL, hL, -hL, hL, cellsize) # 2D problem, which the computational domain is LxL
17 #------------------------ SET REGIONS -----------------------#
18 R = 0.2
19 region(rBall1, cylinder, -hL+R, -hL+R, R)
20 region(rBall2, cylinder, hL-R, hL-R, R)
21 #------------------------ SET MATERIALS ---------------------#
22 material(mat1, linear, rho, E, nu)
23 #------------------------ SET SOLID -------------------------#
24 ppc1d = 2
25 solid(sBall1, region, rBall1, ppc1d, mat1, cellsize,0)
26 solid(sBall2, region, rBall2, ppc1d, mat1, cellsize,0)
27 #------------------ IMPOSE INITIAL CONDITIONS ---------------#
28 group(gBall1, particles, region, rBall1, solid, sBall1)
29 group(gBall2, particles, region, rBall2, solid, sBall2)
30 v = 0.1
31 fix(v0Ball1, initial_velocity_particles, gBall1, v, v, NULL)
32 fix(v0Ball2, initial_velocity_particles, gBall2, -v, -v, NULL)
33 #------------------------ OUTPUT ----------------------------#
34 N_log = 50
35 dumping_interval = N_log*2
36 dump(dump1, all, particle/gz, dumping_interval, dump_p.*.LAMMPS.gz, x, y, vx, vy, seq)
37 dump(dump2, all, grid/gz, dumping_interval, dump_g.*.LAMMPS.gz, x, y, vx, vy)
38 #------------------------ RUN -------------------------------#
39 set_dt(0.001)    # constant time increments of 0.001
40 run_time(3.5)    # run for a period of 3.5 seconds
```

7.5 Parallelization Using MPI

`Karamelo` supports multi-CPU and multi-GPU computations through the use of MPI (Message Passing Interface). In `Karamelo`, the total domain defined by the span of the background grid is equally split amongst the different CPUs (or GPUs) used (see Fig. 7.2). The connection between the different sub-domains is performed by the means of ghost nodes. The use of ghost nodes was preferred to that of ghost particles since Ruggirello and Schumacher (2014) observed that the later is superior over the former for scaling to large-scale problems. Therefore, the nodes making of all the boundary mesh cells are shared amongst multiple CPUs (or GPUs) (Fig. 7.2). First, the interaction between these nodes and the local particles (i.e., the particles present in the domain allocated to the current CPU or GPU) are calculated. Then, the results are reduced over all the CPUs (or GPUs) sharing the same nodes. In one time step, reduction is done once after calculating the nodes' mass, and everytime their forces and velocities are computed. In the case of CPDI, however, as the domain of a given particle can be overlapping multiple CPUs, both ghost nodes and ghost particle's approach is used.

For more detail concerning parallelization a MPM code, we refer to Huang et al. (2008) who has described a parallel MPM using OpenMP for shared memory machines and Li and Sulsky (2000) presented a 3D parallel MPM code, written in Fortran, modified from a serial MPM code using MPI.

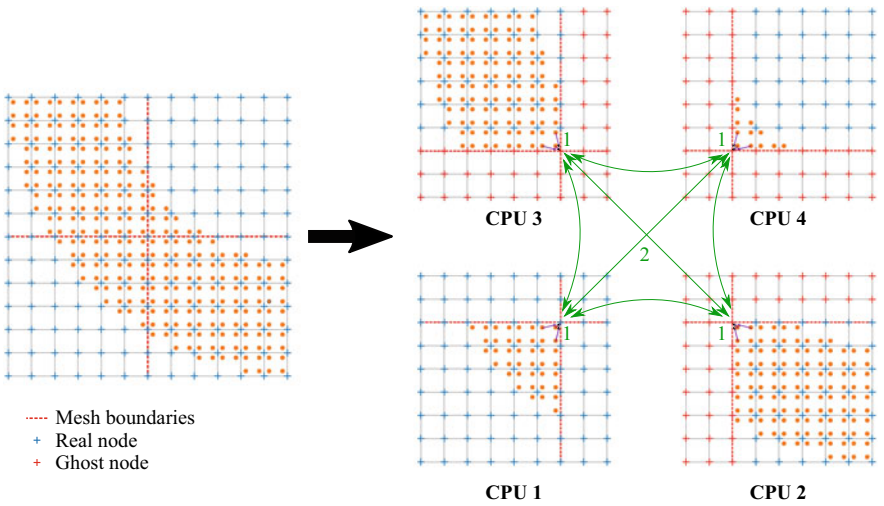


Fig. 7.2 Illustration of the mapping from particles to node order for nodes that are shared amongst different CPUs. 1: mapping is done locally, 2: the mass, or forces and velocities are reduced. The CPU that on which resides the real node is the one that manages the reduction (de Vaucorbeil and Nguyen 2021a)

7.6 Compilation

Building an open-source project can be a daunting task for many users. However, with Karamelo, there is no hardship as building it has been made easy with the use of CMake (https://cmake.org). CMake automatically checks dependencies and generates the Makefile.

As Karamelo relies on two non-standard libraries that are gzstream (https://www.cs.unc.edu/Research/compgeom/gzstream/) for the compression of dump files and Kokkos (https://kokkos.org/) for GPU support, the only user intervention needed to correctly build Karamelo is to make sure that these libraries' source code are present in the source directory via the dedicated git command: `git submodule update --init --recursive`.

7.7 Extending Karamelo

Karamelo has been created to combine rapid prototyping of new ideas with the use of efficient software engineering. This has been achieved using the ingenious hierarchical class system inherited from LAMMPS. Extending Karamelo is therefore relatively easy as is explained in the following.

The easiest way to extend Karamelo is by the implementation of new fixes. Using fixes, users can implement many operations that can alter the system such as: changing particles or node attributes (position, velocity, forces, etc.), implement boundary conditions, reading/writing data, or even saving information about particles for future use (previous positions, for instance).

Adding a Fix. All fixes are derived from the class Fix and must have a constructor with the signature: `FixNew(class MPM *, vector<string>)`.

Every fix must be registered in Karamelo by writing the following lines of code in the header before include guards:

```
#ifdef FIX_CLASS
FixStyle(fix_name,FixNew)
#else
```

where "fix_name" is a name of the fix in the script and FixNew is the name of its class. This little piece of code allows Karamelo to find the new fix when it reads an input file. In addition, the fix header must be included in the file "style_fix.h". Once this is done and the project build again, the fix can be used using the following function:

```
fix(fix_ID, fix_name, arg_1, arg_2, ..., arg_N)
```

Remark 38 The number of arguments the fix function takes is variable.

Adding a computational cycle schemes. New computational cycle schemes can be added as easily as fixes. This could allow a given user to simply test a new scheme or add one that is not currently supported. Schemes are derived from the class `Scheme`. They must have constructor with the signature: `SchemeNew(class MPM *, vector<string>)` and feature their own definition of the `setup()` and `run(Var condition)` functions. The former is used to do any required setup at the beginning of the simulation, before timestepping start. The later features the implementation of the timestepping loop and takes a condition as only argument.

Similarly to fixes, every scheme must be registered by writing the following lines of code in the header before include guards:

```
#ifdef SCHEME_CLASS
SchemeStyle(scheme_name,SchemeNew)
#else
```

where "scheme_name" is a name of the new scheme in the script and SchemeNew is the name of its class. Also, the scheme header must be included in the file "style_scheme.h". The use of the new added scheme is done by invocated the follow:

```
scheme(scheme_name)
```

Remark 39 The scheme selected by default is modified update stress last (MUSL).

Adding an MPM variant. The same process used for adding fixes and schemes is used to add MPM variants. The class corresponding to a variant is derived from the class `Method`. It must have a signature of the type `NewMPM(class MPM *, vector<string>)` and feature the definition of a number of functions corresponding to the different steps of the algorithm. New variants must be registered as follows:

```
#ifdef METHOD_CLASS
MethodStyle(newmpm,NewMPM)
#else
```

where "newmpm" is a name of the added variant. Just like for fixes and schemes its header must be included in the file "style_method.h". In the input file, the function method defines the use of a given MPM variant. For example the function

```
method(newmpm, PIC, linear)
```

tells `Karamelo` to use the "newmpm" variant of MPM with the PIC integration scheme and linear shape functions.

7.8 GPU Support

With MPM's ability to handle robustly complex simulation problems, there is a demand and need for faster and efficient MPM codes. `Karamelo` was originally created as only a multi-CPU support. This enabled `Karamelo` to handle more complex problems than single-CPU codes. However, even though its parallel performances are honest, they are far from that of codes that use GPUs.

GPUs are modern massively parallel simulation hardware now readily available for desktop machines which are able to generate three order of magnitude speed gains compared to CPU based MPM codes (Dong and Grabe 2018). To harvest these performances, the standard approach (and still the most efficient) is to write code directly in CUDA.

Codes written in CUDA can only be run on Nvidia GPUs. Nvidia is currently the leader of this marker, but other players such as AMG are catching up. CUDA is not the only single-platform; moreover, it is also hard to write in. Therefore, using it for `Karamelo` would not only limit its hardware compatibility but also the ease with which users can add functionalities or change the code.

An elegant solution is to use `Kokkos`. The `Kokkos` ecosystem as a C++ library implements a manycore portable programming model that provides abstracted parallelization and memory management (Edwards et al. 2014). Basically, `Kokkos` allows to harvest the parallel capabilities of different hardware without writing code that is machine specific. As a C++ library, no other language needs to be used.

With the use of `Kokkos`, `Karamelo`'s code is fully parallelized and can run easily on multiple CPUs or GPUs. With GPUs, speed-ups are of the order of one or two orders of magnitude compared to CPUs, depending on the problem simulated.

7.9 Some Simulations

This section presents simulations that involve large deformation and contacts which are signature application of the MPM. Furthermore, these simulations are practical engineering problems of which experiments have been conducted. Concretely, the following problems are presented

- Taylor anvil test (Sect. 7.9.1);
- Upsetting of a cylindrical billet (Sect. 7.9.2);
- Cold spraying (Sect. 7.9.3)
- Scability test (Sect. 7.9.4)

All results were obtained with the MUSL formulation with the blending PIC/FLIP parameter $\alpha = 0.99$. We refer to Sect. F.3 for a simple 3D problem if you need to verify your 3D MPM implementation.

7.9.1 Taylor Anvil Test

The Taylor anvil test is a well suited problem to evaluate the performance of elasto-plastic constitutive models and numerical codes when large plastic deformations occur (Wilkins and Guinan 1973; Johnson and Holmquist 1988; Predebon et al. 1991). This test involves a OFHC Copper cylinder of original length $L_0 = 25.4$ mm, and original diameter $D_0 = 7.6$ mm, hitting a stationary rigid wall at a high velocity $v_0 = 190$ m/s. It was used by Johnson and Holmquist (1988) to compare various constitutive models for both OFHC Copper and Armco Iron. In their work, they performed experiments to determine the material parameters for these two materials using different constitutive models. They also performed numerical simulations to compare the models.

Herein, we are interested in how the MPM prediction of the deformed bar in terms of the final diameter, bulge and length (see Fig. 7.3 for an illustration of these quantities) compares with the experiments. This problem is now solved using the ULMPM and the TLMPM. In the literature, this test has been studied using the axi-symmetric ULMPM (Sulsky and Schreyer 1996) and recently by Liang et al. (2019).

The OFHC Copper material is modeled using the elasto-plastic constitutive model presented in Sect. 4.3, but without damage and temperature effects. The material parameters used are taken from Sulky's work and are listed in Table 7.1. The performance of the MPM is accessed using the error measure introduced by Johnson and Holmquist (1988) and defined as:

$$\bar{\Delta} = \frac{1}{3}\left[\frac{|\Delta L|}{L_T} + \frac{|\Delta D|}{D_T} + \frac{|\Delta W|}{W_T}\right] \tag{7.1}$$

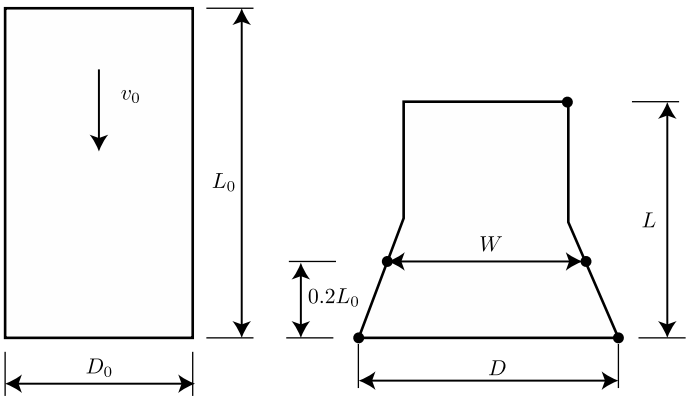
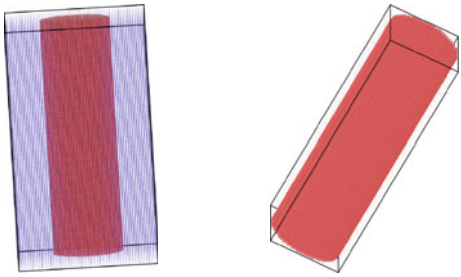


Fig. 7.3 Taylor bar impact for an elasto-plastic OFHC copper cylinder given an initial downward velocity of 190 m/s

Table 7.1 Material parameters for the OFHC Copper material used in the simulation of the Taylor bar impact test

Material parameters		Parameters for Johnson-Cook model		Parameters for EOS	
Density	8940 kg/m^3	A	65 MPa	c_0	3933 m/s
Young's modulus	115 GPa	B	356 MPa	S_α	1.5
Poisson's ratio	0.31	C	0.013	Γ_0	0
		n	0.37		



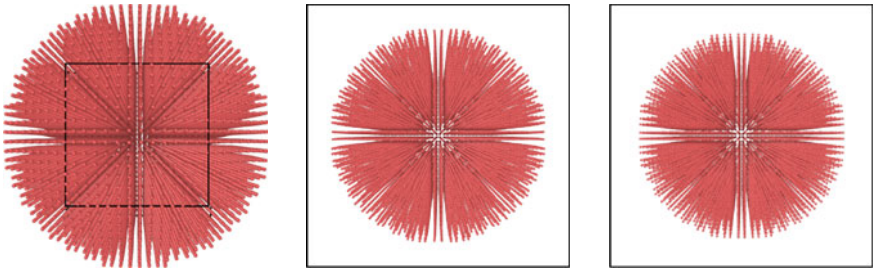
(a) ULMPM (b) TLMPM

Fig. 7.4 Taylor anvil test: initial particle distribution and grid of ULMPM and TLMPM. Note that the cylinder axis is in the x direction and for the TLMPM there is no gap between the cylinder and the wall

where L_T, D_T and W_T are the length, diameter and bulge measured experimentally and $\Delta L = L_f - L_T$, $\Delta D = D_f - D_T$ and $\Delta W = W_f - W_T$. The MPM length, diameter and bulge are denoted by L_f, D_f and W_f, respectively. All these lengths are measured based on the distances between particles marked in Fig. 7.3. As it is impossible to have particles at exact locations defining the bulge W, our prediction for W is worse than the length and the diameter. Locating these particles and their distances were done manually using tools provided in `Ovito` (Stukowski 2009).

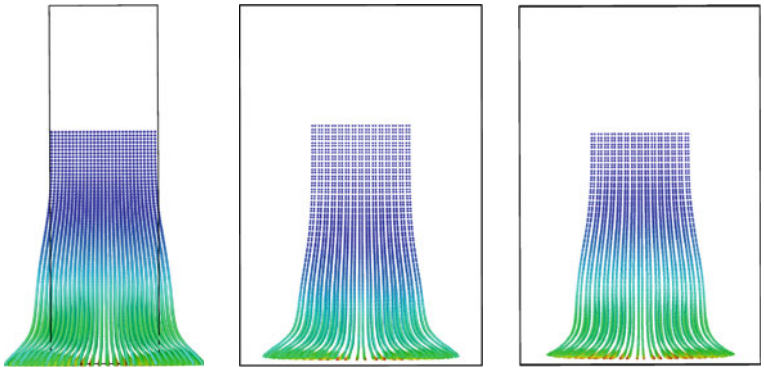
Figure 7.4 shows the distribution of the material points and the grid used in the ULMPM and TLMPM. As can be seen, the TLMPM requires a grid just encompassing the solid in its initial undeformed configuration, while for the ULMPM the grid must cover a larger space that will cover the entire deformation space of the solid. For the TLMPM (linear shape functions), the cell size is $h = 0.25$ mm with 1 material point per element for a total of 74052 points. For the ULMPM (or just MPM) we use the hat functions and the cubic B-spline functions (BSMPM). Eight particles per cell are used for both cases. The presence of the wall is simulated by zeroing v_{xI} of the bottom nodes (where the wall is located).

Figure 7.5 presents the geometry (looking from the bottom) of the deformed cylinder at the end of the simulation i.e., $t \approx 63$ μs when the kinetic energy is close to zero. All MPM variants perform very well and much better than LS-DYNA SPH given in Ma et al. (2009b). Figure 7.6 presents the deformed configuration of the bar. The predicted values by TLMPM and ULMPM (MPM and BSMPM) for the final diameter, bulge and length are given in Table 7.2. Also presented is the results from Sulsky and Schreyer (1996), just for reference.



(a) TLMPM (b) MPM (c) BSMPM

Fig. 7.5 Final configurations of the Taylor bar (bottom view). The black squares denote the background grids



(a) TLMPM (b) MPM (c) BSMPM

Fig. 7.6 Taylor anvil test. Deformed configurations obtained with different MPM variants. The color presents the equivalent plastic strain. Visualization done with `Ovito` (Stukowski 2009)

Table 7.2 Results of the Taylor anvil test

	Experiment	TLMPM	MPM	BSMPM	MPM (Sulsky)
Diameter D [mm]	13.5	13.9	14.4	13.9	14.6
Bulge W [mm]	10.1	9.4	8.9	9.5	9.12
Length L [mm]	16.2	16.2	17.2	16.5	18.3
$\bar{\Delta}$ [–]	n/a	0.03	0.08	0.04	0.1

Immediately upon impact, elastic waves followed by plastic waves are generated at the impact interface, and travel back and forth between the bottom and top surfaces. The bottom part of the cylinder bulges out due to compression while the top part keeps almost undeformed. This deformation of the cylinder shows a familiar mushroom-like deformation mode in the Taylor test.

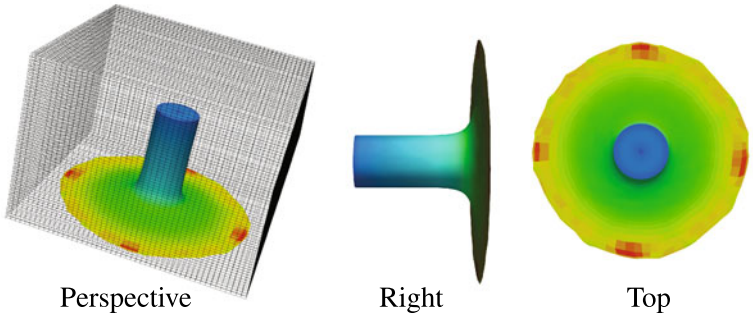
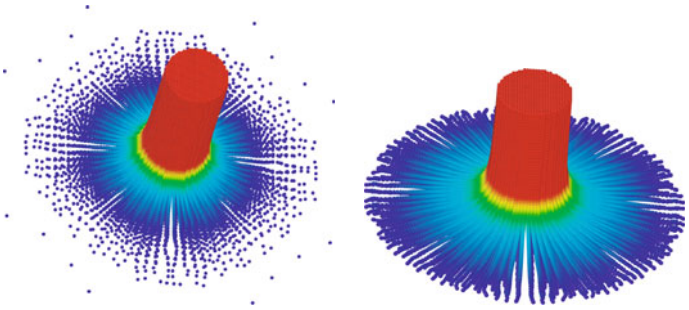


Fig. 7.7 Final configuration of the Taylor bar with $v_0 = 750$ m/s. GPIC (TL) set up: computational domain is $40 \times 28 \times 40$ discretized by a Eulerian grid of $50 \times 60 \times 50$ cells with 52920 eight-node hexahedral elements for the cylinder

Remark 40 It was this Taylor anvil test that led Ma et al. (2009b) to conclude that explicit MPM is faster than LS-DYNA ULFEM, but with a slightly lower accuracy. In our humble opinion, having a faster method would not make industry people to choose it. This is simple as people are so familiar to the FEM and there are excellent FEM packages with user friendly user interfaces for both pre and post processing. For the MPM to be used by industry the MPM community should consider solving problems that the FEM struggle to solve. Sadly, up to now, no such simulations are published.

We turn now to the case in which the cylinder velocity is 750 m/s, to see if MPM/TLMPM/GPIC can capture the extreme deformation. For this high velocity, we are not aware of any experiments, so we just evaluate the robustness of different MPMs. The results given in Fig. 7.7 indicates that GPIC is robust for solving massive deformation (the TLMPM is also robust). The deformed shape is similar to the one obtained by the OTM (a truly meshfree method) in Li et al. (2010). On the other hand, the MPM with linear functions exhibits numerical fractures i.e., particle separation not due to a physical fracture model, see Sect. 2.6.1 for a simple demonstration; using a smoother basis such as quadratic B-splines improves the solution (Fig. 7.8).

In the MPM, numerical fracture is treated by a 'remeshing' technique called particle splitting in which a particle is split into many particles when the particle spacing is large relatively to the grid cell size (Ma et al. 2009b). However, there exists more than one splitting criteria to determine when one should perform particle splitting (Gracia et al. 2019).



(a) Linear basis (b) Quadratic B-splines

Fig. 7.8 Final configuration of the Taylor bar with $v_0 = 750$ m/s: numerical fracture obtained with the ULMPM. Details: 50220 particles on a grid with $50 \times 60 \times 50$ cells

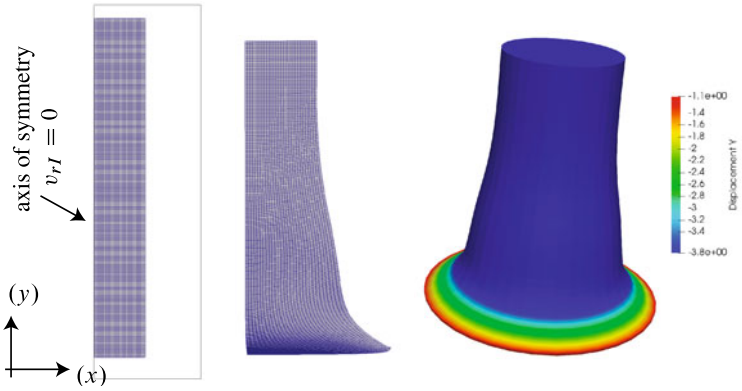


Fig. 7.9 Final configuration of the Taylor bar with $v_0 = 190$ m/s: axis-symmetric GPIC (TL): the model (left), the result (middle) and processed result in `Paraview` to have a full model (right). See Fig. 7.10 for how to do this

Axi-symmetric formulations. To verify the implementation of axi-symmetric MPM, we solve this Taylor anvil test again using an axi-symmetric model, see Fig. 7.9. We only present the results obtained with GPIC (TL and eight-node hexahedral elements) but ULMPM and TLMPM behave similarly.

7.9.2 Upsetting of a Cylindrical Billet

In upsetting, a cylindrical billet is compressed between two platens. This process is used in manufacturing as a means to pre-form workpieces prior to applying another operation such as rolling or extrusion. The benchmark problem consists of a steel cylinder 20 mm in diameter and 30 mm in height. Due to symmetry, only half of the

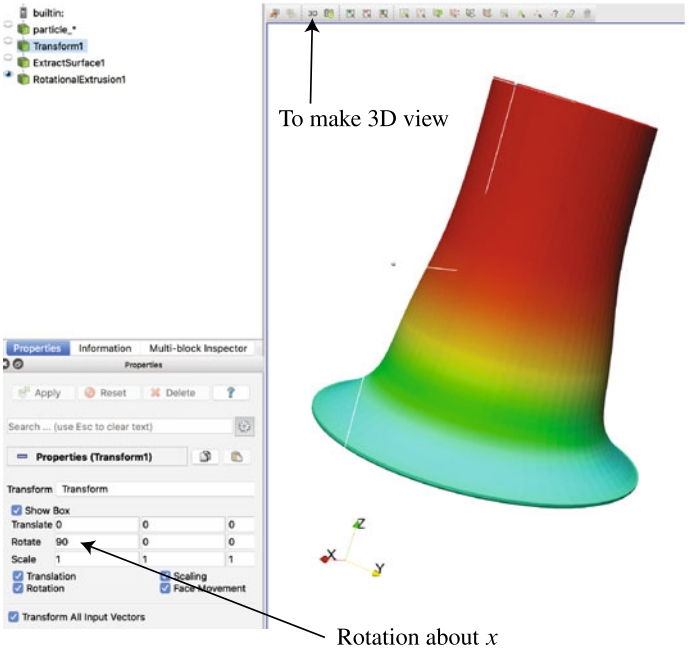
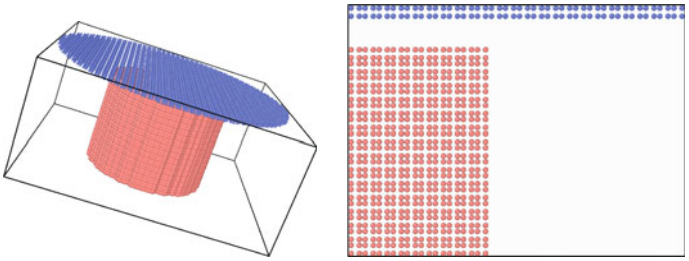


Fig. 7.10 From axi-symmetric model to 3D in `Paraview`: this is achieved using this sequence of filters: (1) Transform with rotation w.r.t x 90 degrees (to make the cylinder axis aligned with z direction as the cylinder was aligned in the y direction in our code), (2) Extract Surface and (3) Rotational Extrusion with resolution equals 64



(a) Full 3D model (b) 2D axi-symmetric model

Fig. 7.11 Upsetting of a cylindrical billet: initial particle distribution (de Vaucorbeil and Nguyen 2021a)

cylinder is modeled. The initial distribution of the particles is shown in Fig. 7.11 for both 3D and axi-symmetric simulations.

The platen starts above the workpiece and moves down at a constant velocity of 1 m/s, compressing the billet. For simplicity, it is assumed that there is no slip between the platen and the cylindrical workpiece. The billet is made of steel with a Young's

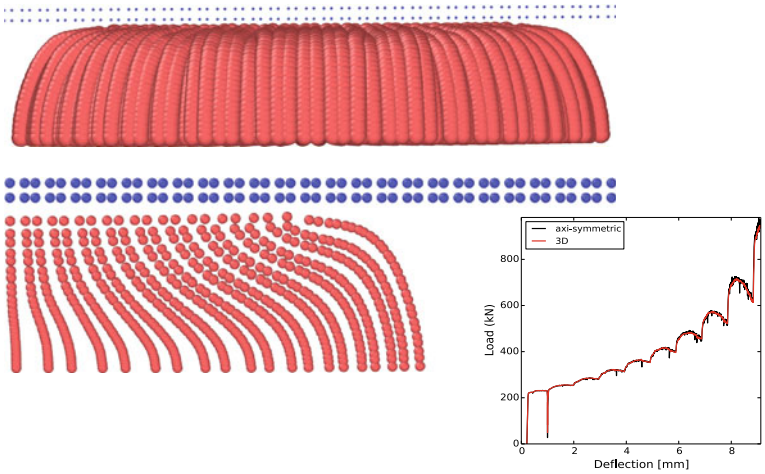


Fig. 7.12 Upsetting of a cylindrical billet: results in terms of deformed shape and load-deflection curves. The deflection is the averaged displacement of the particles on the top surface of the cylinder and the load is computed as a sum of all nodal internal force in the cylinder direction of all nodes locating on the bottom surface of the cylinder

modulus of 200 GPa, Poisson's ratio of 0.3, density of 7800 kg/m^3, and an initial yield strength of 0.70 GPa with a strain hardening slope of 0.30 GPa. That is, in the Johnson-Cook model (Johnson and Cook 1985), $A = 0.70$ GPa, $B = 0.30$ GPa, $C = 0$ and $n = 1.0$.

The 3D simulation consists of 47744 particles (including the rigid particles modeling the platen) whereas the axi-symmetric one contains only 696 particles. And yet, their results, given in Fig. 7.12, are very similar to that of Sulsky and Kaul (2004).

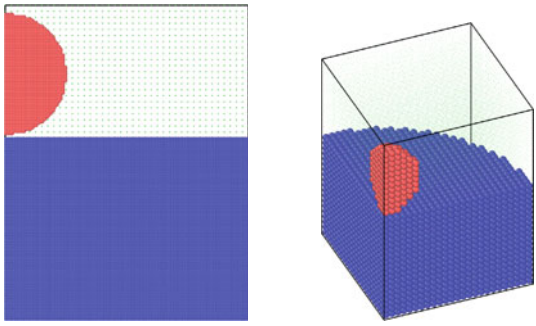
7.9.3 Cold Spraying

Cold spraying (CS) is a coating deposition method in which solid powders (1 to 50 micrometers in diameter) traveling at velocities up to 1200 m/s impinge on a substrate. During this impact, particles undergo plastic deformation and adhere to the surface. Unlike thermal spraying techniques, e.g., plasma spraying, flame spraying, or high velocity oxygen fuel, the powders are not melted during the spraying process.

The coating is formed from many individual feedstock impact events. Therefore, the understanding of a single feedstock impact and its resulting morphology is vital to shed lights on the key parameters affecting bulk coating properties. It is very difficult to observe the whole deformation process via experiments because of the extremely short impact time scale. Numerical simulations thus play an important role in studying the spray powder deformation process. Common numerical methods include Lagrangian FEM, see e.g. Assadi et al. (2003), Eulerian methods (Li

Table 7.3 Material parameters copper (Assadi et al. 2003)

Elastic parameters		JC model		EOS		Damage		Temperature	
Density	8960 kg/m^3	A	90 MPa	c_0	3940 m/s	D_1	0.54	T_r	298 K
Young modulus	124 GPa	B	292 MPa	S_α	1.489	D_2	4.89	T_m	1356 K
Poisson ratio	0.34	C	0.025	Γ_0	2.02	D_3	–3.03		
		n	0.31			D_4	0.014		
		m	1.09			D_5	1.12		



(a) axi-symmetrica model (b) 1/4 3D model

Fig. 7.13 Cold spraying with a single impact: 2D axi-symmetric model and 1/4 3D model. The bottom surface of the substrate is fixed whereas in the symmetric planes, the nodes are fixed in the direction normal to these planes. The green dots denote the background grid nodes (de Vaucorbeil and Nguyen 2021a)

et al. 2016) and Smoothed Particle Hydrodynamics (SPH), see e.g. Mason (2015), Gnanasekaran et al. (2019) have been used for cold spraying simulations. From the review article of Yin et al. (2010), Eulerian and SPH methods are more accurate than the FEM.

The most common scenario considered in the literature is a single impact: a spherical powder particle impacts a cylinder substrate. The spherical powder has a radius r of 10 μm and the cylindrical substrate is $8r$ in diameter and $3r$ in height. Both the powder and the substrate are made of copper, as it is the most widely used material in cold spray simulations. The material parameters used are given in Table 7.3 taken from Assadi et al. (2003).

Both 2D axi-symmetric model and 3D model are considered, see Fig. 7.13. As heat conduction is not yet coded in `Karamelo`, only adiabatic condition is considered and the temperature increase is due only to plastic deformation. Note that the contact between the powder and the substrate is no-slip contact—the default contact in the MPM.

Some results are shown in Figs. 7.14 and 7.15. Figure 7.15 presents the evolution of the plastic strain and temperature in time. The axi-symmetric and 3D results are quite similar.

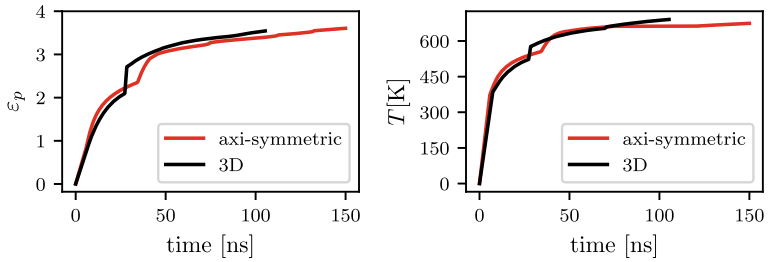
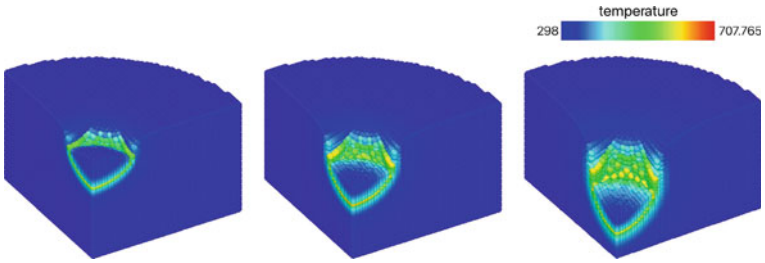


Fig. 7.14 Cold spraying with a single impact: evolution of plastic strain and temperature



(a) 49 ns (b) 74 ns (c) 105 ns

Fig. 7.15 Cold spraying with a single impact: deformation process (3D model)

7.9.4 Scalability Tests

As Moore's law is flattening (the power of CPUs do not double every two years anymore), chip manufacturer are relying on packing more computational cores onto a single chip to increase their computational power. Modern application therefore need to be fully parallized to make the best use of the technology available nowadays. In Sects. 7.5 and 7.8, the way `Karamelo` was parallelized using both CPUs and GPUs has been presented. Here, the speedup gain obtained as the number of CPU cores increases and by the use of GPUs are presented, using the well known Taylor bar impact test. These tests are done using both TLMPM and ULMPM.

Similarly to what was done in the work by de Vaucorbeil et al. (2020), the mesh cell size is $h = 0.25$ mm with 1 material point per element for a total of 74052 points. Moreover, linear shape functions are used. The initial velocity is set at $v_0 = 190$ m/s. When using the ULMPM, the simulation cell is $25.4 \times 15.0 \times 15.0$ mm^3. All simulations are ran for a total of 20000 steps.

CPU. The CPU scalability tests have been performed using a Xeon-E5-2667-v3 chip made of 16 cores. The same simulations for TLMPM and ULMPM, respectively were performed using numbers of core varying from 1 to 16. The simulation times were recorded and divided by those obtained when using a single core. As one can see in Fig. 7.16, the speedup gain for ULMPM and TLMPM is comparable. The increase of

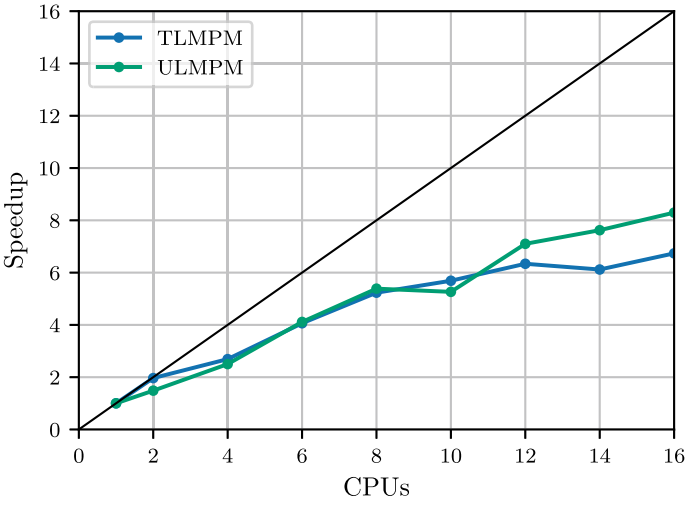


Fig. 7.16 Scalability of the TLMPM and the ULMPM compared using a Xeon-E5-2667-v3 type CPUs. The black line corresponds to the maximum possible speedup gain (de Vaucorbeil and Nguyen 2021a)

speed is substantial when the number of cores is lower than 10. When this number is higher, though the performances tend to plateau. This suggest that the computation time is dominated by the time passing messages, or waiting for messages.

GPU. The GPU speedup tests were conducted using a single Nvidia Tesla Volta V100-SXM2-32GB GPU. The time it took to run the same simulation was compared with that of a single CPU. With ULMPM, the simulation ran 141 times faster on the GPU than on a single CPU. With TLMPM, the speedup factor was 34.

GPUs are amazing parallel computing devices and `Karamelo` can make great use of this power. But not everyone has access to them. For those, `Karamelo` will remain CPU compatible.

7.10 Conclusions

While we are happy with `Karamelo`, specially its GPU support, there is room to improve the MPI parallelization and it only supports explicit dynamics. It only has a few constitutive models, but you can help with that! However, for completeness sake's, its is important to briefly present other MPM implementations.

Amongst the CPU codes, there is `Uintah` developed by Parker (2002). It is a parallel MPM code using Message Passing Interface (MPI) with excellent scalability of more than 1000 processors (16 million particles) as demonstrated in Parker et al. (2006). A large number of constitutive models is implemented in `Uintah` with

implicit solvers. Ma et al. (2010) described an object-oriented C++ implementation of MPM and Sinaie et al. (2017) presented a MPM implementation using Julia.

A couple of GPU codes also exists like `ep2-3De` an explicit GIMP implementation for GPUs developed by Wyser et al. (2021) for elasto-plastic problems in Geomechanics. We would also like to mention the impressive multi-GPU `Claymore` code developed by computer scientists for computer animations (Wang et al. 2020). This code is unfortunately not designed for physical simulations and is extremely complex. This makes `Karamelo` the only easy to run and easy to modify open source GPU MPM code for engineering simulations.

We refer back to Sect. 1.7 for a list of MPM codes.

References

Assadi, H., Gärtner, F., Stoltenhoff, T., Kreye, H.: Bonding mechanism in cold gas spraying. Acta Mater. **51**(15), 4379–4394 (2003)

Carter Edwards, H., Trott, C.R., Sunderland, D.: Kokkos: enabling manycore performance portability through polymorphic memory access patterns. J. Parallel Distrib. Comput. **74**(12), 3202–3216 (2014). ISSN 0743-7315. https://doi.org/10.1016/j.jpdc.2014.07.003. http://www.sciencedirect.com/science/article/pii/S0743731514001257. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing

de Vaucorbeil, A., Nguyen, V.-P., Hutchinson, C.R.: A total-Lagrangian material point method for solid mechanics problems involving large deformations. Comput. Methods Appl. Mech. Eng. **360**, 112783 (2020). https://doi.org/10.1016/j.cma.2019.112783

de Vaucorbeil, A., Nguyen, V.P.: Karamelo: an open source parallel C++ package for the material point method. Comput. Particle Mech. **8**, 767–789 (2021a)

Dong, Y., Grabe, J.: Large scale parallelisation of the material point method with multiple gpus. Comput. Geotech. **101**, 149–158 (2018)

Geuzaine, C., Remacle, J.F.: Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. Int. J. Numer. Meth. Eng. **79**(11), 1309–1331 (2009)

Gnanasekaran, B., Liu, G.-R., Yao, F., Wang, G., Niu, W., Lin, T.: A smoothed particle hydrodynamics (sph) procedure for simulating cold spray process-a study using particles. Surface Coatings Technol. **377**, 124812 (2019)

Gracia, F., Villard, P., Richefeu, V.: Comparison of two numerical approaches (DEM and MPM) applied to unsteady flow. Comput. Particle Mech. 1–19 (2019)

Huang, P., Zhang, X., Ma, S., Wang, H.K.: Shared memory OpenMP parallelization of explicit MPM and its application to hypervelocity impact. Comput. Model. Eng. Sci. **38**(2), 119–147 (2008)

Johnson, G.R., Cook, W.H.: Fracture characteristics of three metals subjected to various strains, strain rates, temperatures and pressures. Eng. Fract. Mech. **21**(1), 31–48 (1985)

Johnson, G.R., Holmquist, T.J.: Evaluation of cylinder? impact test data for constitutive model constants. J. Appl. Phys. **64**(8), 3901–3910 (1988)

Li, X., Sulsky, D.: A parallel material-point method with application to solid mechanics. In: Brebbia, C.A., Ingber, M., Power, H. (eds.), Computational Science–ICCS 2002. Applications of High-Performance Computing in Engineering VI, vol. 2331. WIT Press, Southampton (2000)

Li, B., Habbal, F., Ortiz, M.: Optimal transportation meshfree approximation schemes for fluid and plastic flows. Int. J. Numer. Meth. Eng. **83**(12), 1541–1579 (2010)

Li, W.Y., Yang, K., Yin, S., Guo, X.P.: Numerical analysis of cold spray particles impacting behavior by the Eulerian method: a review. J. Therm. Spray Technol. **25**(8), 1441–1460 (2016)

Liang, Y., Zhang, X., Liu, Y.: An efficient staggered grid material point method. Comput. Methods Appl. Mech. Eng. **352**, 85–109 (2019)
Ma, S., Zhang, X., Lian, Y., Zhou, X.: Simulation of high explosive explosion using adaptive material point method. Comput. Modeling Eng. Sci. (CMES) **39**(2), 101 (2009)
Ma, Z.T., Zhang, X., Huang, P.: An object-oriented MPM framework for simulation of large deformation and contact of numerous grains. Comput. Model. Eng. Sci. **55**(1), 61–87 (2010)
Mason, L.S.: Modelling cold spray splat morphologies using smoothed particle hydrodynamics. PhD thesis, Heriot-Watt University (2015)
Parker, S.G.: A component-based architecture for parallel multi-physics pde simulation. In: Sloot, P.M.A., Hoekstra, A.G., Kenneth Tan, C.J., Dongarra, J.J. (eds.), Computational Science – ICCS 2002. Lecture Notes in Computer Science, vol. 2331, pp. 719–734. Springer, Berlin (2002)
Parker, S.G., Guilkey, J., Harman, T.: A component-based parallel infrastructure for the simulation of fluid-structure interaction. Eng. Comput. **22**(3–4), 277–292 (2006)
Plimpton, S.: Fast parallel algorithms for short-range molecular dynamics. J. Comput. Phys. **117**(1), 1–19 (1995)
Predebon, W.W., Anderson, C.E., Walker, J.D.: Inclusion of evolutionary damage measures in Eulerian wavecodes. Comput. Mech. **7**(4), 221–236 (1991)
Ruggirello, K.P., Schumacher, S.C.: A comparison of parallelization strategies for the material point method. In: 11th World Congress on Computational Mechanics, pp. 20–25 (2014)
Sinaie, S., Nguyen, V.P., Nguyen, C.T., Bordas, S.: Programming the material point method in Julia. Adv. Eng. Softw. **105**, 17–29 (2017)
Stukowski, Alexander: Visualization and analysis of atomistic simulation data with ovito-the open visualization tool. Model. Simul. Mater. Sci. Eng. **18**(1), 015012 (2009)
Sulsky, D., Kaul, A.: Implicit dynamics in the material-point method. Comput. Methods Appl. Mech. Eng. **193**(12–14), 1137–1170 (2004)
Sulsky, D., Schreyer, H.L.: Axisymmetric form of the material point method with applications to upsetting and Taylor impact problems. Comput. Methods Appl. Mech. Eng. **139**, 409–429 (1996)
Wang, X., Qiu, Y., Slattery, S.R., Fang, Y., Li, M., Zhu, S.-C., Zhu, Y., Tang, M., Manocha, D., Jiang, C.: A massively parallel and scalable multi-gpu material point method. ACM Trans. Graph. **39**(4) (2020)
Wilkins, M.L., Guinan, M.W.: Impact of cylinders on a rigid boundary. J. Appl. Phys. **44**(3), 1200–1206 (1973)
Wyser, E., Alkhimenkov, Y., Jaboyedoff, M., Podladchikov, Y.Y.: An explicit gpu-based material point method solver for elastoplastic problems (ep2-3de v1.0) (2021)
Yin, S., Wang, X., Bao-peng, X., Li, W.: Examination on the calculation method for modeling the multi-particle impact process in cold spraying. J. Therm. Spray Technol. **19**(5), 1032–1041 (2010)

Chapter 8
Contact and Fracture

Up to this point, a basic MPM formulation for solid mechanics and its implementation has been presented. A tutorial code written in Matlab and a research-oriented code written in C++ were provided. Actually, if you are willing to learn a new programming language, we present in Appendix F another MPM code written in `Julia`. While this is sufficient for simulating various interesting problems in solid mechanics, topics such as frictional contacts and fracture were not throughly discussed.

This chapter is devoted to such topics. We present the most widely used contact algorithm for the updated Lagrangian MPM in Sect. 8.1. This is followed by a contact model for the total Lagrangian MPM in Sect. 8.2 and for GPIC in Sect. 8.3. Next, we provide some contact simulations in Sect. 8.4 to evaluate the performance of the two contact models. Finally, we discuss fracture modeling in Sect. 8.5. Recent advances in fracture mechanics including variational fracture theories and their approximations (phase-field approximation and eigendeformation approximation) are presented.

8.1 Contacts in the ULMPM

This section presents the contact algorithm of Bardenhagen et al. (2000) for deformable bodies. The case of contact between deformable bodies and rigid bodies will be treated subsequently as simplification. This contact algorithm might be viewed as a *predictor-corrector* scheme, in which the (trial) nodal velocities are predicted from the solution of each body separately (as if no contact occurred) and then corrected using a contact model. The contact algorithm applies only for *contact nodes* which are defined as those who receive contribution from particles of more than one body, cf. Fig. 8.1. For simplicity, the discussion is confined to contacts between different bodies, self-contact requires a special treatment and is discussed in Homel and Herbold (2017).

V. P. Nguyen et al., *The Material Point Method*, Scientific Computation,
https://doi.org/10.1007/978-3-031-24070-6_8

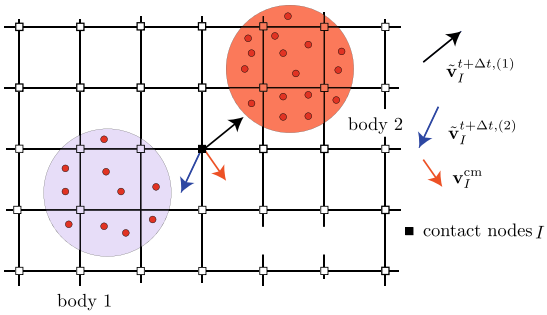


Fig. 8.1 Two bodies come into contact. Contact or overlapped nodes (black solid square) are those who receive contribution from particles of both bodies

For each body $k = 1, 2, \ldots n$ where n denotes the number of bodies, one solves the standard MPM problem

$$\begin{aligned} m_I^{t,(k)} &= \sum_{p=1}^{n_p^{(k)}} \phi_{Ip} m_p, \quad \mathbf{v}_I^{t,(k)} = \frac{1}{m_I^{t,(k)}} \sum_{p=1}^{n_p^{(k)}} \phi_{Ip} m_p v_p \\ \mathbf{a}_I^{t,(k)} &= \frac{\mathbf{f}_I^{(k)}}{m_I^{t,(k)}} \\ \tilde{\mathbf{v}}_I^{t+\Delta t,(k)} &= \mathbf{v}_I^{t,(k)} + \Delta t \mathbf{a}_I^{t,(k)} \end{aligned} \tag{8.1}$$

where $n_p^{(k)}$ denotes the number of particles making up body k. Note that the tilded velocity field is not final and needs to be corrected for contact nodes. The corrected velocity $\mathbf{v}_I^{t+\Delta t,(k)}$ (without a tilde) will be used for updating particle's stress, position and velocity.

The next step after getting $\tilde{\mathbf{v}}_I^{t+\Delta t,(k)}$ is to detect, at contact nodes, whether two bodies are approaching or departing each other. In what follows the presentation is restricted to two bodies for sake of simplicity. The algorithm is general and can be applied to multiple bodies though. Bardenhagen et al. (2000) proposed an algorithm which is linear in the number of bodies[1]:

$$\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\mathrm{cm}}\right) \cdot \mathbf{n}_I^{(k)} = \begin{cases} \geq 0 & \text{contact} \\ < 0 & \text{release} \end{cases} \tag{8.2}$$

where $\mathbf{a} \cdot \mathbf{b} = a_i b_i$ is the dot product of two vectors, and $\mathbf{v}_I^{\mathrm{cm}}$ is the so-called *center of mass* velocity field which is given by

$$\mathbf{v}_I^{\mathrm{cm}} = \frac{(m_I \tilde{\mathbf{v}}_I)^{t+\Delta t,(1)} + (m_I \tilde{\mathbf{v}}_I)^{t+\Delta t,(2)}}{m_I^{t,(1)} + m_I^{t,(2)}} \tag{8.3}$$

[1] Actually York, in his Ph.D. dissertation York (1997), proposed to use center-of-mass velocities.

which is the velocity obtained from the contribution of the particles of the two bodies. One can refer to this velocity as the system velocity field. The key to the algorithm is not to consider pairwise interactions of bodies, but rather use a common frame (global quantities) so that contact of all bodies can be achieved at once. In Eq. (8.2), $\mathbf{n}_I^{(k)}$ is the normal vector of body k at node I. This is a crucial quantity in contact calculations and will be discussed in Sect. 8.1.4.

If Eq. (8.2) determines that the two bodies are getting closer to each other, the velocities $\tilde{\mathbf{v}}_I^{t+\Delta t,(k)}$ need to be corrected to get the final one $\mathbf{v}_I^{t+\Delta t,(k)}$. Otherwise they are kept unchanged. How to correct the grid velocities depends on the contact model (to be discussed in Sect. 8.1.1 for the case of non-slip contact and in Sect. 8.1.2 for the case of Coulomb frictional contact). Afterwards, the particle velocity, position and stresses are updated using the following equations

$$\begin{aligned}
\mathbf{a}_I^{t+\Delta t,(k)} &= \frac{\mathbf{v}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{t,(k)}}{\Delta t} \\
\mathbf{x}_p^{t+\Delta t,(k)} &= \mathbf{x}_p^{t,(k)} + \Delta t \sum_{p=1}^{n_p^{(k)}} \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t,(k)} \\
\mathbf{v}_p^{t+\Delta t,(k)} &= \mathbf{v}_p^{t,(k)} + \Delta t \sum_{p=1}^{n_p^{(k)}} \phi_I(\mathbf{x}_p^t)\mathbf{a}_I^{t,(k)}
\end{aligned} \tag{8.4}$$

where the first equation is to compute the corrected accelerations (only needed for contact nodes). Note that the stress update was skipped as it is standard and that the above discussion corresponds to the USL formulation.

After the frictionless and frictional contact models have been presented, the overall algorithm of the MPM contact will be presented in Sect. 8.1.5 for implementation. This model for contacts between deformable solids is simplified to the case of contact between a deformable solid and a rigid one in Sect. 8.1.6. Then, we provide a Matlab implementation of these contact algorithms in Sect. 8.1.7. Finally, we discuss the differences of MPM contact with other contact models in Sect. 8.1.8.

8.1.1 Contact Without Friction

The contact-release algorithm applied for a contact node I is quite simple. If contact is occurring, the nodal velocity is corrected so that the normal component of the body velocity is set equal to the normal component of the center-of-mass velocity. Otherwise, the two bodies move in their own velocities. Mathematically, one writes

$$\mathbf{v}_I^{t+\Delta t,(k)} = \begin{cases} \tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\mathrm{cm}}\right) \cdot \mathbf{n}_I^{(k)}\right] \mathbf{n}_I^{(k)} & \text{contact} \\ \tilde{\mathbf{v}}_I^{t+\Delta t,(k)} & \text{release} \end{cases} \tag{8.5}$$

If there is no friction between the bodies, then the above adjustment of the normal component of the body velocity, Eq. (8.5), is all that is required for contact treatment. The tangential component of the body velocity is unconstrained.

Remark 41 The fact that the normal component of the body velocity is equal to the normal component of the center-of-mass velocity can be proved as:

$$\begin{aligned}\mathbf{v}_I^{t+\Delta t,(k)} \cdot \mathbf{n}_I^{(k)} &= \tilde{\mathbf{v}}_I^{t+\Delta t,(k)} \cdot \mathbf{n}_I^{(k)} - \left\{\left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\mathrm{cm}}\right) \cdot \mathbf{n}_I^{(k)}\right] \mathbf{n}_I^{(k)}\right\} \cdot \mathbf{n}_I^{(k)} \\ &= \tilde{\mathbf{v}}_I^{t+\Delta t,(k)} \cdot \mathbf{n}_I^{(k)} - \left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\mathrm{cm}}\right) \cdot \mathbf{n}_I^{(k)} \\ &= \tilde{\mathbf{v}}_I^{t+\Delta t,(k)} \cdot \mathbf{n}_I^{(k)} - \tilde{\mathbf{v}}_I^{t+\Delta t,(k)} \cdot \mathbf{n}_I^{(k)} + \mathbf{v}_I^{\mathrm{cm}} \cdot \mathbf{n}_I^{(k)} = \mathbf{v}_I^{\mathrm{cm}} \cdot \mathbf{n}_I^{(k)}\end{aligned}$$

The fact that the tangential component of the corrected entity velocity is the same as the tangential component before the correction can be proved as:

$$\begin{aligned}\mathbf{v}_I^{t+\Delta t,(k)} - [\mathbf{v}_I^{t+\Delta t,(k)} \cdot \mathbf{n}_I^{(k)}]\mathbf{n}_I^{(k)} &= \tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\mathrm{cm}}\right) \cdot \mathbf{n}_I^{(k)}\right] \mathbf{n}_I^{(k)} - (\mathbf{v}_I^{\mathrm{cm}} \cdot \mathbf{n}_I^{(k)})\mathbf{n}_I^{(k)} \\ &= \tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - (\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} \cdot \mathbf{n}_I^{(k)})\mathbf{n}_I^{(k)}\end{aligned}$$

8.1.2 Contact with Coulomb Friction

In the case of frictional sliding, one needs to modify the tangential component of the grid velocity. To apply Coulomb friction, first calculate the force necessary to cause the bodies to stick completely. This force can be determined from the relative tangential velocity: $\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\mathrm{cm}}\right) - \left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\mathrm{cm}}\right) \cdot \mathbf{n}_I^{(k)}\right] \mathbf{n}_I^{(k)} = \mathbf{n}_I^{(k)} \times \left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\mathrm{cm}}\right) \times \mathbf{n}_I^{(k)}\right]$. This allows us to compute the stick force and then the Coulomb friction force. From that, the corrected velocity field is given by Bardenhagen et al. (2001) (see Sect. 8.1.3 for detail)

$$\mathbf{v}_I^{t+\Delta t,(k)} = \tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \left[\Delta \mathbf{v} \cdot \mathbf{n}_I^{(k)}\right] \left(\mathbf{n}_I^{(k)} + \mu' \mathbf{n}_I^{(k)} \times \boldsymbol{\omega}\right) \tag{8.6}$$

where

$$\Delta \mathbf{v} := \tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\mathrm{cm}}, \quad \boldsymbol{\omega} = \frac{\left[\Delta \mathbf{v} \times \mathbf{n}_I^{(k)}\right]}{\left|\left|\Delta \mathbf{v} \times \mathbf{n}_I^{(k)}\right|\right|}, \quad \mu' = \min\left[\mu, \frac{\left|\left|\Delta \mathbf{v} \times \mathbf{n}_I^{(k)}\right|\right|}{\Delta \mathbf{v} \cdot \mathbf{n}_I^{(k)}}\right] \tag{8.7}$$

where the symbol $||\bullet||$ denotes the norm of a vector i.e., $||\mathbf{a}|| = \sqrt{a_1^2 + a_2^2 + a_3^2}$, and $\mathbf{a} \times \mathbf{b} = ||\mathbf{a}||\,||\mathbf{b}|| \sin\theta \mathbf{n}$ is the cross product, θ is the angle between $\mathbf{a}$ and $\mathbf{b}$ in the

plane containing them, **n** is a unit vector perpendicular to the plane containing **a** and **b** in the direction given by the right-hand rule.

For 2D problems, Eq. (8.6) is explicitly given as

$$\begin{bmatrix} v_{xI}^{t+\Delta t,(k)} \\ v_{yI}^{t+\Delta t,(k)} \end{bmatrix} = \begin{bmatrix} \tilde{v}_{xI}^{t+\Delta t,(k)} \\ \tilde{v}_{yI}^{t+\Delta t,(k)} \end{bmatrix} - D \left(\begin{bmatrix} n_{xI}^{(k)} \\ n_{yI}^{(k)} \end{bmatrix} + \frac{\mu'}{||\mathbf{C}||} \begin{bmatrix} n_{yI}^{(k)} (\Delta v_x n_{yI}^{(k)} - \Delta v_y n_{xI}^{(k)}) \\ -n_{xI}^{(k)} (\Delta v_x n_{yI}^{(k)} - \Delta v_y n_{xI}^{(k)}) \end{bmatrix} \right) \tag{8.8}$$

with $\mu' = \min(\mu, ||\mathbf{C}|| / D)$, $D = \Delta \mathbf{v}_I \cdot \mathbf{n}_I^{(k)}$. This equation is obviously reduced to Eq. (8.5) for a frictionless contact if $\mu = 0$ and $D > 0$ since $\mu' = 0$.

8.1.3 Derivation

In order to apply the Coulomb friction one needs to compute the normal force corresponding to the corrected velocity in Eq. (8.5). This normal force is given by

$$\mathbf{f}_I^{\text{normal},(k)} = -\frac{m_I^{(k)}}{\Delta t} \left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}} \right) \cdot \mathbf{n}_I^{(k)} \right] \mathbf{n}_I^{(k)} \tag{8.9}$$

where $m_I^{(k)}$ is the nodal mass associated with body k.

The relative tangential velocity is given by

$$\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}} \right) - \left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}} \right) \cdot \mathbf{n}_I^{(k)} \right] \mathbf{n}_I^{(k)} = \mathbf{n}_I^{(k)} \times \left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}} \right) \times \mathbf{n}_I^{(k)} \right] \tag{8.10}$$

where use was made of the relation $\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) = (\mathbf{a} \cdot \mathbf{c})\mathbf{b} - (\mathbf{a} \cdot \mathbf{b})\mathbf{c}$ with $\mathbf{a}, \mathbf{b}, \mathbf{c}$ being arbitrary vectors.

From Eq. (8.10) we can get the stick force $\mathbf{f}_I^{\text{stick},(k)}$ as

$$\mathbf{f}_I^{\text{stick},(k)} = -\frac{m_I^{(k)}}{\Delta t} \mathbf{n}_I^{(k)} \times \left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}} \right) \times \mathbf{n}_I^{(k)} \right] \tag{8.11}$$

Next, we determine the friction force. This friction force equals the sticking force if the magnitude of the sticking force is small. That is, friction just balances the tangential force and prevents relative tangential motion, when the magnitude of the tangential force is small. For larger tangential forces, the magnitude of the friction force is proportional to the magnitude of the normal force. Limiting the frictional force to have magnitude less than the sticking force allows tangential slip between the contacting bodies since the applied frictional force is not sufficient to prevent relative tangential motion. Therefore, the Coulomb friction force is written as

$$\mathbf{f}_I^{\text{fric},(k)} = \frac{\mathbf{f}_I^{\text{stick},(k)}}{\left|\left| \mathbf{f}_I^{\text{stick},(k)} \right|\right|} \min \left(\mu \left|\left| \mathbf{f}_I^{\text{norm},(k)} \right|\right|, \left|\left| \mathbf{f}_I^{\text{stick},(k)} \right|\right| \right) \tag{8.12}$$

where μ denotes the coefficient of friction. Next, we compute the different terms in this expression.

Using Eq. (8.11) one can write

$$\frac{\mathbf{f}_I^{\text{stick},(k)}}{\left\|\mathbf{f}_I^{\text{stick},(k)}\right\|} = -\frac{\mathbf{n}_I^{(k)} \times \left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}}\right) \times \mathbf{n}_I^{(k)}\right]}{\left\|\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}}\right) \times \mathbf{n}_I^{(k)}\right\|} \tag{8.13}$$

And it can be shown

$$\begin{aligned}
\min\left(\mu\left\|\mathbf{f}_I^{\text{norm},(k)}\right\|, \left\|\mathbf{f}_I^{\text{stick},(k)}\right\|\right) &= \frac{m_I^k}{\Delta t}\left(\mu\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}}\right)\cdot\mathbf{n}_I^{(k)}, \left\|\mathbf{n}_I^{(k)} \times \left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}}\right) \times \mathbf{n}_I^{(k)}\right]\right\|\right) \\
&= \frac{m_I^k}{\Delta t}\left(\mu\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}}\right)\cdot\mathbf{n}_I^{(k)}, \left\|\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}}\right) \times \mathbf{n}_I^{(k)}\right\|\right) \\
&= \frac{m_I^k}{\Delta t}\left(\mu, \frac{\left\|\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}}\right) \times \mathbf{n}_I^{(k)}\right\|}{\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}}\right)\cdot\mathbf{n}_I^{(k)}}\right)\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}}\right)\cdot\mathbf{n}_I^{(k)}
\end{aligned} \tag{8.14}$$

where use was made of Eqs. (8.9) and (8.11) and the fact that $(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}}) \cdot \mathbf{n}_I^{(k)} > 0$ (as contact is happening).

Upon substitution of Eqs. (8.13) and (8.14) into Eq. (8.12), one obtains

$$\begin{aligned}
\mathbf{f}_I^{\text{fric},(k)} &= -\frac{m_I^k}{\Delta t}\frac{\mathbf{n}_I^{(k)} \times \left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}}\right) \times \mathbf{n}_I^{(k)}\right]}{\left\|\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}}\right) \times \mathbf{n}_I^{(k)}\right\|}\mu'\left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}}\right)\cdot\mathbf{n}_I^{(k)}\right] \\
&= -\frac{m_I^k}{\Delta t}\mu'\left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}}\right)\cdot\mathbf{n}_I^{(k)}\right]\left(\mathbf{n}_I^{(k)} \times \boldsymbol{\omega}\right)
\end{aligned} \tag{8.15}$$

where μ' is given by

$$\mu' = \min\left[\mu, \frac{\left\|\left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}}\right) \times \mathbf{n}_I^{(k)}\right]\right\|}{\left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}}\right)\cdot\mathbf{n}_I^{(k)}\right]}\right] \tag{8.16}$$

and the unit vector $\boldsymbol{\omega}$

$$\boldsymbol{\omega} = \frac{\left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}}\right) \times \mathbf{n}_I^{(k)}\right]}{\left\|\left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\text{cm}}\right) \times \mathbf{n}_I^{(k)}\right]\right\|} \tag{8.17}$$

The correction term for the tangential velocity is determined from the friction force given in Eq. (8.15). It is given by

$$\left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\mathrm{cm}}\right) \cdot \mathbf{n}_I^{(k)}\right] \mu' \mathbf{n}_I^{(k)} \times \boldsymbol{\omega} \tag{8.18}$$

Combining this term and Eq. (8.5) the final corrected velocity can be written as where both normal and tangential components of the body velocity was corrected

$$\mathbf{v}_I^{t+\Delta t,(k)} = \tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \left[\left(\tilde{\mathbf{v}}_I^{t+\Delta t,(k)} - \mathbf{v}_I^{\mathrm{cm}}\right) \cdot \mathbf{n}_I^{(k)}\right] \left(\mathbf{n}_I^{(k)} + \mu' \mathbf{n}_I^{(k)} \times \boldsymbol{\omega}\right) \tag{8.19}$$

8.1.4 Calculation of Normal Vector

Determining the normal vector at grid nodes for each body $\mathbf{n}_I^{(k)}$ is necessary to complete the contact algorithm. The way it is calculated has a crucial impact on the accuracy of the results (Lemiale et al. 2010; Nairn 2007b).

The usual practice for finding the normal is to handle each body separately with relation to the system velocities. Thus, the normal is found from the mass gradient of the material under consideration. For each entity, the particle mass is interpolated to the element centers, $\mathbf{x}_c$ and divided by the element volume V_e, to obtain a density ρ_c. The gradient of ρ_c evaluated at the grid nodes provides the normal direction at the surface of each body (Sulsky and Brackbill 1991; York 1997; Bardenhagen et al. 2000).

The cell-centered density is defined by

$$\rho_c = \frac{1}{V_e} \sum_{p=1}^{n_p} m_p S^2(\mathbf{x}_p - \mathbf{x}_c) \tag{8.20}$$

where S^2 are bi-quadratic B-spline functions. In two dimensions, they are defined by $S^2 = S^x(x)S^y(y)$ where the one-dimensional quadratic B-spline function is given by

$$S^x(x) = \begin{cases} \dfrac{1}{2h_x^2}x^2 + \dfrac{3}{2h_x}x + \dfrac{9}{8}, & -\dfrac{3}{2h_x} \le x \le -\dfrac{1}{2h_x} \\ -\dfrac{1}{h_x^2}x^2 + \dfrac{3}{4}, & -\dfrac{1}{2h} \le x \le \dfrac{1}{2h_x} \\ \dfrac{1}{2h_x^2}x^2 - \dfrac{3}{2h_x}x + \dfrac{9}{8}, & \dfrac{1}{2h_x} \le x \le \dfrac{3}{2h_x} \\ 0, & \text{otherwise} \end{cases} \tag{8.21}$$

where h_x denotes the cell spacing in the x direction. Figure 8.2 depicts some quadratic B-splines on a one-dimensional mesh. As it can be seen, in 1D, each particle con-

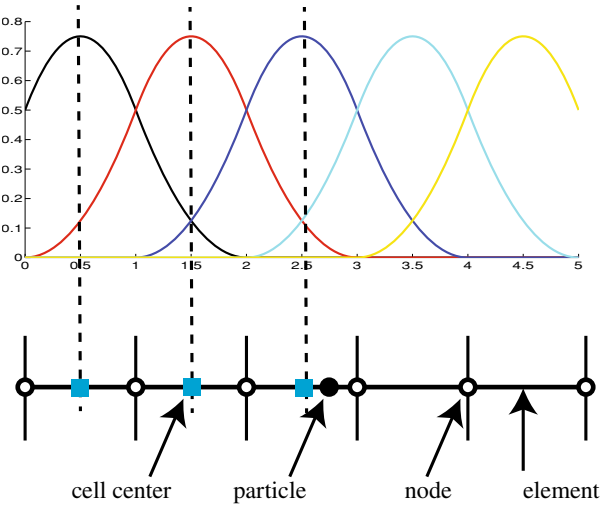


Fig. 8.2 Quadratic B-spline functions used in defining cell center density (de Vaucorbeil et al. 2020)

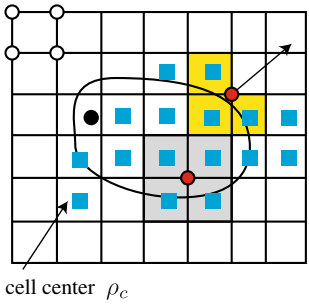


Fig. 8.3 Computation of grid normal using Eq. (8.22)

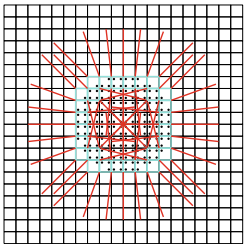
tributes to three cells–the one it which it is located and the two neighboring cells. In 2D and 3D, each particle contributes to 9 and 27 cells, respectively.

The grid normal vector is then given by

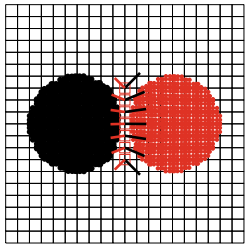
$$\mathbf{n}_I = \sum_c \nabla\phi_I(\mathbf{x}_c)\rho_c, \quad \mathbf{n}_I = \frac{\mathbf{n}_I}{||\mathbf{n}_I||} \tag{8.22}$$

where $\nabla\phi_I$ denotes the gradient of the MPM weighting functions. Note that the sum is performed on the cells that have the node under consideration in their connectivity, cf. Fig. 8.3. This way of computing the normals is similar to SPH (Randles and Libersky 1996).

Example on calculation of grid normals. Let us consider a circular disk which is represented by particles and a background grid. We are going to use Eqs. (8.20) and



(a) Normals of a circle



(b) Violation of collinearity of $\mathbf{n}$

Fig. 8.4 An example of normal vectors (red lines) defined at boundary nodes for a circular disk (**a**). The disk is represented by unconnected points or particles (black dots). Cyan squares are the boundary elements which are defined as those cut by the circle. Also shown are unneeded grid normals. Violation of collinearity of normal vectors in the MPM (**b**)

(8.22) to compute the normal vectors at boundary grid nodes. First, the density at the cell centers are computed using Eq. (8.20). Next, we identify boundary elements which are those cut by the circle and are not empty. Finally, we loop over the boundary elements and their nodes and use Eq. (8.22) to compute the grid normals. Figure 8.4 shows the result. The most important result is that the normal vectors are not collinear which leads to the non-conversation of momentum (Bardenhagen et al. 2001). This occurred because contacts are not handled at the true contact points but rather at the grid nodes which do not locate on either body. In the literature some tricks have been proposed; for example one trick is to use the average of the two normals, the other is to use the normal of the body which is more rigid. In Fig. 8.8 we demonstrate the importance of having good normals for modeling contacts.

Remark 42 Recent works adopts a simpler method to compute the normals (Lemiale et al. 2010; Huang et al. 2011; Homel and Herbold 2017). In this method, the normals are simply calculated from the gradient of the particle mass:

$$\mathbf{n}_I = \sum_p \nabla \phi_I(\mathbf{x}_p) m_p, \quad \mathbf{n}_I = \frac{\mathbf{n}_I}{||\mathbf{n}_I||} \tag{8.23}$$

There is no need to define the cell-centered density. We have tested both ways (Eqs. (8.22) and (8.23)) and found that they provide identical normals.

8.1.5 Algorithm

The previously presented contact algorithm can be incorporated into the standard MPM algorithm quite straightforwardly. Algorithm 13 is the algorithm for non-friction contact described in Sect. 8.1.1. One just need to modify line 21 for Coulomb

friction. Even though the USF is shown, modification for USL is easy. One modification to the standard MPM code is that each grid node now contains the body velocity, mass and the system mass and velocity.

Algorithm 13 Solution procedure of an explicit contact MPM (USF).

Solve momentum equations for body b
  Mapping from particles to nodes (P2G)
    Compute nodal mass $m_I^{t,(b)} = \sum_p \phi_I(\mathbf{x}_p^t) m_p$
    Compute nodal momentum $(m\mathbf{v})_I^{t,(b)} = \sum_p \phi_I(\mathbf{x}_p^t)(m\mathbf{v})_p^t$
    Compute body nodal velocities $\mathbf{v}_I^{t,(b)} = (m\mathbf{v})_I^{t,b} / m_I^{t,(b)}$
    Compute gradient velocity $\mathbf{L}_p^t = \sum_I \nabla\phi_I(\mathbf{x}_p)\mathbf{v}_I^{t,(b)}$
    Compute gradient deformation tensor $\mathbf{F}_p^t = (\mathbf{I} + \mathbf{L}_p^t \Delta t)\mathbf{F}_p^t$
    Update volume $V_p^t = \det \mathbf{F}_p^t V_p^0$
    Update stresses $\boldsymbol{\sigma}_p^{t+\Delta t} = \boldsymbol{\sigma}_p^t + \Delta\boldsymbol{\sigma}_p$
    Compute external force $\mathbf{f}_I^{\text{ext},t,(b)}$, internal force $\mathbf{f}_I^{\text{int,t,(b)}} = -\sum_{p=1}^{n_p} V_p^t \boldsymbol{\sigma}_p^{t+\Delta t} \nabla\phi_I(\mathbf{x}_p^t)$
    Compute nodal force $\mathbf{f}_I^{t,(b)} = \mathbf{f}_I^{\text{ext},t,(b)} + \mathbf{f}_I^{\text{int},t,(b)}$
  end
  Update the body momenta $(m\mathbf{v})_I^{t+\Delta t,(b)} = (m\mathbf{v})_I^{t,(b)} + \mathbf{f}_I^{t,(b)} \Delta t$
  Update the system momenta $(m\mathbf{v})_I^{t+\Delta t} = (m\mathbf{v})_I^{t+\Delta t} + (m\mathbf{v})_I^{t+\Delta t,(b)}$
  Update the system mass $m_I^t = m_I^t + m_I^{t,(b)}$
end
Correct velocity for contact nodes
  for contact node I of body b **do**
    Compute normal to b, $\mathbf{n}_I^{(b)}$
    Retrieve center of mass velocity $\mathbf{v}_I^{\text{cm}} = (m\mathbf{v})_I^{t+\Delta t} / m_I^t$
    Check contact or release $\alpha = \left(\mathbf{v}_I^{t+\Delta t,(b)} - \mathbf{v}_I^{\text{cm}}\right) \cdot \mathbf{n}_I^{(b)}$
    If $\alpha \geq 0$: $\mathbf{v}_I^{t+\Delta t,(b)} = \mathbf{v}_I^{t+\Delta t,(b)} - \alpha \mathbf{n}_I^{(b)}$
    If $\alpha < 0$: keep $\mathbf{v}_I^{t+\Delta t,(b)}$
    Compute nodal acceleration $\mathbf{a}_I^{t+\Delta t,(b)} = (1/\Delta t)(\mathbf{v}_I^{t+\Delta t,(b)} - \mathbf{v}_I^{t,(b)})$
  end for
end
Update particle positions and velocities (G2P)
  Update particle velocities $\mathbf{v}_p^{t+\Delta t,(b)} = \mathbf{v}_p^{t,(b)} + \Delta t \sum_I \phi_I(\mathbf{x}_p^t)\mathbf{a}_I^{t+\Delta t,(b)}$
  Update particle positions $\mathbf{x}_p^{t+\Delta t,(b)} = \mathbf{x}_p^{t,(b)} + \Delta t \sum_I \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t,(b)}$
end

Remark 43 To the best of our knowledge MUSL stress update has not yet been used with frictional contacts. We anticipate that the reason for this is efficiency concerns. In the MUSL, one would need to do the contact treatment twice.

8.1.6 Contact Between a Deformable Solid and a Rigid Wall

In this section, the general contact algorithm previously presented for deformable solids is specialized to the case of contact between a solid and a rigid body. Rigid bodies can be stationary such as rigid walls or moving rigid tools such as indenters in machining processes. Some examples are shown in Fig. 8.5. These rigid bodies are discretized by the so-called rigid particles, cf. Fig. 8.6.

Let us denote the velocity of a rigid body by $\mathbf{v}^{\mathrm{r}}$, which is apparently zero for rigid walls and $\mathbf{v}^{\mathrm{r}}(t)$ for moving bodies. Since a rigid body has an infinite mass, the center-of-mass velocity defined in (8.3) is actually $\mathbf{v}^{\mathrm{r}}(t)$. Therefore, the contact algorithm previously presented also applies with the following minor modifications:

- in the P2G step, for rigid bodies, one just computes the normal vectors (i.e., no mass, momentum, forces projection);
- the center-of-mass velocity is the velocity of the rigid body;
- in the step of velocity correction, skip rigid bodies;
- update the position of moving rigid bodies.

8.1.7 Matlab Implementation

Herein, we present a simple, easy to understand implementation of the previously presented MPM contact formulation. The implementation is general as it can handle

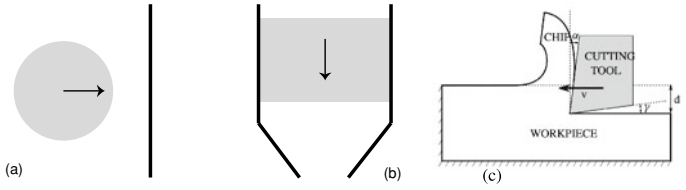


Fig. 8.5 Contact between a deformable body and rigid walls: (**a**) moving ball impacting on a rigid wall, (**b**) silo discharging problem and (**c**) metal cutting

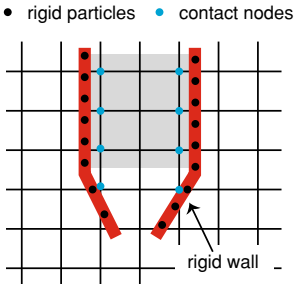


Fig. 8.6 Rigid wall is represented by rigid particles

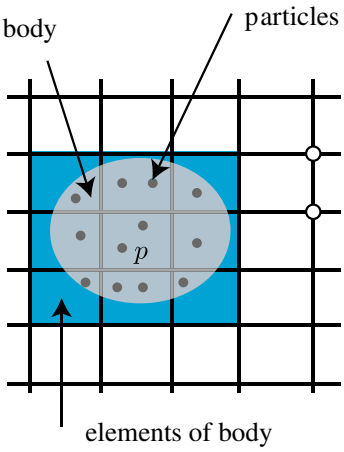


Fig. 8.7 Data structure of a multiple body MPM: each body contains (i) the particles, (ii) the elements the body occupy and (iii) the indices of the particles each element contain. Note that the data structure evolves in time as the body moves

contacts of N bodies but at any grid node, it only allows contact of two bodies. Furthermore, frictional self-contacts are not coded. No-slip self contacts are, certainly, included by the built-in contact of the MPM.

Data structures. For multiple bodies simulations it is more convenient to have a proper data structure for the bodies, the particles, the nodes and the cells. Figure 8.7 shows the data structures to be used. Each body contains (i) the particles making up the body (actually the particle information such as position, mass, stress etc.), (ii) the elements (cells) which contain particles of the body under consideration, (iii) the indices of particles in each element and (iv) the indices of the nodes of the cells. Listing 8.1 gives an implementation to create these data structures for an example of two bodies. Listing 8.2 is the code snippet used to build the elements and nodes of all the bodies in the system for 2D (extension of the code given in Listing 6.1 to multiple bodies). This is called at the beginning of the simulation and at the end of every time steps. The grid data structure is created using Listing 8.3. Arrays `nmass` and `nmomentum` store the grid mass and momentum for a body. They are overwritten when the code iterates to the next body. Arrays `nmassS` and `nmomentumS` store the grid mass and momentum for the system. They store the accumulation of arrays `nmass` and `nmomentum` for all bodies. Array `nvelo` stores the updated $(t + \Delta t)$ grid velocities for all bodies and the system velocity–the first two columns are the grid velocities of body 1 and so on. Array `nvelo0` stores the previous (t) grid velocities for all bodies. Array `nacce` stores the grid accelerations for all bodies–the first two columns are the grid accelerations of body 1 and so on.

Listing 8.1 Matlab code to build the body data structure.

```
% volume, mass, coord have been computed
body1.volume  = volume;                          % volume
body1.volume0 = volume;   % pCount: no of particles per body
body1.mass    = mass;                            % mass
body1.coord   = coord;                           % position
body1.deform  = repmat([1 0 0 1],pCount,1); % gradient defor. F
body1.stress  = zeros(pCount,3);                 % stress
body1.strain  = zeros(pCount,3);                 % strain
body1.velo    = ones(pCount,2)*v;                % velocity
% the same is done for particles of body2
% store them into bodies
bodies    = cell(2,1);
bodies{1} = body1;
bodies{2} = body2;
```

Listing 8.2 Matlab code to generate element/particle lists for each body.

```
for ib=1:length(bodies)
    body       = bodies{ib};
    elems      = ones(length(body.volume),1);
    % for each particle of body "ib", find element contains it
    for ip=1:length(body.volume)
        x = body.coord(ip,1); y = body.coord(ip,2);
        e = floor(x/deltax) + 1 + numx2*floor(y/deltay);
        elems(ip) = e;
    end
    bodies{ib}.elements = unique(elems);
    % for each element, find particles it contains
    mpoints = cell(elemCount,1);
    for ie=1:elemCount
        id  = find(elems==ie);
        mpoints{ie}=id;
    end
    bodies{ib}.mpoints  = mpoints;
end
```

Listing 8.3 Matlab code to build grid data structures

```
nmassS     = zeros(nodeCount,1);  % nodal mass vector of the system
nmomentumS= zeros(nodeCount,2);  % nodal momentum vector of the system
nmass      = zeros(nodeCount,1);  % nodal mass vector of each body
nmomentum = zeros(nodeCount,2);  % nodal momentum vector of each body
niforce    = zeros(nodeCount,2);  % nodal internal force of each body
neforce    = zeros(nodeCount,2);  % nodal external force of each body
% nodal velocities (body1,body2,...,center of mass)
nvelo      = zeros(nodeCount,2*(bodyCount+1));
nvelo0     = nvelo;
nacce      = zeros(nodeCount,2*bodyCount);
```

Solution. Listing 8.4 gives the code used to compute $\tilde{\mathbf{v}}_I^{t+\Delta t,(k)}$ for all bodies i.e., body velocities without contact. Detection of contact nodes and the correction of its velocities are next performed using the code given in Listing 8.5. Finally particle update is achieved using Listing 8.6. Note that the code snippets are not complete in the sense that some parts irrelevant to contact (identical to a standard MPM) are skipped for brevity. Furthermore, the presentation is for the USL formulation. Again some C++ operators such as += and –= are utilized to gain spaces: In C++, a += b means $a = a + b$.

Listing 8.4 Matlab code for updating momentum for all bodies without contact

```
nvelo(:)      = 0; nmassS(:)      = 0; nmomentumS(:) = 0;
for ib=1:bodyCount %loop over bodies (update nodal momenta w/o contact)
    %% reset grid data (body contribution)
    nmass(:)      = 0; nmomentum(:) = 0;
    niforce(:)    = 0; neforce(:)   = 0;
    body  = bodies{ib}; elems  = body.elements; mpoints  = body.mpoints;
    for ie=1:length(elems) % loop over computational cells or elements
        e      = elems(ie);
        esctr = element(e,:);        % element connectivity
        enode = node(esctr,:);       % element node coords
        mpts  = mpoints{e};          % particles inside element e
        for p=1:length(mpts) % loop over particles
            pid     = mpts(p);
            xp      = body.coord(pid,:); Mp=body.mass(pid); ...
            stress =  bodies{ib}.stress(pid,:);
            for i=1:length(esctr) % loop over nodes of element "ie"
                id     = esctr(i);
                dNIdx = dNdx(i,1); dNIdy = dNdx(i,2);
                nmass(id)         = nmass(id)         + N(i)*Mp;
                nmomentum(id,:) = nmomentum(id,:) + N(i)*Mp*vp;
                niforce(id,1)   -= Vp*(stress(1)*dNIdx + stress(3)*dNIdy);
            end
        end
    end
    activeNodes = bodies{ib}.nodes;
    nvelo(activeNodes,2*ib-1) = nmomentum(activeNodes,1).*massInv;
    nvelo(activeNodes,2*ib)   = nmomentum(activeNodes,2).*massInv;
    % update nodal momenta (for body ib)
    % body velocity (v_I^{t+\Delta t,(k)} and acceleration
    nmomentum(activeNodes,:) += niforce(activeNodes,:)*dtime;
    massInv = 1./nmass(activeNodes);
    nvelo(activeNodes,2*ib-1) = nmomentum(activeNodes,1).*massInv;
    nvelo(activeNodes,2*ib)   = nmomentum(activeNodes,2).*massInv;
    nacce(activeNodes,2*ib-1) = niforce(activeNodes,1).*massInv;
    nacce(activeNodes,2*ib  ) = niforce(activeNodes,2).*massInv;
    % store system momentum and mass
    nmomentumS(activeNodes,:) +=  nmomentum(activeNodes,:);
    nmassS    (activeNodes  ) +=  nmass(activeNodes);
end
```

Listing 8.5 Matlab code to detect contact nodes and correct velocities (frictionless contact).

```
%% detection of contact nodes
contactNodes = intersect(bodies{1}.nodes,bodies{2}.nodes);
if ~isempty(contactNodes)
    for ib=1:bodyCount
        body       = bodies{ib};
        nodes      = body.nodes;
        [cellDensity,normals] = computeGridNormal(grid,body);
        bodies{ib}.normals = normals;
    end
end
%% correct contact node velocities
for ib=1:bodyCount
    for in=1:length(contactNodes)
        id         =  contactNodes(in);
        velo1      = nvelo(id,2*ib-1:2*ib);          % body velocity
        velocm     = nmomentumS(id,:)/nmassS(id); % system velocity
        nl         = bodies{ib}.normals(id,:); nl = nl / norm(nl);
        alpha      = dot(velo1 - velocm, nl);
        if ( alpha >= 0 )
          nvelo(id,2*ib-1:2*ib) = velo1 - alpha*nl;
          nacce(id,2*ib-1:2*ib) = (1/dtime)*( nvelo(id,2*ib-1:2*ib) - ...
                                              nvelo0(id,2*ib-1:2*ib) );
        end
    end
end
```

Listing 8.6 Matlab code to update particles.

```
for ib=1:bodyCount
    body  = bodies{ib}; elems = body.elements; mpoints = body.mpoints;
    % loop over computational cells or elements
    for ie=1:length(elems)
        e      = elems(ie);
        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);
            xp    = body.coord(pid,:); ...
            Lp    = zeros(2,2);
            for i=1:length(esctr)
                id = esctr(i);
                vl = nvelo(id,2*ib-1:2*ib); % body corrected velocity
                al = nacce(id,2*ib-1:2*ib); % body corrected acceleration
                vp  = vp  + dtime * N(i)*al;
                xp  = xp  + dtime * N(i)*vl;
                Lp  = Lp + vl'*dNdx(i,:);
            end
            bodies{ib}.velo(pid,:) = vp;
            bodies{ib}.coord(pid,:)= xp;
            % update stress last
        end
    end
end
```

8.1.8 Differences of MPM Contacts with Other Contacts

Classically, contacts are treated directly by handling the contacts on the contact surface i.e., the common surface of two contacting solids. As contacts are mostly solved using finite elements, we are talking actually about contact between two finite element meshes. It is where algorithms such as node-to-segment, segment-to-segment etc. were developed, see Wriggers (2006) for a nice discussion. The most time consuming step in these direct contact algorithms is the *contact search* where one needs to find which segment is in contact with which node (in the context of segment-to-node algorithm). We do not go into details on how contact is actually treated such as penalty or Lagrangian multiplier or mortar method.

The expensive contact search can be avoided using indirect contact models. The idea is to mesh the empty space between the solids. Oliver et al. (2009) presented a contact domain method where the contact surface is regularized by a contact domain. Wriggers et al. (2013), on the other hand, discretized the entire empty space with finite elements (they called it a third medium) and contact is implicitly treated by the deformation of this third medium. In computer graphics, a similar idea was pursued: Müller et al. (2015) presented an air mesh method for contact.

8.1.9 Final Remarks

The MPM can handle contacts quite efficiently as there is no need to find the contact surfaces and master/slave nodes commonly used in the FEM. We have just presented the most basic algorithm. There are refinements such as how to improve the normal vector calculation, see e.g. Lemiale et al. (2010); Nairn et al. (2018), and better contact detection criteria, see Bardenhagen et al. (2001).

To demonstrate the important role of the normal vectors in contact calculations, we consider the problem of a cylinder rolling down an inclined plane (Fig. 8.8). In this case where one of two contacting bodies is more rigid than the other, the best option is to use the normal vector of the rigid body (assume it is body (2)) i.e., $\mathbf{n}_I^{(1)} = -\mathbf{n}$ and $\mathbf{n}_I^{(2)} = \mathbf{n}$, where $\mathbf{n} = (0, 1)$ for this particular rolling example.

8.2 Contacts in the TLMPM

With the removal of numerical fracture, TLMPM lost the inherent ability that the MPM has to simulate no slip, no penetration contacts out-of-the-box. Indeed, in the TLMPM, each solid has its own background grid which covers only the area where the solid lies in the reference configuration (Fig. 8.9). As multiple solids do not share a common grid, the algorithm is unable to identify when contacts occur.

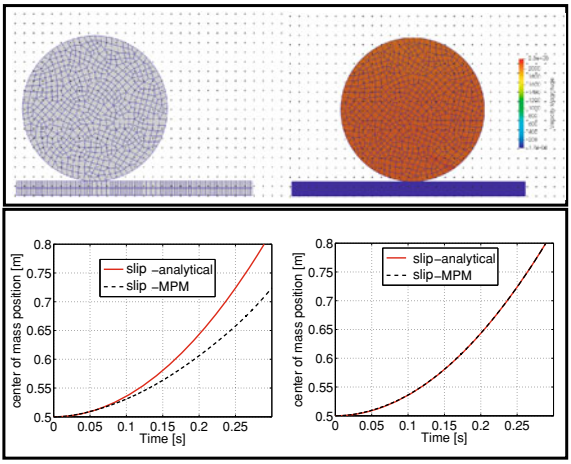


Fig. 8.8 Vital role of accurate normal vectors in contact modelling: a cylinder rolling down an inclined (θ) rigid plane, see Sect. 8.4.4 for detail. Top: GPIC set up and plot of the displacement. Bottom: center-of-mass position in time for $\theta = \pi/3$ and friction coefficient $\mu = 0.3$. The result on the left was obtained using the normal vector of the disk and the one on the right was obtained using the normal vector of the rigid plane

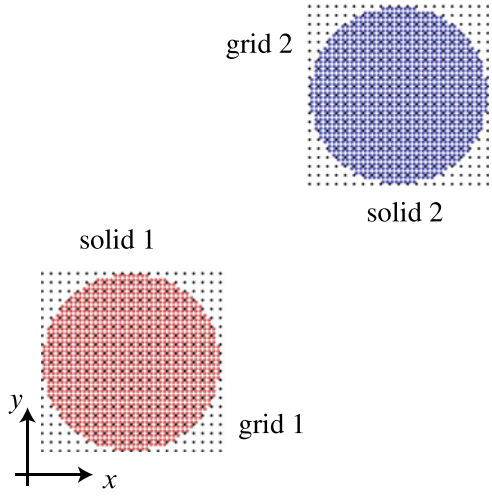


Fig. 8.9 Multiple grids in the TLMPM: each solid is associated with its own grid (de Vaucorbeil et al. 2020)

The TLMPM is not the only particle based method that needs addition numerical treatment to handle contact. This is also the case of Smooth Particle Hydrodynamics, see for instance Campbell et al. (2000). Contact algorithms have long been available for this method. The contact algorithm developed here is directly adapted from what is done in SPH.

To overcome this limitation and following the pinball contact algorithm of Belytschko and Neal (1991)—which was inspired by the discrete element method de Vaucorbeil and Nguyen (2021b) performed contact detection at the particle level. Each particle is assumed to be a sphere of radius $R_p = {}^1\!/_2 V_P^{1/3}$ where V_p is the volume attached to particle p. Contact happens when penetration exists between two particles p and q, i.e.,

$$\delta := R_p + R_q - ||\mathbf{x}_q - \mathbf{x}_p|| \geq 0 \tag{8.24}$$

where R_p and R_q are the radii of particles p and q, respectively.

Based on the non-penetration condition, we compute the contact forces with and without friction (Sect. 8.2.1). The resulting algorithm is presented in Sect. 8.2.2.

8.2.1 Enforcing Non-penetration

By just applying a contact force, we are not checking that the non-penetration condition of a normal contact is satisfied. Let's consider the problem where particles p and q from solids 1 and 2, respectively, enter in contact at step n (total time t). The penetration is checked at the beginning of the step and reads:

$$\delta^t = R_p + R_q - ||\mathbf{x}_q^t - \mathbf{x}_p^t|| \tag{8.25}$$

The idea is to find the contact forces $\mathbf{F}_{pq}$ and $\mathbf{F}_{qp}$ applied by the particle q onto p, and vice-versa (Fig. 8.10) such that if applied directly, the penetration at the end of the time step $\delta^{t+\Delta t}$ is zero. The position and velocity of the particles would therefore be:

$$\mathbf{x}_p^{t+\Delta t} = \mathbf{x}_p^t + \Delta t^2 \frac{\mathbf{F}_{pq}}{m_p} \tag{8.26}$$

$$\mathbf{x}_q^{t+\Delta t} = \mathbf{x}_q^t + \Delta t^2 \frac{\mathbf{F}_{qp}}{m_q} \tag{8.27}$$

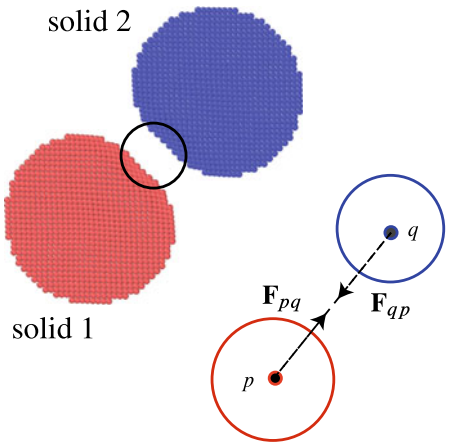


Fig. 8.10 Contact forces between two contacting particles p and q, respectively from solid 1 and 2, in the TLMPM (de Vaucorbeil and Nguyen 2021b)

where m_p and m_q are the masses of the particles p and q, respectively and Δt is the time step.

Upon substitution of Eqs. (8.26) and (8.27) into Eq. (8.25), and enforcing conservation of local linear momentum by using $\mathbf{F}_{pq} = -\mathbf{F}_{qp}$, we obtain the non-penetration condition as

$$\delta^{t+\Delta t} = R_p + R_q - \left\| \mathbf{x}_q^t - \mathbf{x}_p^t - \Delta t^2 \mathbf{F}_{pq} \frac{m_p + m_q}{m_p m_q} \right\| = 0 \tag{8.28}$$

of which the solution is:

$$\mathbf{F}_{pq} = \frac{1}{\Delta t^2} \frac{m_p m_q}{m_p + m_q} \left(1 - \frac{R_p + R_q}{\|\mathbf{x}_{pq}^t\|} \right) \mathbf{x}_{pq}^t \tag{8.29}$$

Force given in Eq. (8.29) is the contact force for a contact without friction. The effect of friction can be straightforwardly included by adding a tangential force such that

$$\mathbf{F}_{pq} = \frac{1}{\Delta t^2} \frac{m_p m_q}{m_p + m_q} \left(1 - \frac{R_p + R_q}{\|\mathbf{x}_{pq}^t\|} \right) \left[\mathbf{x}_{pq}^t + \mu \|\mathbf{x}_{pq}^t\| \mathbf{m}_{pq}^t \right] \tag{8.30}$$

where μ is the friction coefficient, $\mathbf{m}_{pq}^t = \mathbf{v}_{pq}^{t,T} / \|\mathbf{v}_{pq}^{t,T}\|$ is the direction of the difference of velocity between particles p and q in the plane normal to $\mathbf{x}_{pq}^t$, and with

$$\mathbf{v}_{pq}^{t,T} = \mathbf{v}_{pq}^t - \frac{(\mathbf{x}_{pq}^t \cdot \mathbf{v}_{pq}^t)\mathbf{x}_{pq}^t}{\|\mathbf{x}_{pq}^t\|^2} \tag{8.31}$$

where $\mathbf{v}_{pq}^t = \mathbf{v}_q^t - \mathbf{v}_p^t$.

8.2.2 Complete Algorithm

Check for contacts occurs at the beginning of each time step. Once contact occurs between particles p and q, i.e., $\delta^t \geq 0$, a force $\mathbf{F}_{pq}$ is applied at the center of particle p and the opposite force $\mathbf{F}_{qp} = -\mathbf{F}_{pq}$ is applied at the center of particle q. These forces are added to the external forces of both solids:

$$\mathbf{f}_I^{\text{ext},t} = \sum_p \phi_I(\mathbf{x}_p) m_p \mathbf{b}(\mathbf{x}_p) + \frac{1}{m_I} \sum_p \sum_q \phi_I(\mathbf{x}_p) m_p \mathbf{F}_{pq} \tag{8.32}$$

$$\mathbf{f}_J^{\text{ext},t} = \sum_q \phi_J(\mathbf{x}_q) m_q \mathbf{b}(\mathbf{x}_q) + \frac{1}{m_J} \sum_q \sum_p \phi_J(\mathbf{x}_q) m_q \mathbf{F}_{qp} \tag{8.33}$$

where I and J correspond to nodes of solid 1 and 2, respectively. The rest of the algorithm is identical to that of the TLMPM. Therefore the complete algorithm of TLMPM with contact is as given in Algorithm 14.

Remark 44 From the double sum in Eqs. (8.32) and (8.33), one can see that the computation time of TLMPM with contacts scales as $\mathcal{O}(n^2)$. This is the main disadvantages of this algorithm compared to the built-in no-splip no-penetration contact in the ULMPM. Computation time could be reduced by performing the double sums in Eqs. (8.32) and (8.33) only over the surface particles. However, this would require to detect the surface particles, which is not the subject of this paper.

Algorithm 14 Solution procedure of explicit TLMPM (MUSL) with contacts.

Initialization
  Set up particle data: $\mathbf{X}_p, \mathbf{v}_p^0, \boldsymbol{\sigma}_p^0, \mathbf{F}_p^0, V_p^0, m_p, \rho_p^0$
  Compute nodal mass $m_I = \sum_p \phi_I(\mathbf{X}_p) m_p$
  Compute and store weighting and gradient $\phi_I(\mathbf{X}_p)$ and $\nabla_0 \phi_I(\mathbf{X}_p)$
end
while $t < t_f$ **do**
  Reset grid quantities: $(m\mathbf{v})_I^t = \mathbf{0}, \mathbf{f}_I^{\text{ext},t} = \mathbf{0}, \mathbf{f}_I^{\text{int},t} = \mathbf{0}$
  Compute contact forces $\mathbf{f}_p^{c,t}$
  Mapping from particles to nodes (P2G)
    Compute nodal momentum $(m\mathbf{v})_I^t = \sum_p \phi_I(\mathbf{X}_p)(m\mathbf{v})_p^t$
    Compute external force $\mathbf{f}_I^{\text{ext},t} = \sum_p \phi_I(\mathbf{X}_p) \left(m_p \mathbf{b}_p^t + \mathbf{f}_p^{c,t} \right)$
    Compute internal force $\mathbf{f}_I^{\text{int},t} = -\sum_{p=1}^{n_p} V_p^0 \boldsymbol{P}_p^t \nabla_0 \phi_I(\mathbf{X}_p)$
    Compute nodal force $\mathbf{f}_I^t = \mathbf{f}_I^{\text{ext},t} + \mathbf{f}_I^{\text{int},t}$
  end
  Update the momenta $(m\tilde{\mathbf{v}})_I^{t+\Delta t} = (m\mathbf{v})_I^t + \mathbf{f}_I^t \Delta t$
  Fix Dirichlet nodes I e.g. $(m\tilde{\mathbf{v}})_I^{t+\Delta t} = \mathbf{0}$ and $(m\mathbf{v})_I^t = \mathbf{0}$
  Update particle velocities and grid velocities (double mapping)
    Get nodal velocities $\tilde{\mathbf{v}}_I^{t+\Delta t} = (m\tilde{\mathbf{v}})_I^{t+\Delta t} / m_I^t$
    Update particle velocities $\mathbf{v}_p^{t+\Delta t} = \alpha \left(\mathbf{v}_p^t + \sum_I \phi_I(\mathbf{X}_p) \left[\tilde{\mathbf{v}}_I^{t+\Delta t} - \mathbf{v}_I^t \right] \right) + (1 - \alpha) \sum_I \phi_I(\mathbf{X}_p) \tilde{\mathbf{v}}_I^{t+\Delta t}$
    Update grid velocities $(m\mathbf{v}_I)^{t+\Delta t} = \sum_p \phi_I(\mathbf{X}_p)(m\mathbf{v})_p^{t+\Delta t}$
    Fix Dirichlet nodes $(m\mathbf{v})_I^{t+\Delta t} = \mathbf{0}$
  end
  Update particle (G2P)
    Compute $\dot{\mathbf{F}}_p^{t+\Delta t} = \sum_I \nabla_0 \phi_I(\mathbf{X}_p) \mathbf{v}_I^{t+\Delta t}$
    Updated gradient deformation tensor $\mathbf{F}_p^{t+\Delta t} = \mathbf{F}_p^t + \Delta t \dot{\mathbf{F}}_p^{t+\Delta t}$
    Velocity gradient $\mathbf{L}_p^{t+\Delta t} = \dot{\mathbf{F}}_p^{t+\Delta t} (\mathbf{F}_p^{t+\Delta t})^{-1}$
    Update stresses $\boldsymbol{\sigma}_p^{t+\Delta t} = \boldsymbol{\sigma}_p^t + \Delta \boldsymbol{\sigma}_p$
    Covert stresses to 1st PK stresses $\mathbf{P}_p^{t+\Delta t} = J \boldsymbol{\sigma}_p^{t+\Delta t} (\mathbf{F}_p^{t+\Delta t})^{-\mathrm{T}}$
    Update particle positions $\mathbf{x}_p^{t+\Delta t} = \mathbf{x}_p^t + \Delta t \sum_I \phi_I(\mathbf{X}_p) \mathbf{v}_I^{t+\Delta t}$
  end
end while

8.3 Contact in GPIC

As GPIC offers some advantages (smooth stress field, material interfaces and easy boundary condition handling) compared with other MPM variants, we present a frictional contact algorithm for GPIC. The contact treatment follows the predictor-corrector formulation presented in Sect. 8.1. For computer implementation, we present the full algorithm in Algorithm 15. The presentation was only for frictionless contact.

Algorithm 15 Solution procedure of an explicit contact GPIC (TL).

Solve momentum equations for body b
  Mapping from particles to nodes (M2G)
    Compute nodal momentum $(m\mathbf{v})_I^{t,(b)} = \sum_J \phi_I(\mathbf{x}_J^t)(m\mathbf{v})_J^t$
    Compute external force $\mathbf{f}_I^{\text{ext},t,(b)} = \sum_J \phi_I(\mathbf{x}_J^t)\mathbf{f}_J^{\text{ext},t}$
    Compute internal force $\mathbf{f}_I^{\text{int},t,(b)} = -\sum_J \phi_I(\mathbf{x}_J^t)\mathbf{f}_J^{\text{int},t}$
    Compute nodal force $\mathbf{f}_I^{t,(b)} = \mathbf{f}_I^{\text{ext},t,(b)} + \mathbf{f}_I^{\text{int},t,(b)}$
    Update the body momenta $(m\mathbf{v})_I^{t+\Delta t,(b)} = (m\mathbf{v})_I^{t,(b)} + \mathbf{f}_I^{t,(b)}\Delta t$
    Update the system momenta $(m\mathbf{v})_I^{t+\Delta t} = (m\mathbf{v})_I^{t+\Delta t} + (m\mathbf{v})_I^{t+\Delta t,b}$
    Update the system mass $m_I^t = m_I^t + m_I^{t,(b)}$
  end
end
Correct velocity for contact nodes
  for contact node I of body b **do**
    Compute normal to b, $\mathbf{n}_I^{(b)}$
    Retrieve center of mass velocity $\mathbf{v}_I^{\text{cm}} = (m\mathbf{v})_I^{t+\Delta t}/m_I^t$
    Check contact or release $\alpha = \left(\mathbf{v}_I^{t+\Delta t,(b)} - \mathbf{v}_I^{\text{cm}}\right)\cdot\mathbf{n}_I^{(b)}$
    If $\alpha \geq 0$: $\mathbf{v}_I^{t+\Delta t,(b)} = \mathbf{v}_I^{t+\Delta t,(b)} - \left[(\mathbf{v}_I^{t+\Delta t,(b)} - \mathbf{v}_I^{\text{cm}})\cdot\mathbf{n}_I^{(b)}\right]\mathbf{n}_I^{(b)}$
    If $\alpha < 0$: keep $\mathbf{v}_I^{t+\Delta t,(b)}$
  end for
end
Update particle velocity, position & displacement for body b (G2M)
  Get nodal velocities $\mathbf{v}_I^{t+\Delta t,(b)} = (m\mathbf{v})_I^{t+\Delta t,(b)}/m_I^t$
  Update mesh velocities $\mathbf{v}_J^{t+\Delta t,(b)} = \mathbf{v}_J^{t,(b)} + \sum_I \phi_I(\mathbf{x}_J^t)\left[\mathbf{v}_I^{t+\Delta t,(b)} - \mathbf{v}_I^{t,(b)}\right]$
  Update mesh positions $\mathbf{x}_J^{t+\Delta t,(b)} = \mathbf{x}_J^{t,(b)} + \Delta t\sum_I \phi_I(\mathbf{x}_J^t)\mathbf{v}_I^{t+\Delta t,(b)}$
  Update mesh incremental displacement $\mathbf{du}_J^{t+\Delta t,(b)} = \Delta t\sum_I \phi_I(\mathbf{x}_J^t)\mathbf{v}_I^{t+\Delta t,(b)}$
  Fix Dirichlet nodes K: $\mathbf{v}_K^{t+\Delta t,(b)} = \mathbf{0}$, $\mathbf{du}_K^{t+\Delta t} = \mathbf{0}$, $\mathbf{x}_K^{t+\Delta t} = \mathbf{X}_K$
end
Update stress and forces on the FE mesh for body b (UMF)
  Update Cauchy stress at element center $\boldsymbol{\sigma}(\boldsymbol{\xi}_0)$
  Convert Cauchy stress to 1st PK stress $\mathbf{P}(\boldsymbol{\xi}_0) = J\boldsymbol{\sigma}(\boldsymbol{\xi}_0)(\mathbf{F}(\boldsymbol{\xi}_0))^{-\text{T}}$
  Compute internal force $\mathbf{f}_J^{\text{int},t} = \mathbf{P}(\boldsymbol{\xi}_0)\nabla_0\phi_J^{\text{FE}}(\boldsymbol{\xi}_0)w(\boldsymbol{\xi}_0)$
  Compute external force $\mathbf{f}_J^{\text{ext},t}$
end

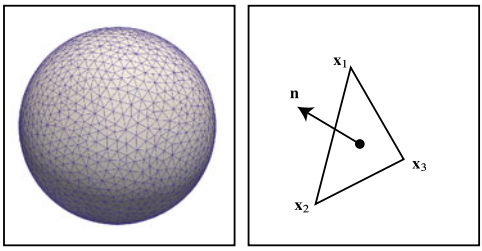


Fig. 8.11 Normal vectors in GPIC

With GPIC it is more straightforward to get contact nodes. From the finite element meshes, we can get surface nodes,[2] and the Eulerian grid nodes to which these surface nodes contribute are potential contact nodes. Actual contact nodes are then those of these potential contact nodes that have contribution from more than one bodies.

Normal vectors in GPIC. We compute the normals a bit differently than the MPM because of the FE mesh. For each element on the surface of a solid, we compute the normal to this element at its centroid, $\mathbf{n}_0$, by computing the two tangent vectors and take the cross product of them (Fig. 8.11):

$$\mathbf{n}_0 = \mathbf{s} \times \mathbf{t}, \quad \mathbf{s} = \sum_{J=1} \frac{\partial \phi_J^{\text{FE}}}{\partial \xi} \mathbf{x}_J, \quad \mathbf{t} = \sum_{J=1} \frac{\partial \phi_J^{\text{FE}}}{\partial \eta} \mathbf{x}_J \tag{8.34}$$

We, then, interpolate these normals at the centroids to the Eulerian grid. Alternatively, we can also compute the normals at the FE nodes. This is more involved as one needs to find the unique normals at the nodes (each node has more than 1 normal). One crucial issue here is the surface elements must be such oriented that the normals are poining outward, see Fig. 8.12.

8.4 Contact Simulations

To demonstrate the performance of the MPM for contact simulations, we present the following tests

- Test 1: collision of two compressible Neo-Hookean rings (Sect. 8.4.1)
- Test 2: high velocity impact of a steel disk to an alloy sample (Sect. 8.4.2)
- Test 3: contact of a rigid sphere with a half plane (Sect. 8.4.3)
- Test 4: cylinder rolling on an inclined plane (Sect. 8.4.4)
- Test 5: stress wave in a granular material (Sect. 8.4.5)

[2] Which are simply the nodes of the elements locating on the surface.

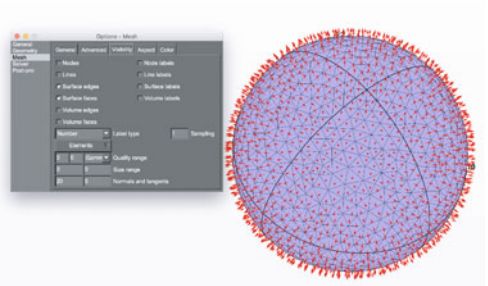


Fig. 8.12 The mesh must be created such that the normals are pointed outward. In `Gmsh`—a popular mesh generator—one can see the normals by going to Option/Mesh/Visibility/Normals and tangents. The normal direction can be changed by manually adding a minus sign to the surface definition in the geometry file

- Test 6: 3D version of Test 2 (Sect. 8.4.6)
- Test 7: scratch test (Sect. 8.4.7)

All these tests are popular problems in the MPM literature, except Test 7 which simulates the scratching of metals; scratch test is a popular experiment to determine the hardness of certain solid material. We will evaluate the performance of TLMPM, ULMPM and GPIC. And for Test 7, a comparison with SPH is given.

8.4.1 Test 1: Collision of Two Compressible Neo-Hookean Rings

This example problem involves a collision of two hollow elastic cylinders, under the assumption of plane strain (Fig. 8.13). This example was used in Gray et al. (2001) to show that SPH suffers from numerical fracture due to tensile instability. A similar problem involving one ring was solved by Sulsky et al. (1995) using the standard MPM i.e., ULMPM with linear weighting function. The setup used here and shown in Fig. 8.13 was given in Huang et al. (2011). The material is a compressible Neo-Hookean with bulk modulus $K = 121.7$ MPa, shear modulus $G = 26.1$ MPa and density $\rho = 1010 \times 10^{-12}$ kg/mm^3. The magnitude of the rings initial velocity is $v_0 = 30$ m/s.

The aim of this example are multi-fold: (1) to show that the ULMPM stress field is noisy and the TLMPM can solve this stress noisy problem, and (2) to show that better energy conservation is obtained using the TLMPM than the ULMPM and (3) GPIC is the best in terms of energy conservation.

This problem is solved with both the ULMPM and the TLMPM using the hat weighing function. For both simulations the cell size is 1.25 mm (that is the grid consists of 160×120 cells. The rings are formed by inserting four particles per cell at the location of the Gauss quadrature points. Though, only the particles lying within

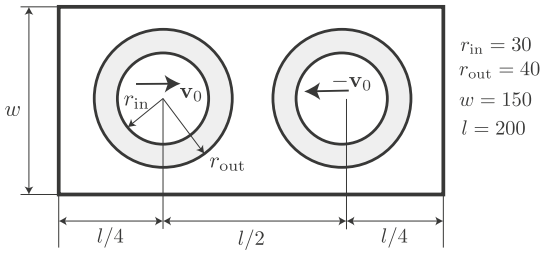


Fig. 8.13 Impact of two elastic bodies: problem description. Dimensions are in millimeters (de Vaucorbeil et al. 2020)

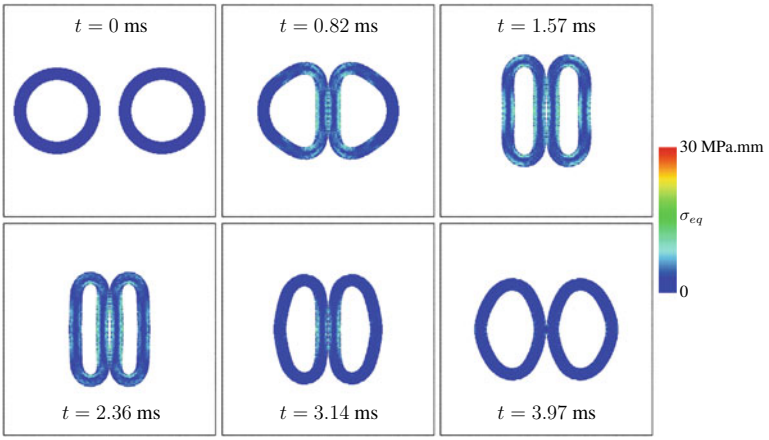


Fig. 8.14 Snapshots of the impact of two compressible Neo-Hookean rings using the ULMPM (hat functions) (de Vaucorbeil et al. 2020)

the boundaries of the rings are kept. The simulations are run until the time lapsed reaches 5.5 ms, or when using the ULMPM, until the particles leave the simulation domain.

A time-lapse of the ULMPM and TLMPM simulations are given in Figs. 8.14 and 8.15. There we observe that in the first half of the simulation (i.e., $t < 2$ ms), the results are very similar. However, in the second half, when using the ULMPM, the rings have difficulties to separate from each other (see Fig. 8.14). This is due to contacts not being unilateral in the ULMPM. This can be fixed using the multi-material contact algorithm of Bardenhagen et al. (2000). However, the here-proposed contact algorithm for TLMPM is unilateral: when the penetration between two particles is null or negative, no force is applied.

We present the geometries of the two rings as well as the different stress fields obtained at $t = 1.59$ ms, i.e., during impact, by both ULMPM and TLMPM in Fig. 8.16. The observation is two-fold: (a) a significant gap exists between the two rings when using the ULMPM and no gap is visible in the TLMPM result (Fig. 8.16b), (b) the TLMPM stress field is much smoother than that obtained with the ULMPM.

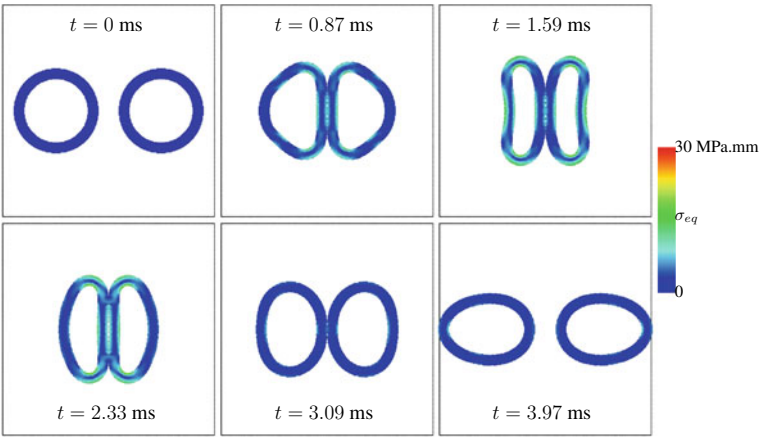
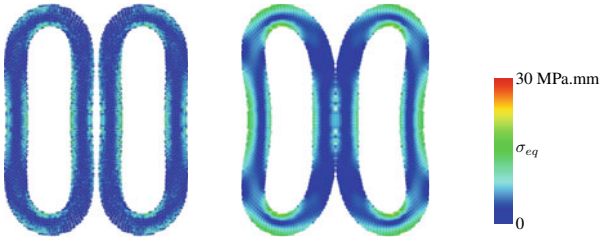


Fig. 8.15 Snapshots of the impact of two compressible Neo-Hookean rings using the TLMPM (hat functions) (de Vaucorbeil et al. 2020)

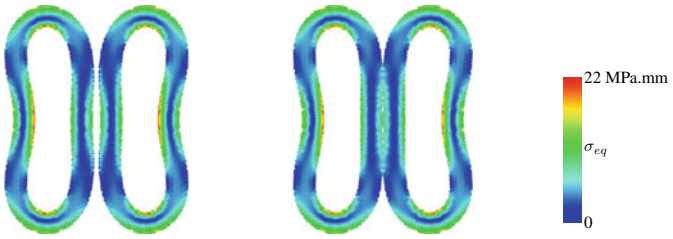


(a) ULMPM (b) TLMPM

Fig. 8.16 Comparison of the stress field obtained at $t = 1.59$ ms of the impact of two compressible Neo-Hookean rings using ULMPM and TLMPM with hat weighting functions (de Vaucorbeil and Nguyen 2021b)

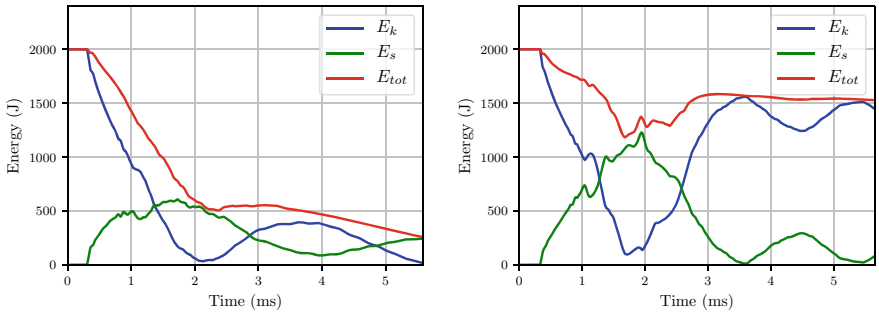
Of course, it is well known that the ULMPM with hat functions present cell-crossing instabilities which are the root cause of the noisy stress field seen in Fig. 8.16a (Bardenhagen and Kober 2004). However, Steffen et al. (2008a) showed that using cubic B-splines instead can reduce these instabilities. And indeed, when doing so, the stress field obtained in this example is smoother as can be seen in Fig. 8.17. But little improvement can be seen for the results when using TLMPM with cubic B-splines. The use of cubic B-splines do not improve the contact gap in ULMPM, and its use does not create a gap when using the TLMPM. This shows that in the TLMPM, unlike in the MPM, the separation of two solids in contact is independent of the type of the weighting function used.

Remark 45 Our discussion on the contact gap present in the MPM was applied only for the built-in no-slip no-penetration contact. It is certain that this gap can be removed when extra contact algorithm is added to the MPM, see e.g. Huang et al. (2011).



(a) ULMPM (b) TLMPM

Fig. 8.17 Comparison of the stress field obtained at $t = 1.59$ ms of the impact of two compressible Neo-Hookean rings using TLMPM and ULMPM with cubic B-spline weighting functions (de Vaucorbeil and Nguyen 2021b)

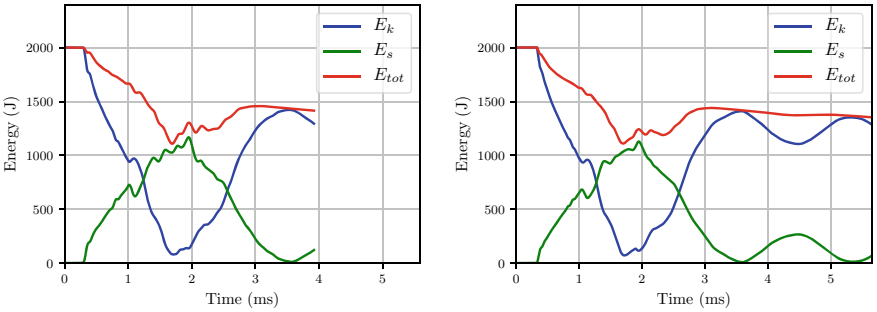


(a) ULMPM (b) TLMPM

Fig. 8.18 Impact of two compressible Neo-Hookean rings: comparison of the evolution of the kinetic, strain and total energies, respectively E_k, E_s and E_{tot} using **a** ULMPM and **b** TLMPM. Both simulations use the linear (hat) weighing functions (de Vaucorbeil and Nguyen 2021b)

As the contact between these two rings is purely elastic, perfect energy conservation is sought. Although the energy before impact is the same when using both the ULMPM and TLMPM, a substantial loss of energy is observed when using the ULMPM with the hat weighting functions (Fig. 8.18). In the same conditions, the TLMPM exhibits much better energy conservation. Energy conservation can be improved by using cubic B-splines with the ULMPM (see Fig. 8.19). However, this increases the energy loss with the TLMPM. This is due to the interpolation of the contact forces onto the background grid nodes. Indeed, when using B-splines, this interpolation distributes these forces onto nodes that are further from the contact surface than when using the hat functions. This shows that amongst the methods presented here, for this example, TLMPM with hat weighting functions is the most efficient.

Finally we solve this problem using GPIC (Fig. 8.20). In GPIC, the strain and kinetic energy E_s and E_k are computed as



(a) ULMPM (b) TLMPM

Fig. 8.19 Impact of two compressible Neo-Hookean rings: comparison of the evolution of the kinetic, strain and total energies, respectively E_k, E_s and E_{tot} using **a** ULMPM and **b** TLMPM. Both simulations use the cubic B-splines weighting functions (de Vaucorbeil and Nguyen 2021b)

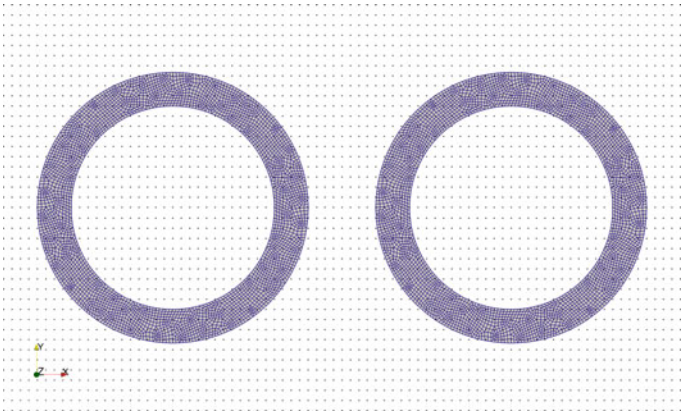


Fig. 8.20 Collision of two compressible rubber rings: GPIC set-up with four-node quadrilateral elements for the rings. The mesh of the rings is created using Gmsh (Geuzaine and Remacle 2009a). The dots are the nodes of the Eulerian grid (Nguyen et al. 2021)

$$E_s = \sum_{g=1} u_g V_g, \quad E_k = \frac{1}{2} \sum_{J=1} \mathbf{v}_J \cdot \mathbf{v}_J m_J \tag{8.35}$$

where u_g denotes the strain energy density of quadrature point g, $u_g = 1/2\boldsymbol{\sigma}_g :: \boldsymbol{\varepsilon}_g$; $\boldsymbol{\varepsilon}_g$ is the strain tensor; $\mathbf{v}_J$ and m_J denote the velocities and mass at node J of the solid mesh.

Remarkably, GPIC performs better in terms of energy conservation than TLMPM/ULMPM (Fig. 8.21). The performance of GPIC is close to Abaqus. In GPIC, thanks to the mesh only a coarse discretization of 6 368 four-node quadrilateral elements (or particles) were sufficient to accurately represent the curved boundaries. Thus, GPIC is a very efficient MPM version.

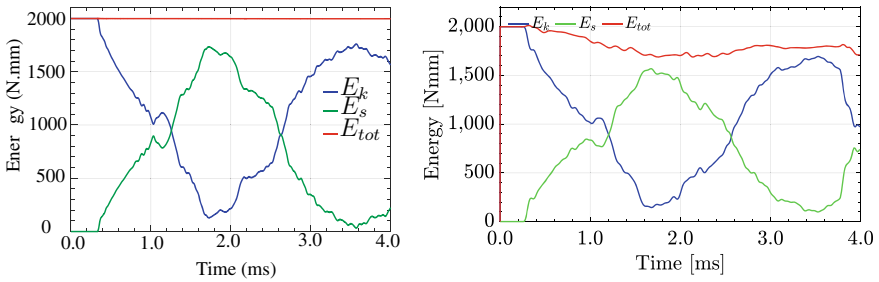


Fig. 8.21 Impact of two compressible Neo-Hookean rings: energy profiles of FEM (left) and GPIC (right). GPIC: Eulerian grid of 80 × 40 cells and 6 368 four-node quadrilateral elements. Abaqus: 20 000 eight-node hexahedral elements with one point quadrature (we did a 3D analysis in Abaqus). Note that the total energy in Abaqus consists of not only E_s and E_k but also the hourglass energy (Nguyen et al. 2021)

8.4.2 *Test 2: High Velocity Impact of a Steel Disk Onto an Aluminum Target*

The high velocity impact problem of a steel disk onto an aluminum target is interesting as this problem involves non-elastic contacts. In this problem, presented in Fig. 8.22, an AISI 52-100 chromium steel disk impacts an elastic-perfectly plastic target made out of 6061-T6 aluminum under plane strain conditions. This problem, introduced in Sulsky et al. (1995); Coetzee (2003), was inspired by the experiments carried out by Trucano and Grady (1985). The steel disk is assumed to be linear elastic and the aluminum target to be elastic-perfect plastic and obeying a von Mises yield criterion. The boundary of the computational domain is fixed.

This test shows the capabilities of the TLMPM in the simulation of high velocity impacts and large deformations. The simulation is run for 40 μs, it uses hat weighting functions and the results are compared with the same setup run with the ULMPM.

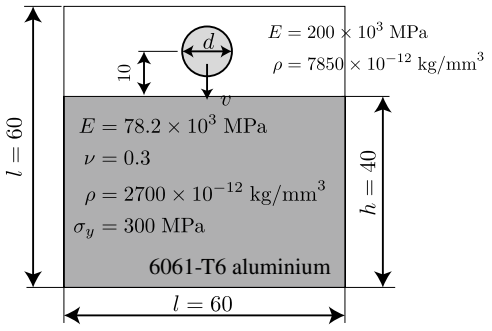


Fig. 8.22 Impact of a steel disk into an aluminum target. The disk has an initial velocity of 1160 m/s and the disk's diameter is 9.53 mm. Units are N, mm, and s

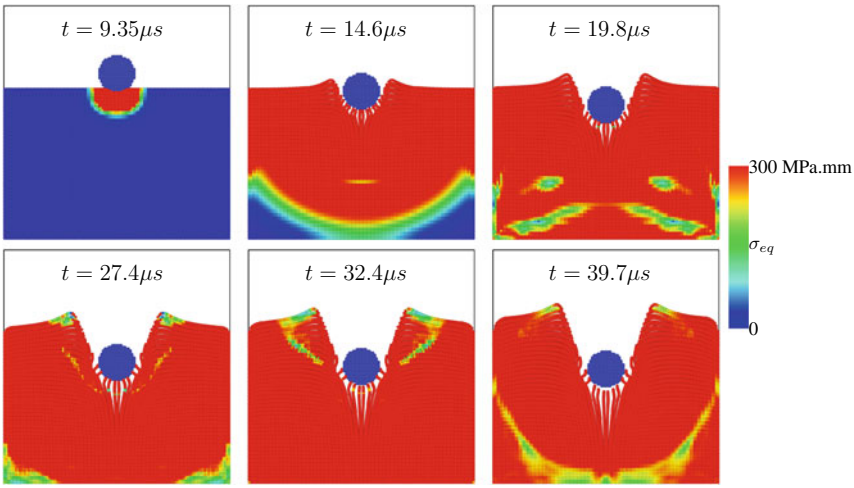


Fig. 8.23 Impact of a steel disk into an aluminum target: snapshots of the simulation performed with TLMPM using linear shape functions and two particles per cell. The cell size is 1 mm for the target and 0.953 mm for the disk (de Vaucorbeil and Nguyen 2021b)

The time-lapse of the simulations are shown in Figs. 8.23 and 8.24, when performed with TLMPM and ULMPM, respectively. In both cases, the simulation is stable. However, the stress field obtained by the TLMPM is much smoother than that obtained with the ULMPM. Moreover, the deformed surface of the aluminium target also looks smoother when TLMPM is used.

The disk penetration as a function of time is given in Fig. 8.25 and shows that results obtained with both methods are in good agreement with each other. The lack of experimental data for the example does not allow to check which solution is the more realistic, though.

8.4.3 Test 3: Contact of a Rigid Sphere with a Half Plane

As the expression of the force needed to cause a penetration δ_s of a rigid sphere into an elastic half plane is known analytically, this problem is used to assess how well the contact forces are predicted using the TLMPM. The problem setup is as shown in Fig. 8.26. The diameter of the substrate is much larger than that of the contact surface and its height much larger than the penetration of the sphere, thus boundary effects are negligible.

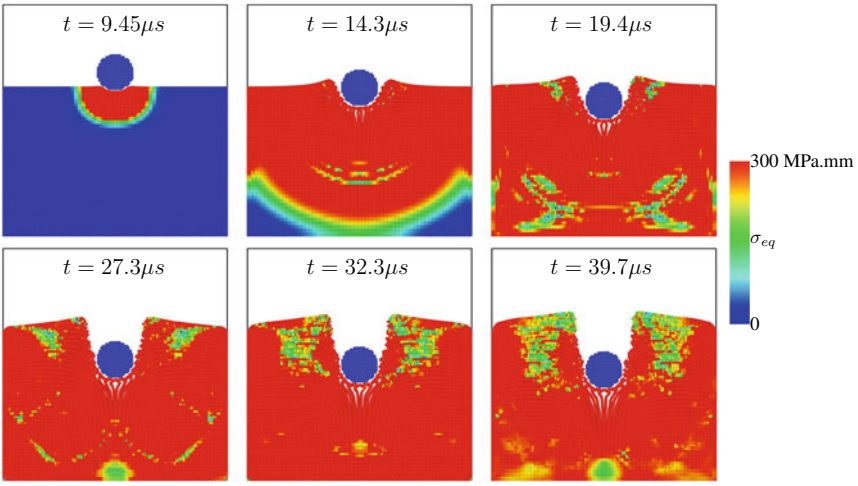


Fig. 8.24 Impact of a steel disk into an aluminum target: snapshots of the simulation performed with ULMPM using linear shape functions and two particles per cell. The cell size is 1 mm in the whole domain (de Vaucorbeil and Nguyen 2021b)

Fig. 8.25 Impact of a steel disk into an aluminum target: comparison of the disk penetration as a function of time as predicted by TLMPM and ULMPM (de Vaucorbeil and Nguyen 2021b)

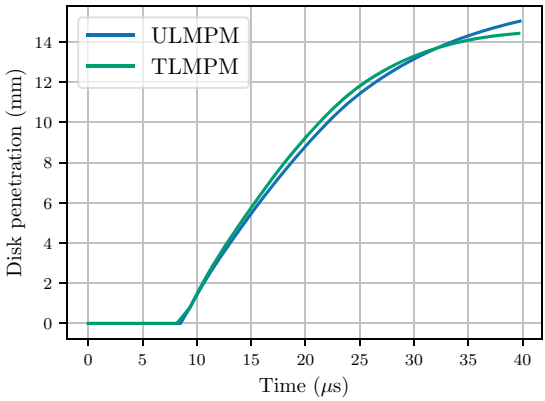
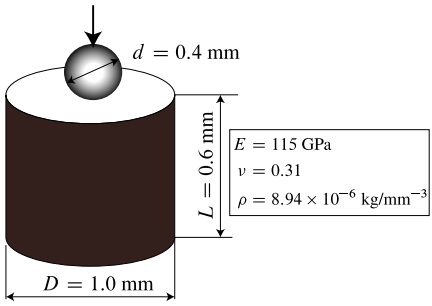


Fig. 8.26 Contact of a rigid sphere with a half plane, problem setup (de Vaucorbeil and Nguyen 2021b)



The force applied on the sphere to obtain a penetration δ_s into the half plane is given by the Hertz theory as

$$F = \frac{4}{3}\frac{E}{1-\nu^2}\sqrt{\frac{d}{2}}\delta_s^{\frac{3}{2}} \tag{8.36}$$

where E is the half plane's Young's modulus, ν its Poisson's ratio, and d the sphere's diameter.

As the problem is axisymmetric around the vertical axis passing through the center of the sphere, an axisymmetrical model is used here to reduce simulation time as we will do a mesh convergence analysis. The sphere being rigid, it is modeled as a spherical shell which thickness is a few cell sizes. The TLMPM with hat weighing functions and one particle per cell is used, and its convergence towards the analytical solution given in Eq. (8.36) is studied as a function of the background grid cell size. The error made in the prediction of the applied force is calculated as

$$e = \max_{\delta_s^t}(|F_{\text{sim}}(\delta_s^t) - F_{\text{th}}(\delta_s^t)|)/F_{\text{th}}(\delta_{s,m}^t) \tag{8.37}$$

where $\delta_{s,m}^t$ is the penetration at which $|F_{\text{sim}}(\delta_s^t) - F_{\text{th}}(\delta_s^t)|$ is maximized with $F_{\text{sim}}(\delta_s^t)$ and $F_{\text{th}}(\delta_s^t)$ are the predicted and theoretical applied force for a given depth δ_s^t. The same simulations are performed with the ULMPM using both hat and cubic B-splines.

The force-penetration profiles obtained using the TLMPM are smooth as one can see in Fig. 8.27. Although the applied force is always over-predicted, it converges towards the analytical solution as the background grid cell size decreases. The force-penetration profiles obtained using the ULMPM with hat weighting functions, on the other hand, presents high levels of noise (see Fig. 8.28), which are believed to come from cell-crossing instabilities. This generates a much higher error than the TLMPM, though convergences can still be seen.

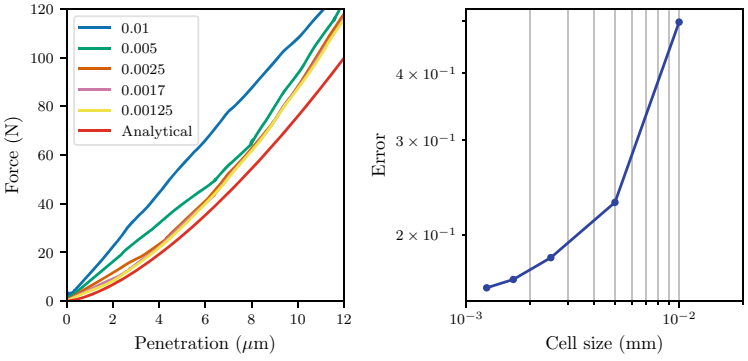


Fig. 8.27 Effect of the cell size on the applied force penetration profile of the sphere into the half plane when using TLMPM with linear shape functions and one particle per cell (de Vaucorbeil and Nguyen 2021b)

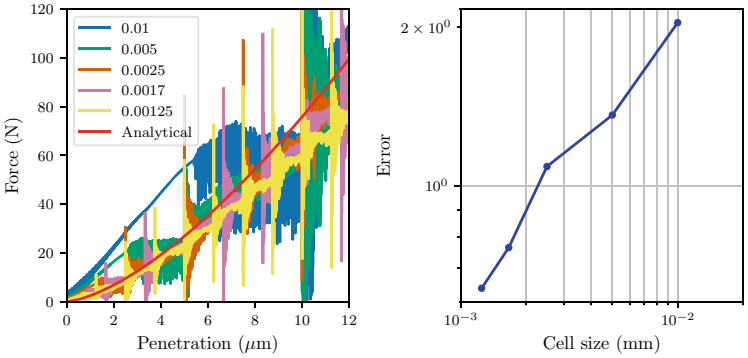


Fig. 8.28 Effect of the cell size on the force penetration profile of the sphere into the half plane when using ULMPM with linear shape functions and one particle per cell at the start of the simulation (de Vaucorbeil and Nguyen 2021b)

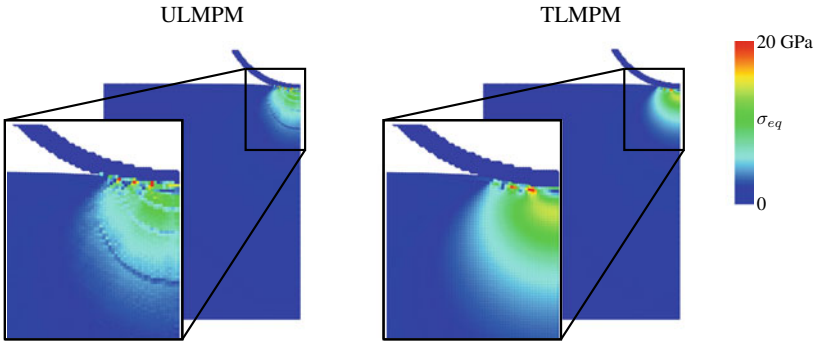


Fig. 8.29 Comparison of the equivalent stress field due to the contact between a rigid indenter and an elastic half plane using ULMPM and TLMPM. These results were obtained using linear shape functions and one particle per cell with a cell size of 0.0025 mm when the indentation force equals 120 N. One can clearly see that with ULMPM, the stress is not smooth (de Vaucorbeil and Nguyen 2021b)

Just like what was seen in the test cases 1 and 2 (Sects. 8.4.1 and 8.4.2), the stress field obtained using the TLMPM is much smoother than that obtained with the ULMPM (when using hat functions) as one can see in Fig. 8.29.

The noise observed in the force-penetration profiles obtained with ULMPM could be significantly reduced by using cubic B-splines (Fig. 8.30). In this case, the applied force is under-predicted, which is the opposite behavior from that seen when using TLMPM. Even though the error monotonically decreases with the cell size, for high cell size (larger than 2×10^{-3} mm), the error is larger than that given by the TLMPM with hat weighting functions (see Fig. 8.31). This demonstrates that the contact algorithm for the TLMPM with hat weighing functions performs well: low error is made in the prediction of the applied forces. There is no need here to use cubic B-splines as they increase the problem's complexity.

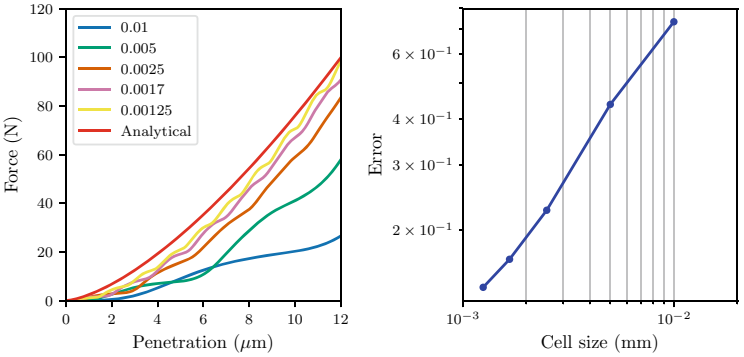
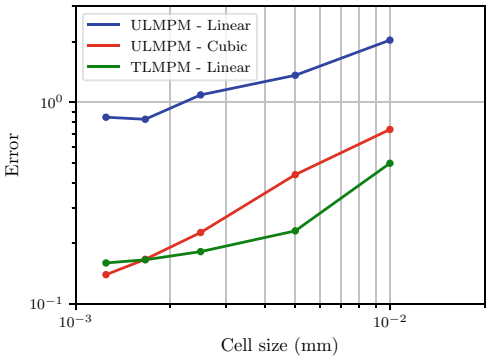


Fig. 8.30 Effect of the cell size on the force penetration profile of the sphere into the half plane when using ULMPM with cubic B-splines shape functions and one particle per cell at the start of the simulation (de Vaucorbeil and Nguyen 2021b)

Fig. 8.31 Comparison of the error made by ULMPM—using both linear (blue curve) and cubic B-splines (red curve)—and TLMPM (green curve) onto the prediction of the force penetration of a sphere into a half plane when using one particle per cell (de Vaucorbeil and Nguyen 2021b)



8.4.4 Test 4: Cylinder Rolling on an Inclined Plane

To test the frictional contact algorithm, the problem of a cylinder rolling on an inclined plane is considered (Fig. 8.32a), which is probably the simplest test for frictional contact. This problem has been studied by many authors in the MPM literature e.g. Bardenhagen et al. (2000, 2001); Huang et al. (2011). The plane is inclined at an angle θ from the horizontal, while gravity points vertically downward. Figure 8.32b shows the computational model in which the inclined plane is aligned with the boundary of the computational mesh and gravity makes an angle θ to the vertical.

A rigid disk on an inclined surface will roll, and stick or slip at the contact point depending on the angle of inclination θ and the friction coefficient μ. For an initially stationary, rigid disk, the x-component of the center-of-mass position $x_{\text{cm}}(t)$ is given by Bardenhagen et al. (2000)

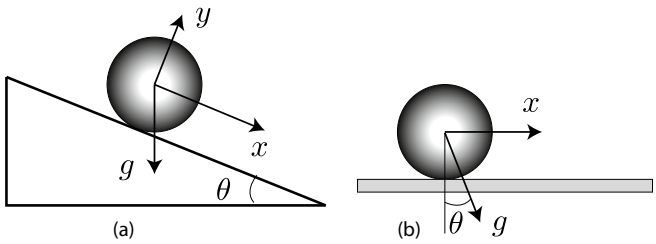


Fig. 8.32 Cylinder rolling on an inclined plane (de Vaucorbeil and Nguyen 2021b): **a** problem description and **b** computational configuration. The radius of cylinder is $R = 0.5$ m, and the computational domain is a rectangle of length 1.625 m and a width of 1.250 m. The magnitude of gravity g is taken to be 10 m/s^2. The cylinder is considered elastic with a bulk modulus of 10 MPa, a shear modulus of 2.5 MPa and a density of 3 g/cm^3. The plane is modeled as a rigid material. Actually there is no need to include the plane in the MPM simulation as the contact nodes are known prior to simulations and this was also done in Bardenhagen et al. (2000); Coetzee (2003)

$$x_{\text{cm}}(t) = \begin{cases} x_0 + 0.5gt^2(\sin\theta - \mu\cos\theta) & \tan\theta > 3.5\mu \quad \text{(slip)} \\ x_0 + \dfrac{1}{3}gt^2\sin\theta & \tan\theta \leq 3.5\mu \quad \text{(stick)} \end{cases} \tag{8.38}$$

where x_0 denotes x-component of the initial center-of-mass position.

Solving with the TLMPM. The inclined surface is modeled as an impenetrable half domain (not discretized). Following what has been done in Sect. 8.2.1, the penetration is:

$$\delta^t = R_p - d_p \tag{8.39}$$

where d_p is the distance between the particle p and the rigid surface. Similarly to Eq. (8.29), the force necessary to enforce $\delta^t = 0$ is:

$$\mathbf{F}_p = \frac{m_p}{\Delta t^2}\left(R_p - d_p^t\right)\mathbf{n}^t = \frac{m_p}{\Delta t^2}\delta^t\mathbf{n}^t \tag{8.40}$$

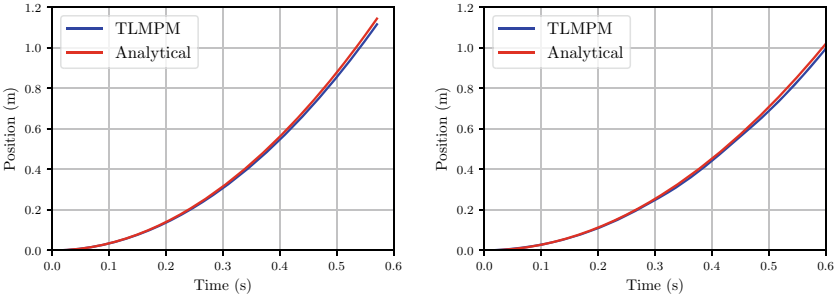
where $\mathbf{n}^t$ is the normal to the surface at time step t. With Coulomb friction, it becomes:

$$\mathbf{F}_p = \frac{m_p}{\Delta t^2}\delta^t\left[\mathbf{n}^t - \mu\left(\mathbf{v}_p^t - \frac{\mathbf{v}_p^t\cdot\mathbf{n}^t}{\|\mathbf{v}_p^t\|}\right)\right] \tag{8.41}$$

where μ is the friction coefficient. Note that this formulation is equivalent to the contact between a particle with an infinite radius and a normal particle p.

We adopt a plane strain 2D model with a uniform grid of equal spacing $h = 25$ mm in all directions and one material points per cell (resulting in 1 264 deformable particles with 25 cells across the cylinder). The gravity is modeled as a body force which is given by

$$\mathbf{b} = \begin{bmatrix} g\sin\theta & -g\cos\theta \end{bmatrix}^{\text{T}} \tag{8.42}$$



(a) $\mu = 0.3$ (b) $\mu = 0.6$

Fig. 8.33 Cylinder rolling on an inclined plane (de Vaucorbeil and Nguyen 2021b): comparison of the position of the center of mass of the cylinder as predicted by the TLMPM and the theoretical solution from Eq. (8.38)

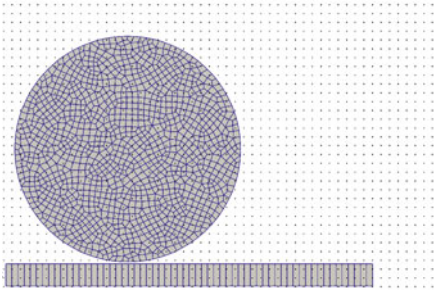


Fig. 8.34 Cylinder rolling on an inclined plane: GPIC set-up: the cylinder is meshed with 1 297 four-node quadrilateral elements, the rigid plane meshed with 60 Q4 elements and the Eulerian grid meshed with 50×30 cell. There are about 25 cells across the cylinder

And during the simulation, we record the center of mass position of the cylinder which is defined by

$$\mathbf{x}_{\text{cm}}^{\text{num}} = \frac{\sum_p m_p \mathbf{x}_p}{\sum_p m_p} \tag{8.43}$$

where the sum is over the cylinder particles.

The incline angle is chosen to be $\theta = \pi/3$ and two friction coefficients are tested: $\mu = 0.3$ and $\mu = 0.6$. The first corresponds to rolling-slip case and the second to the rolling-stick case. In both cases, excellent agreement was obtained (Fig. 8.33).

Solving with GPIC. Even though for this problem there is no need to explicitly model the rigid plane, we still do that. This is because for other problems the rigid bodies might have complex geometries. The GPIC set-up is shown in Fig. 8.34. The normal vectors at contact nodes are determined using the normal vectors to the rigid plane. In the calculation, only the nodes on the boundary of the rigid plane are needed, that is why only one layer of elements across the plane thickness is required. Figure 8.35 confirms the implementation of frictional contact in GPIC.

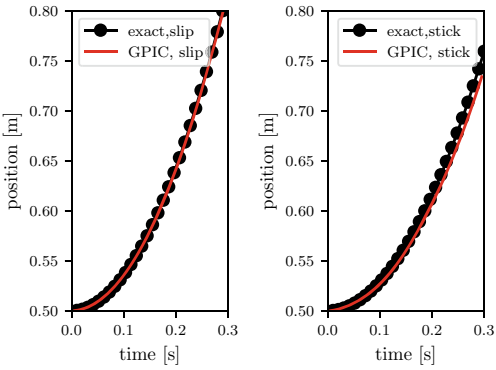


Fig. 8.35 Cylinder rolling on an inclined plane: comparison of the position of the center of mass of the cylinder as predicted by GPIC (Eq. (8.43) applied to the FE nodes) and the theoretical solution from Eq. (8.38)

8.4.5 Test 5: Stress Wave in a Granular Material

All previous tests featured the contact between only two solids. However, the MPM contact algorithm is able to handle multiple contacts efficiently. To demonstrate this, we reproduce in this section two interesting simulations of stress wave propagation in a granular media presented in Bardenhagen et al. (2001). The first problem consists of four identical collinear disks impacted by a striker from the right traveling at a speed of 5.6 m/s which is summarized graphically in Fig. 8.36. Both the disks and the impactor are considered as made of linear elastic material. As there is no sliding in this set up no frictional contact was used to model the interaction between the disks.

Solving using TLMPM. Each background grid has a cell size of $h = 1.25$ mm. One particle per cell is used for both the disks and the impactor. The total number of particles is thus 20 896. The simplest of weighting functions is used: hat functions. In the experiments, the stress distribution in the disks is evaluated using photoelasticity. This process generates dark fringes at contours of constant maximum difference in the principal stresses.

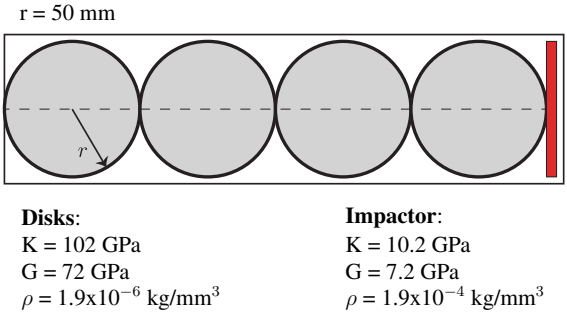
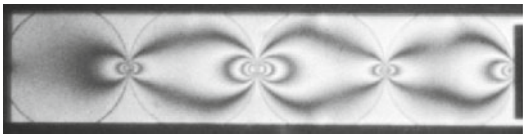
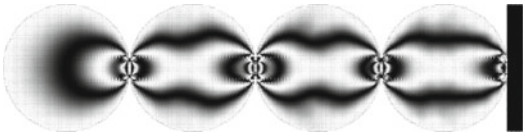


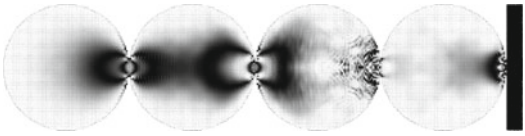
Fig. 8.36 Stress wave in granular media: problem configuration. The impactor is given a constant velocity of 0.0006 cm/s to the left (de Vaucorbeil and Nguyen 2021b)



(a) Experiments. Reprinted under the Creative Commons Attribution 4.0 International License from Bardenhagen et al. [2001] Copyright ©2001 Tech Science Press



(b) TLMPM



(c) ULMPM

Fig. 8.37 Stress wave in a granular material. Comparison between the stress fringes obtained by experiments and using total Lagrangian and Updated Lagrangian MPM at $t = 0.0659$ ms after impact (de Vaucorbeil and Nguyen 2021b)

In the simulations, fringes are generated by the following equation

$$1 - \sin^2\left(k_f(\sigma_1 - \sigma_3)\right) \tag{8.44}$$

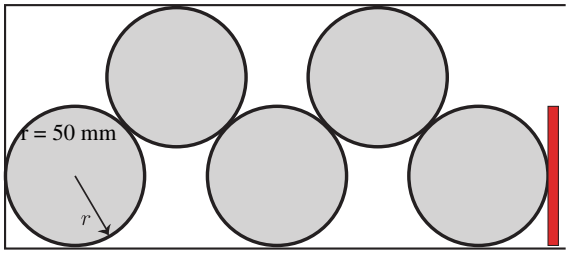
with k_f is an unknown optical parameter that controls the finge density taken here as $k_f = \pi/0.07$ GPa^{-1}, and the difference of in-plane principal stress is given by

$$\sigma_1 - \sigma_3 = 2R = \sqrt{4\tau_{xy}^2 + (\sigma_x - \sigma_y)^2} \tag{8.45}$$

where R denotes the radius of the Mohr's circle.

The fringe pattern obtained with the TLMPM and ULMPM is shown alongside the experimental results in Fig. 8.37. The results from the TLMPM (Fig. 8.37b) are in good agreement with the experiments (Fig. 8.37a) and those from the ULMPM are not as good. This is once again due to the cell-crossing instabilities. Therefore, the TLMPM with the present contact algorithm allows for a smoother stress field than the ULMPM when using hat weighing functions. Of course one could use cubic B-splines and get a smoother stress field with the ULMPM, but even with the simplest of shape functions, the TLMPM gives good stress predictions. Thus, the degree of computational complexity is decreased.

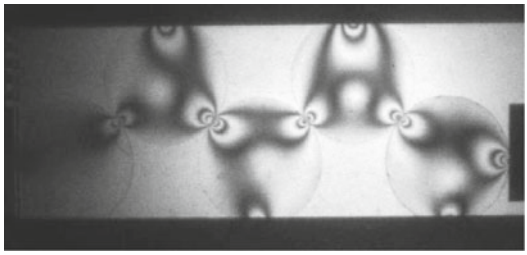
As the contact algorithm for the TLMPM can model frictional contacts, it allows the simulation of stress wave propagation in a granular material which involves frictional contacts between the different grains. To show this, we consider a problem in which five disks contacting each other at 45° are impacted by a striker from the right, traveling at the same speed as before: 5.6 m/s. The disks are prevented from moving out of their position by a rigid box (Fig. 8.38). The disk, the impactor, as



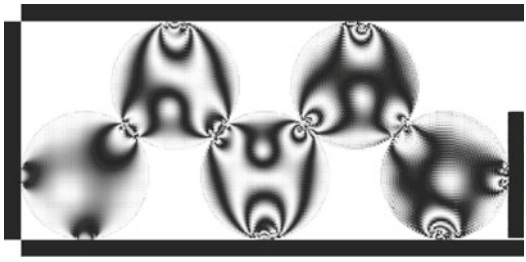
Disks:
K = 102 GPa
G = 72 GPa
ρ = 1.9x10^{-6} kg/mm^3

Impactor:
K = 10.2 GPa
G = 7.2 GPa
ρ = 1.9x10^{-4} kg/mm^3

Fig. 8.38 Stress wave in granular media: problem configuration. The impactor is given a constant velocity of 0.0006 cm/s to the left (de Vaucorbeil and Nguyen 2021b)



(a) Experiments Reprinted under the Creative Commons Attribution 4.0 International License from Bardenhagen et al. [2001] Copyright ©2001 Tech Science Press



(b) TLMPM

Fig. 8.39 Stress wave in a granular material. Comparison between the stress fringes obtained by experiments and using total Lagrangian MPM at $t = 0.12$ ms after impact (de Vaucorbeil and Nguyen 2021b)

well as their material properties are the same as in the four collinear disk problem. The discretization of the problem is identical as before, i.e., the background grid cell size is $h = 1.25$ mm and one particle per cell, resulting in a total of 25 910 particles. The sides of the box are simulated as unpenetrable surfaces, just like what was done for the inclined surface in Sect. 8.4.4. As the ULMPM algorithm does not allow frictional contact (without adding an extra contact algorithm), only the TLMPM is used here. And just as for the previous problem, it features hat weighing functions.

Just like in the case of the 4 disks, as one can see in Fig. 8.39 good agreement is obtained between the simulation and the experiments. One might observe that the stress field in the first impacted disk (the one on the far right side of the picture) is more

noisy than in the others. This is attributed to the use of a high PIC/FLIP mixing factor (i.e., $\alpha = 0.99$). Indeed, FLIP is known for being prone to instabilities. However, PIC would be far too dissipative, this is why such a value for α was chosen and good results are nonetheless obtained.

8.4.6 *Test 6: Penetration of a Steel Sphere Into an Aluminium Cylinder*

This test consists of an elastic sphere impacting an elastic-perfectly plastic target (Fig. 8.40). Sulsky et al. (1995) considered this problem in a 2D setting. Herein, we extend it to 3D, consider a higher impact velocity and study the robustness of GPIC for penetration problems.

This test is solved with MPM (hat functions and the modified update stress last of Sulsky and Schreyer (1996)) and GPIC. The two models are given in Fig. 8.41.

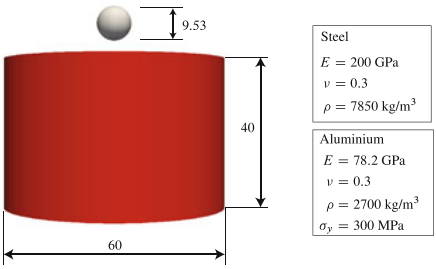


Fig. 8.40 Impact of a steel disk into an aluminum target. The disk (whose center is 10 mm from the cylinder top surface) has an initial velocity of 2000 m/s. Length in mm (Nguyen et al. 2021)

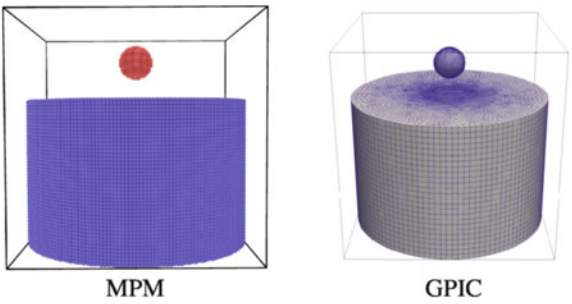


Fig. 8.41 Impact of a steel disk into an aluminum target: model set-ups. MPM: computational domain is $60 \times 60 \times 60$ discretized by a Eulerian grid of $60 \times 60 \times 60$ cells with 182 808 particles for both the cylinder and sphere. GPIC (TL): computational domain is $60 \times 60 \times 60$ discretized by a Eulerian grid of $60 \times 60 \times 60$ cells with 113 825 eight-node hexahedral elements for the cylinder and 21 848 four-node tetrahedral elements for the sphere (Nguyen et al. 2021)

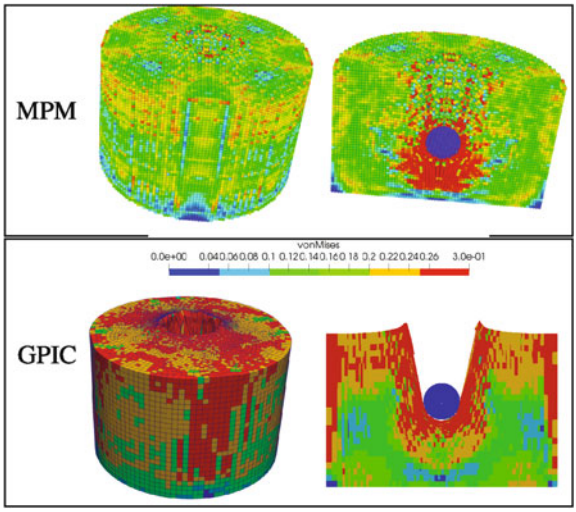


Fig. 8.42 Impact of a steel disk into an aluminum target: final configurations obtained with MPM and GPIC (Nguyen et al. 2021)

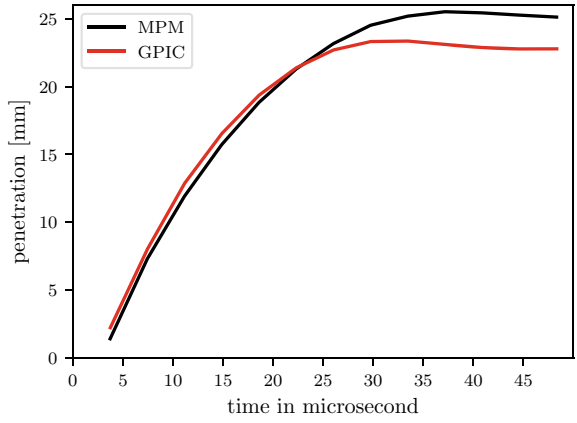


Fig. 8.43 Impact of a steel disk into an aluminum target: penetration curves (Nguyen et al. 2021)

All the faces of the Eulerian grid are fixed. As the geometry is simple, setting up the MPM model is easier and quicker than GPIC (which requires to use `gmsh`).

The final configurations shown in Fig. 8.42 indicate that the MPM and GPIC are robust for high velocity penetration simulations. They yield qualitatively similar results. However, the penetration curves are different: MPM gives a final penetration of 25 mm and GPIC gives a penetration of 23 mm (Fig. 8.43). This difference might come from different discretizations as Sulsky et al. (1995) showed that there is no convergence for this problem due to the ill-posedness of the problem under the assumption of perfect plasticity.

8.4.7 Test 7: Scratch Test

Scratch tests are carried out to determine the abrasive wear resistance of a material. The goal of these simulations is to check how well the MPM can simulate the single scratch test. To assess the performance of our codes, the simulation results are checked against the experimental data published by Leroch et al. (2016). For detail, we refer to de Vaucorbeil et al. (2022a).

Experiments. Experimentally, the scratch tests were done using high purity copper samples (99.999% pure). These samples are rectangular with the following dimensions: length 70 mm, width 30 mm and height 10 mm. The scratch test rig used is formed by a loading unit controlling the displacement/load of the indenter in the vertical direction (assumed to be z here) and a sample holder that move the horizontal direction (assumed to be x here). The scratch tests were performed in two steps:

1. indentation: the sample is fixed while the indenter is lowered until the desired load is reached,
2. scratching: the load of the indenter is kept constant while the sample holder moves horizontally.

Thus these scratches were performed under load-controlled conditions. These scratches were produced at velocities of up to 10 mm/s. The exact velocities have, however, not been reported. The indenter used is a diamond of standard Rockwell-C geometry (200 μm tip radius super-imposed on a 120° cone). This entails a spherical contact geometry until $\approx$ 30 μm scratch depth, while the cone geometry must be taken into account for deeper scratches. In the SPH simulations published in the same paper, the indenter was considered to be only a sphere of radius 200 μm even though depths higher than 30 μm (60 μm) were reached.

In these SPH simulations as well as in the experiments, the temperature was assumed to remain at room temperature, i.e., heating along the scratch was neglected. Leroch et al. argued that due to the combination of a small scratch velocity and the high thermal conductivity of copper any increase of temperature due to plastic deformation can be neglected.

Four different loading conditions were tested: 10 N, 20 N, 30 N and 50 N. The length of all scratches were around 11 mm. The width and depth of the groove as well as the height of the ridge (Fig. 8.44) generated were reported (see Table 8.1).

Materials and simulation set up. The material parameters used are given in Table 8.2. Simulating a whole sample would be too computationally demanding, therefore a smaller sample of dimensions $4 \times 1 \times 0.5$ mm^3 is used instead (Fig. 8.45). These dimensions where chosen such that (a) the simulation domain is as small as possible to minimise the computational cost, (b) the simulation box is big enough for the effects of the boundary conditions to be negligible and (c) that the scratching distance is high enough for a steady state to be reached. Moreover, noting that the xz plane including the centre of the indenter is a plane of symmetry,

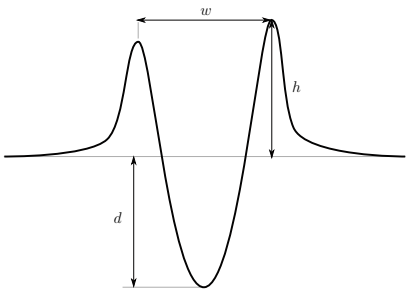


Fig. 8.44 Schematic of a typical scratch cross-section (in the xy plane) displaying how the scratch width w, depth d and the ridge height h are measured

Table 8.1 Experimental scratch topography results (Leroch et al. 2016)

Test load (N)	Scratch width w (μm)	Scratch depth d (μm)	Ridge height h (μm)
10	232.3	16.4	20.3
20	315.5	27.6	26.5
30	365.7	37.5	35.0
50	440.9	50.2	40.8

Table 8.2 Material parameters for the OFHC Copper material (Leroch et al. 2016)

Material parameters		Flow stress parameters		Parameters for EOS	
Density	8960 kg/m^3	A	90 MPa	c_0	3933 m/s
Young's modulus	120 GPa	B	292 MPa	S_α	1.5
Poisson's ratio	0.36	C	0.0	Γ_0	1.99
Reference temperature	294 K	n	0.31		
Melting temperature	1356 K	m	1.09		
		$\dot{\varepsilon}_0$	1.0 1/s		

only half of the specimen is considered in the simulations (Fig. 8.45). For boundary conditions, all sides except the plane of symmetry and the top face ($+z$) are fixed, i.e., the velocity of the background grid nodes coincident with these boundaries are set to zero. The symmetry is imposed by setting the velocity of the background grid nodes coincident with the plane of symmetry to zero along the y axis. The simulation domain is uniformly discretized, the background grid cell size being constant throughout the domain and equal to 0.0125 mm (Fig. 8.45). And with one particle at the centre of each cell we have a total of about one million particles. Adaptive grid refinement would drastically reduce the number of particles, but we have not implemented this yet.

Contrary to the experiments, the indenter is velocity controlled and not load controlled. This means that the simulation steps are slightly different from the exper-

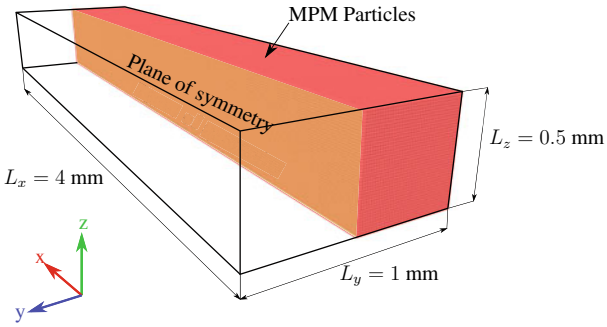


Fig. 8.45 Simulation box (bold lines), plane of symmetry (orange) and particles (de Vaucorbeil et al. 2022a)

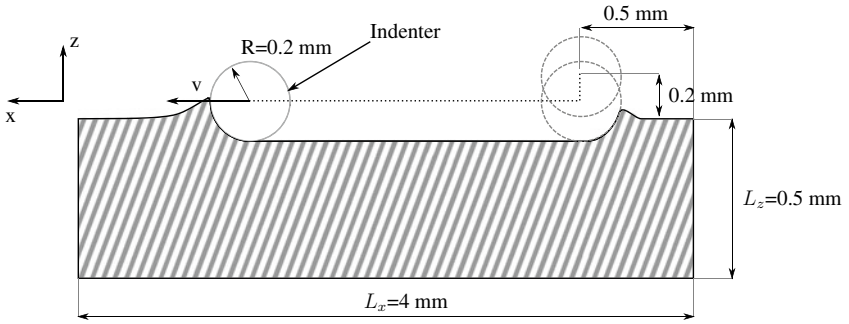


Fig. 8.46 Schematic cut view in the xz plane of the scratch tests setup (de Vaucorbeil et al. 2022a)

imental ones. At the start of the simulation, the indenter is at the limit of being in contact with the top surface of the sample. Thus, the centre of the indenter is located 0.2 mm above the surface, at a distance 0.5 mm of the back surface of the sample (see Fig. 8.46) and in the plane of symmetry of the problem (i.e., xz plane). The three steps of the simulation are as follows:

1. Indentation phase, $0 \leq t \leq t_1$: the indenter is lowered until the desired load is reached. Its vertical velocity is: $v_z = -v_0(1 - \exp(-t))$, while its horizontal position is kept unchanged: $v_x = 0$.
2. Scratching phase, $t_1 \leq t \leq t_2$: the indenter is moved horizontally with a velocity of $v_x = v_0(1 - \exp(-(t - t_1)))$ while its vertical position is kept constant, i.e., $v_z = 0$. This phase ends when the indenter is at 0.5 mm from the front surface.
3. Unloading phase, $t_2 \leq t \leq t_3$: the indenter's horizontal motion is stopped, i.e., $v_x = 0$ and is raised: $v_z = v_0(1 - \exp(-(t - t_2)))$.

The time t_1 corresponds to the time at which the vertical load reaches the desired load. There are no analytical solution for it. As one can see, the velocity profile of the

indenter was choosen such that at the beginning of every step, the acceleration is zero in order to avoid the creation of compressive shock-waves. The maximum velocity v_0 was chosen to 50 m/s. This velocity is much higher than the maximum speed registered during the experiments. However, Leroch et al. (2016) showed that the average forces are independent of the indenter speed. Therefore, since no strain rate effect are taken into account, i.e., $C = 0$, a high value of v_0 was chosen to decrease the necessary computational time.

Virtual indenter. To decrease the computational cost, the spherical indenter is not modelled explicitly as a solid. Instead, it is modelled as a force $\mathbf{f}_p^{\text{ind}}$ exerted on all particles. This is different from Leroch et al. (2016) who modelled the indenter explicitly.

Following Leroch et al. (2016), the indenter is supposed to be a perfect sphere. Therefore, in the absence of friction, $\mathbf{f}_p^{\text{ind}}$ can naturally be derived from the contact force between two particles in TLMPM described in Sect. 8.2. In the frictionless contact algorithm for TLMPM, we recall that the contact force between two particles p and q is given by:

$$\mathbf{f}_{pq} = \begin{cases} \frac{1}{\Delta t^2} \dfrac{m_p m_q}{m_p + m_q} \left(1 - \dfrac{R_p + R_q}{\|\mathbf{x}_{pq}\|}\right) \mathbf{x}_{pq}; & \text{if} \quad \delta = R_p + R_q - \|\mathbf{x}_{pq}\| \geq 0 \\ \mathbf{0}; \quad \text{otherwise} \end{cases} \tag{8.46}$$

where m_p and m_q are their respective masses, R_p and R_q their respective radii and $\mathbf{x}_{pq} = \mathbf{x}_q - \mathbf{x}_p$, and Δt the time-step. Considering that the mass of the indenter is much larger than that of a particle, from Eq. (8.46), one gets:

$$\mathbf{f}_p^{\text{ind},n} = \begin{cases} \dfrac{m_p}{\Delta t^2} \left(1 - \dfrac{R_p + R_{\text{ind}}}{\|\mathbf{x}_{p\text{ind}}\|}\right) \mathbf{x}_{p\text{ind}}; & \text{if} \quad \delta = R_p + R_{\text{ind}} - \|\mathbf{x}_{p\text{ind}}\| \geq 0 \\ \mathbf{0}; \quad \text{otherwise} \end{cases} \tag{8.47}$$

where R_{ind} is the radius of the indenter, and $\mathbf{x}_{p\text{ind}} = \mathbf{x}_{\text{ind}} - \mathbf{x}_p$ is the distance between the centre of the indenter and the particle, respectively. The radius of a particle p is here taken as $R_p = (1/2){V_p}^{1/3}$, with V_p the volume of the particle. We use the superscript n i.e., $\mathbf{f}_p^{\text{ind},n}$ to emphasize that the tangential force is zero. Actually for frictionless contact $\mathbf{f}_p^{\text{ind},n}$ is the total force applied on particle p.

In Eq. (8.47), no friction term is present. de Vaucorbeil and Nguyen (2021a) proposed an extension of Eq. (8.46) for modelling frictional contacts. However, a direct adaptation of this model for the forces exerted by the indenter onto particles proved unstable. Instead, the frictional term is derived according to the algorithm proposed by Wang and Chan (2014) for frictional contact in SPH:

$$\mathbf{f}_p^{\text{ind},\tau} = \begin{cases} \dfrac{m_p}{\Delta t}\mathbf{v}_{p\text{ind}}^{\tau}; & \text{if} \quad |\mathbf{v}_{p\text{ind}}^{\tau}| \leq \mu |\mathbf{v}_{p\text{ind}}^{n}| \\ \mu \dfrac{m_p}{\Delta t}\mathbf{v}_{p\text{ind}}^{n} \dfrac{\mathbf{v}_{p\text{ind}}^{\tau}}{|\mathbf{v}_{p\text{ind}}^{\tau}|}; & \text{otherwise} \end{cases} \tag{8.48}$$

where μ is the friction coefficient, $\mathbf{v}_{p\text{ind}}^{n} = \left[(\mathbf{v}_{\text{ind}} - \mathbf{v}_p) \cdot \mathbf{x}_{p\text{ind}}\right] \mathbf{x}_{p\text{ind}}$ and $\mathbf{v}_{p\text{ind}}^{\tau} = (\mathbf{v}_{\text{ind}} - \mathbf{v}_p) - \mathbf{v}_{p\text{ind}}^{n}$ are the normal and tangential difference of velocity between the indenter and particle p, respectively. The total force applied on particle p is then the sum of $\mathbf{f}_p^{\text{ind},n}$ and $\mathbf{f}_p^{\text{ind},\tau}$.

A quantity of interest is the normal and tangential forces that the indenter exerting on the substrate. We compute them as follows. Since the scratching direction is $\mathbf{x}$ and the out-facing normal of the upper surface of the substrate is $\mathbf{z}$ (see Fig. 8.46), the normal and tangential scratching forces are respectively: $f_{\text{ind},v} = -\sum_{p=1}^{N_p} \left[\mathbf{f}_p^{\text{ind},n} + \mathbf{f}_p^{\text{ind},\tau}\right] \cdot \mathbf{z}$ and $f_{\text{ind},h} = -\sum_{p=1}^{N_p} \left[\mathbf{f}_p^{\text{ind},n} + \mathbf{f}_p^{\text{ind},\tau}\right] \cdot \mathbf{x}$ with N_p being the total number of particles in the domain. Note the negative signs to match what Leroch et al. (2016) did. Therefore, the scratching forces are the forces exerted by the substrate onto the indenter, and not the opposite.

Results: forces. To compare the SPH, TLMPM and the experiments, we plot in Fig. 8.47, the change in the average magnitude of the tangential forces as a function of that of the normal scratching forces for the experiments (blue circles), our TLMPM simulations (green squares) as well as Leroch et al. (2016) SPH simulations (magenta triangles). Even though the magnitude of the tangential scratching force is constantly under-predicted by our TLMPM simulations (without friction), the trend is correctly captured. The constant offset seen between the experimental and TLMPM simulations was suggested by Villumsen and Fauerholdt (2008) to be due to the lack of friction in the simulations. The SPH simulations which were performed without friction do not follow the same trend. While they tend to under-predict the tangential forces at low normal scratching force, they are more accurate at predicting the former

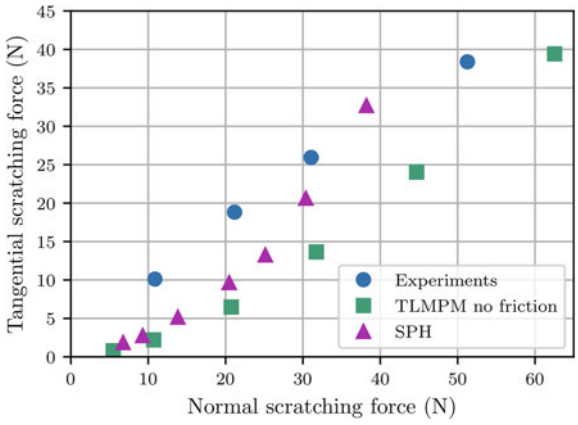


Fig. 8.47 Magnitude of the average tangential forces as a function of the average normal scratching force. The discrepancy is likely to be due to the presence of friction between the indenter and the substrate in the experiments that is not taken into account in the simulations (de Vaucorbeil et al. 2022a)

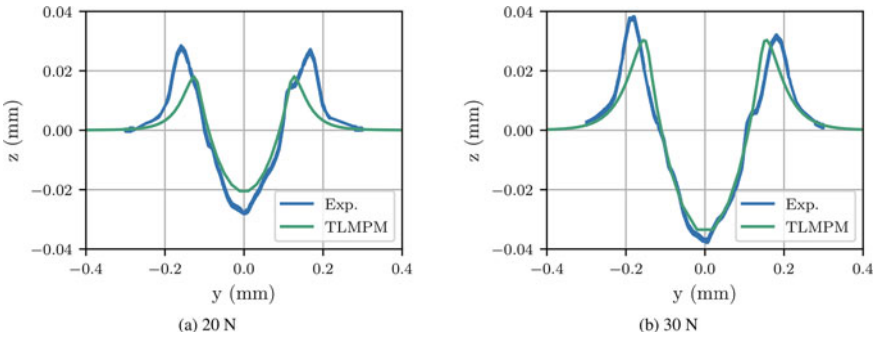


Fig. 8.48 Comparison between the simulated (without friction) and the experimental topographic yz-sections obtained for nominal normal loads of 20 and 30 N (the plane at the middle of the substrate) (de Vaucorbeil et al. 2022a)

at higher loads. However, data is missing to understand what would happen for loads higher than 40 N. If the same trend is followed, however, tangential forces would then be over-predicted. Based on these results, Leroch et al. (2016) argued that the effect of friction is greater at low normal scratching forces, but the results obtained with TLMPM do not suggest that.

Results: scratch groove topologies. As wear manifests itself by the volume of material displaced, it is important to ensure that the simulations are able to capture the profile of scratch grooves. Indeed, a good agreement between the simulations and the experiments is reached (see Fig. 8.48, the cross-section of the scratch groove for respective nominal loads of 20 and 30 N are compared to their experimental counterparts). Importantly, one can observe when comparing Fig. 8.48a and b that the scratch depth, width and shoulder height all increase with the load. Hence, so does the volume of material displaced as expected.

When looking at how the groove profiles evolve along the scratching direction, one can distinguish two different scratch topologies. On the one hand, at low nominal normal loads, i.e., lower or equal to 30 N, the groove is smooth (Fig. 8.49a) and the groove cross section is constant, once the steady state is reached. On the other hand, for higher nominal loads, oscillations in the groove profile appear. These oscillations echo those seen in the force profiles (Fig. 8.49b). The origin of such oscillations seem to be related to the height of the lip formed in front of the moving indenter. During the simulation, it can be clearly seen that the lip starts forming and then is pushed down by the indenter. This suggests that at these simulated loads the rate of formation of the lip is not high enough to form a chip. However, marks of such oscillations cannot be clearly identified from the experimental results published by Leroch et al. (2016). But, oscillations in the retained stresses were observed from their SPH simulations.

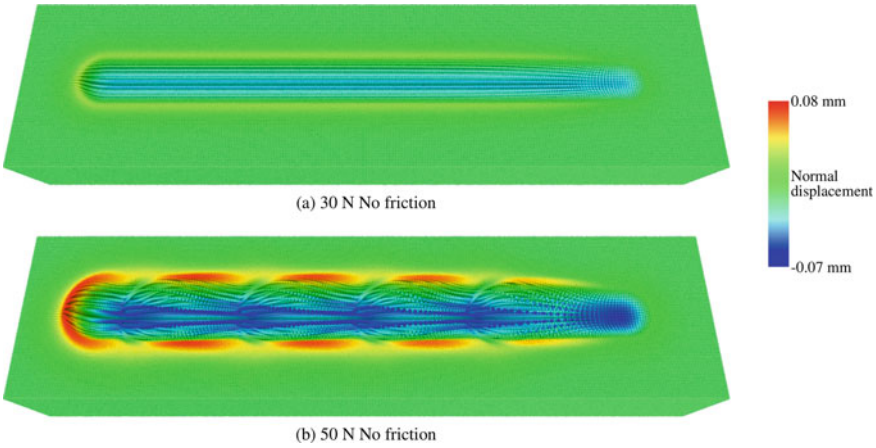


Fig. 8.49 Scratch tests topology obtained with TLMPM without friction considered. Please note that the space between particles do not represent any material separation as the radius of a particle is constant in all representations here (de Vaucorbeil et al. 2022a)

Results: oscillation in scratch groove topologies. Some of the TLMPM simulation results presented above feature oscillatory groove and force patterns. These kinds of oscillations are sometimes observed experimentally (Lin et al. 2001; Wang et al. 2006). These oscillations are nonetheless interesting. They develop at nominal normal loads higher or equal than 20 N and 40 N respectively for simulations with and without friction and shine by its regularity (see Fig. 8.50). They also depend on the load as one can see by comparing Fig. 8.50a and b. The lower the load, the higher the period. Moreover, it has been observed that this trend is independent of the presence of friction.

By observing the results of all the simulations presenting oscillations, it has been found that if at some point in time the equivalent plastic strain at the highest point of the lip formed in front of the indenter reaches a value of 1.5 or above, oscillations will form. In the contrary, if the strain does not reach this threshold value, no oscillation will appear. Moreover, the severity of these oscillations is linked to the work hardening rate. Indeed, the higher the work hardening rate, the lower the severity of these oscillations for a given nominal load. This is not surprising as strain hardening works against local changes so low hardening rates allow bifurcation of the material path in front of the indenter, and high hardening rates prevent such bifurcation to take place.

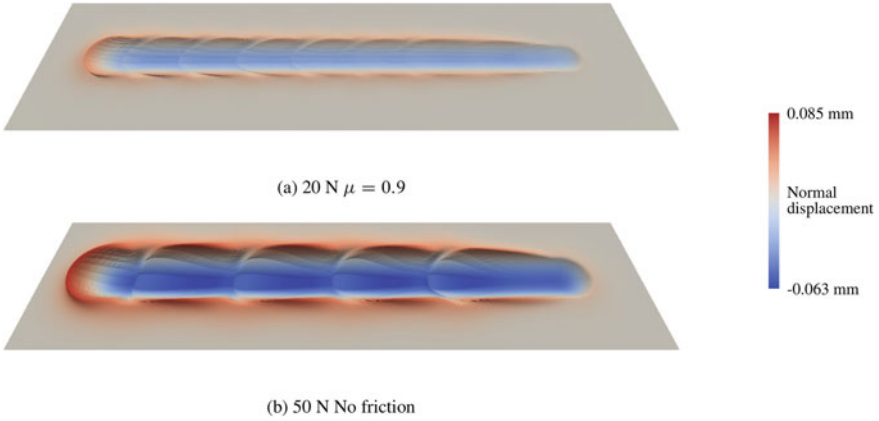


Fig. 8.50 Mesh reconstruction of the deformed upper surface of two samples showing the oscillatory groove topologies obtained with TLMPM with the smallest and highest loads. The colors correspond to the displacement normal to the underformed surface (i.e., in the z direction). Done using Paraview (de Vaucorbeil et al. 2022a)

8.5 Fracture Modeling

As discussed in Sect. 1.5.7, within the framework of continuum mechanics, there are basically two approaches to fracture modeling: a discontinuous approach and a continuous approach. The former approach, which is based on the theory of fracture mechanics, treats cracks as strong discontinuities i.e., real material separation. On the other hand, in a continuous approach, which is mostly based on continuum damage mechanics, strong discontinuities cannot be captured; instead the material properties (such as Young's modulus) are degraded according to some damage laws while the displacement field is always kept continuous everywhere.

To demonstrate the pros and cons of the continuous and discontinuous approach to fracture modeling, let us consider the problem of the growth of ten cracks in a square plate subjected to a bi-axial tension (Fig. 8.51). The crack geometry was taken from Budyn et al. (2004) and tabulated in Table 8.3 for completeness. The material properties are also given in Fig. 8.51 (the fracture toughness $K_c = \sqrt{EG_c}$ for the assumed plane stress condition).

We solve this problem using XFEM–a typical discontinuous approach and a phase field method (PFM), a continuous approach. For the XFEM (extended finite element method) (Moës et al. 1999), we have used the model and code described in Sutula et al. (2017). The code is a serial `Matlab` code. For the phase field method, we adopt `feFRAC`–a parallel `C++` code introduced in Nguyen et al. (2020).

There exists some slight discrepancies in the obtained crack patterns (Fig. 8.52), which we cannot explain as there is no experimental result to verify the results. But this is not the point here. The point is that the XFEM code runs very fast compared with the PFM code even though the latter used multiple CPUs and the former is only

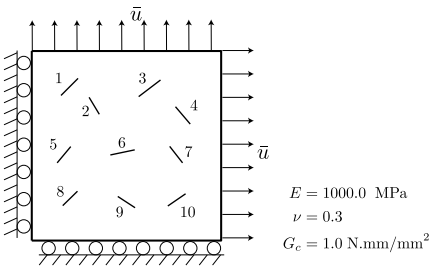
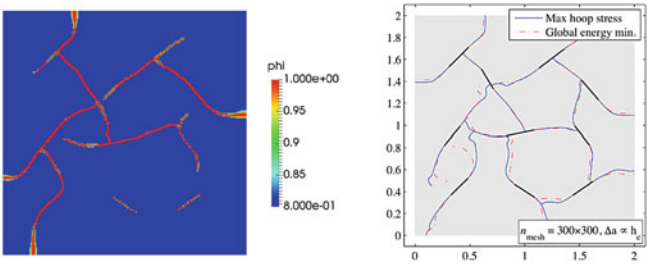


Fig. 8.51 A square plate, of 2×2 mm^2, with 10 random cracks under bi-axial tension

Table 8.3 A plate with 10 random cracks: crack geometry data

Crack ID	x_1	y_1	x_2	y_2
1	0.308514	1.531184	0.488788	1.711458
2	0.605291	1.511563	0.713210	1.332516
3	1.128942	1.518921	1.359495	1.694289
4	1.517694	1.412228	1.673441	1.229502
5	0.268045	0.819902	0.411528	0.991591
6	0.829713	0.903294	1.087246	0.957253
7	1.456377	0.994043	1.592502	0.819902
8	0.326910	0.368605	0.482656	0.520673
9	0.908199	0.465487	1.090925	0.346531
10	1.436755	0.364926	1.624387	0.493693



(a) PFM [Wu et al., 2019] (b) XFEM [Sutula et al., 2017]

Fig. 8.52 A square plate with 10 random cracks under bi-axial tension: PFM versus XFEM. For both analyses, an FE mesh of 300×300 Q4 elements is adopted and for the PFM the length scale is $b = 0.01$ mm. In the legend, "phi" denotes the damage field; thus where damage is close to one, that region denotes the diffuse cracks

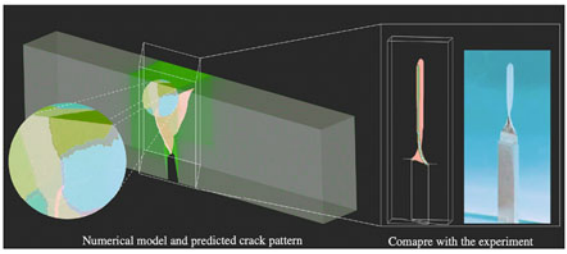


Fig. 8.53 Three dimensional non-planar cracks can be obtained seamlessly with a phase field fracture model (Wu et al. 2021)

a serial code. The distinct pros of XFEM is its coarse mesh efficiency. The cons of XFEM are (1) intricate implementation and (2) the extension to 3D complex crack patterns is difficult. Phase field fracture model is exactly the opposite; it works in 2D and 3D (Fig. 8.53), the implementation is quite straightforward. However the computational cost is huge (Wu et al. 2019).

8.5.1 Fracture Modeling Within the MPM Framework

It is a common misunderstanding that it is easy to model cracks as strong discontinuities in the MPM as it is classified as a meshfree method. On the contrary, it is the fixed background grid in this method that makes it difficult to handle discontinuities in the displacement/velocity fields. To allow discontinuities, one has to introduce multiple velocity fields at the grid nodes in the same manner to duplicated nodes in the FEM. That is why only simple fracture problems, usually 2D ones, are solved using this discontinuous approach (Nairn 2003; Tan and Nairn 2002; Nairn 2007a; Guo and Nairn 2004; Gilabert et al. 2011; Wang et al. 2005). We believe that these problems are better to be solved using the FEM which is more accurate and more efficient. The problem of fracture mechanics is how to represent non-planar 3D crack surfaces. This is more difficult when one considers the merging and branching of these surfaces.

In contrast to other meshfree methods such as EFG, enrichment method using the Partition of Unity is not really picked up by the MPM community. There are only a few work on this topic. For instance, Liang et al. (2017) did present an XFEM way in GIMP, but again analyzed simple 2D fracture problems.

Let us recall the signature applications of the MPM. There are only two: (1) problems involving contacts and very large strain (note that FEM can solve efficiently large displacement fracture problems, see Fig. 8.54) and (2) problems with very complex geometries which are difficult to be converted into good quality FE meshes.

Actually, the MPM is not good for fracture modeling as long as the issue of poor stress accuracy has not been resolved. If a better stress field can be obtained, the MPM can be a promising tool for tackling problems involving fracture, contacts and large deformation. Some problems fall into this category that come to our mind:

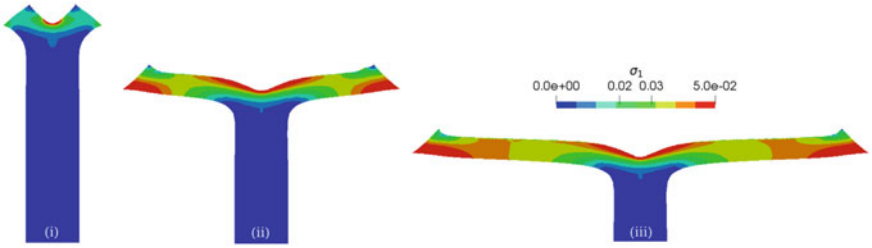


Fig. 8.54 Finite element modeling of large displacement fracture of biological tissues using a phase-field fracture model (Mandal et al. 2020b)

wearing, drilling, and penetration. Li et al. (2021) have recently presented a 3D MPM framework for modeling snow avalanches. The model adopts a continuous approach to fracture.

We are left with the continuous approach to fracture which is mostly based on damage mechanics. In its simplest form $\boldsymbol{\sigma} = (1 - d)\bar{\boldsymbol{\sigma}}$ where d is a scalar damage field and $\bar{\boldsymbol{\sigma}}$ is the effective stress tensor. Even though this damage based model is simple, it suffers from mesh bias and mesh dependence i.e., the results are sensitive to the mesh. Nonlocal models can overcome this issue and we present variational fracture models in Sect. 8.5.2 including the phase-field approximation, the eigen-erosion approximation and a gradient enhanced Johnson-Cook damage model (Sect. 8.5.4).

8.5.2 Variational Fracture Theories

Griffith's fracture theory. In 1921, Griffith conducted experiments on fracture of glass fibers Griffith (1920). He found two things: (1) the fracture strength of glass is significantly smaller than the theoretical value (coming from breaking the atomic bonds) and (2) small glass fibers are stronger than larger fibers. He concluded that small naturally occurring defects existing in the fibers make them weak. Precisely these defects amplify the stress field in front of their tips and thus rendering the fracture stress much smaller compared with the theoretical strength. Actually Griffith was aware of the work of Charles Inglis conducted seven years ago about stress concentration due to an elliptical hole (Inglis 1913).

With defects now in his mind, he made specimens with artificial surface cracks (to overcome natural defects) of varying sizes and quantified the relationship between the remote tensile stress σ and the crack size or length a. What he found is an inverse relation between the strength and the crack length. In symbols, he found that $\sigma\sqrt{a} = C$ where C is a constant. To find this constant, he carried out an energy-based analysis that basically led to the born of what is now known as fracture mechanics.

Griffith computed the energy of the system which consists of the stored elastic strain energy and the surface energy i.e., energy due to the creation of the new crack surfaces. He considered a unit thickness infinite plate with a surface crack of length a subjected to a remote tensile stress σ normal to the crack. The energy of this system is given by

$$U = U_0 - \pi a^2 \frac{\sigma^2}{2E_0} + 2a\gamma_s \tag{8.49}$$

where he used Inglis' solution to obtain $-\pi a^2\sigma^2/(2E_0)$ the elastic strain energy released due to the crack's presence; U_0 is the elastic strain energy of the plate without crack. What is interesting here is the last term $2a\gamma_s$ the surface energy associated with a crack of length a where γ_s is the energy required to creare a unit surface area.

The first derivative of U with respect to the crack length a is

$$\frac{\partial U}{\partial a} = -2\pi a \frac{\sigma^2}{2E_0} + 2\gamma_s \tag{8.50}$$

And the vanishing derivative condition gives us

$$\frac{\partial U}{\partial a} = 0 \Longrightarrow \pi a \frac{\sigma^2}{2E_0} = \gamma_s \Longrightarrow \sigma = \sqrt{\frac{2E_0\gamma_s}{\pi a}} \tag{8.51}$$

which is the well-known Griffith's equation relating the remote stress to the crack length.

The work of Griffith which is applicable only to brittle materials (e.g. glasses) was ignored for almost 20 years. It was not until the modifications made by Orowan and particularly George Rankin Irwin (1907–1998) that, a new field has emerged: Linear Elastic Fracture Mechanics (LEFM). Irwin introduced the concept of energy release rate $\mathcal{G}$ which is the negative of the derivative of the elastic strain energy with respect to the crack length and he and Orowan replaced γ_s by G_c — the critical energy release rate or fracture energy — to take into account other dissipative processes such as plastic deformation, i.e.,

$$\mathcal{G} = -\frac{\partial}{\partial a}\left(U_0 - \pi a^2 \frac{\sigma^2}{2E_0}\right) = 2\pi a \frac{\sigma^2}{2E_0}, \qquad G_c = \gamma_s \tag{8.52}$$

Another significant contribution to fracture mechanics was made by James Rice in 1968, the famous J-integral Rice (1968) which is equal to the fracture energy release rate $\mathcal{G}$.

By recalling the irreversibility of the crack propagation, $\dot{a} \geq 0$, the Griffith crack propagation criterion is given then:

$$\mathcal{G} - G_c \leq 0, \quad \dot{a} \geq 0, \quad \dot{a}\big(\mathcal{G} - G_c\big) \equiv 0 \tag{8.53}$$

That is, for quasi-static loading case, the crack propagates (i.e., $\dot{a} > 0$) if $\mathcal{G} = G_c$ and otherwise remains stationary $\dot{a} = 0$ for $\mathcal{G} < G_c$.

Fracture mechanics has been a great success as it provides the engineers a continuum mechanics tool to quantitatively predict the structural integrity of large structures using data such as fracture toughness (a concept introduced by Irwin which is related to the fracture energy) which can be experimentally measured using laboratory scale specimens. Furthermore it helps material scientists to improve existing materials and design new ones by looking at their fracture toughness.

In conclusion, Griffith presented an energy approach to modeling fracture of brittle solids in which fracture is the outcome of the competition between the elastic energy stored in an elastic solid and the surface energy. This was a milestone in the field of mechanics of solids as it started a field coined *fracture mechanics*. However, Griffith's model is incapable of treating crack nucleation (i.e., a crack is initiated in an intact solids). Furthermore, it is not self-contained as it must be used together with a fracture criterion that tells in which direction a crack would go.

Variational fracture mechanics. To solve these issues, Francfort and Marigo (1998) has presented a variational fracture model, in which one seeks simultaneously the displacement field $\mathbf{u}(\mathbf{x}, t)$ and the crack sets $\Gamma(t)$ by minimizing the following energy functional

$$\mathscr{E}(\mathbf{u}, \Gamma) = \int_{\Omega} \psi_0(\boldsymbol{\epsilon}(\mathbf{u}))\mathrm{d}V + \int_{\Gamma} G_c \mathrm{d}A - \mathscr{P}(\mathbf{u}) \tag{8.54}$$

where G_c is the fracture energy. The above energy functional consists of three terms: the first term is the stored strain energy, the second is the surface energy and the final term is the force energy. This variational approach to fracture can be referred to as a generalization of Griffith's theory for brittle fracture. Unfortunately, it is difficult to directly solve Eq. (8.54) and thus various approximations of the variational fracture model have been developed. In what follows, we discuss the two most popular ones: phase-field fracture and eigen-erosion. The commonality of these two approximations is the introduction of another field and the approximation of the crack surfaces Γ by a volume.

Phase-field fracture. Bourdin et al. (2000) presents a regularization of Francfort and Marigo's model by introducing a scalar field $0 \leq d(\mathbf{x}, t) \leq 1$ defined over the entire solid Ω and approximate the troubling surface integral $\int_{\Gamma} G_c \mathrm{d}A$ by a volume one. The energy functional is now given by

$$\mathscr{E}(\mathbf{u}, d) = \int_{\Omega} \omega(d)\psi_0(\boldsymbol{\epsilon}(\mathbf{u}))\mathrm{d}V + \int_{\Omega} G_c \gamma(d, b; \nabla d)\mathrm{d}V - \mathscr{P}(\mathbf{u}) \tag{8.55}$$

where $\omega(d)$ is a degradation function to reduce the strain energy when $d > 0$ (as this part was converted to the surface energy); $\gamma(d, b; \nabla d)$ is the crack surface

density function with b being a small positive number playing the role of a length scale. One must define $\gamma(d, b; \nabla d)$ such that $\int_\Omega G_c \gamma(d, b; \nabla d) \mathrm{d}V = \int_\Gamma G_c \mathrm{d}A$ as b approaches zero. Section 8.5.3 defines γ.

The problem of solving a fracture problem be it crack nucleation, propagation, branching or merging boils down to solving a two-field PDE defined by the functional given in Eq. (8.55):

$$\begin{cases} \nabla \cdot \boldsymbol{\sigma} + \boldsymbol{f} = \rho_0 \ddot{\mathbf{u}} & \text{in } \Omega_0 \\ \omega'(d) \dfrac{\bar{Y}}{G_c / c_\alpha} + \left(\frac{1}{b} \alpha'(d) - 2b \Delta d \right) + \xi \dot{d} \geq 0 \quad \dot{d} \geq 0 \text{ in } \Omega_0 \end{cases} \tag{8.56a}$$

with natural boundary conditions,

$$\begin{cases} \boldsymbol{\sigma} \cdot \mathbf{n} = \boldsymbol{t}^* & \text{on } \partial\Omega_t \\ \frac{2b}{c_\alpha} \nabla_0 d \cdot \mathbf{n} = 0 & \text{on } \partial\Omega \end{cases} \tag{8.56b}$$

where ξ is a damping parameter (Kuhn and Müller 2010). Note that one can perfectly use $\xi = 0$; the role of ξ is merely numerical: it stablizes the algorithm and it allows an explicit solver. We refer to the review of Wu et al. (2019) for details.

Eigen-erosion fracture. Schmidt et al. (2009) presented a two-field approximation of Eq. (8.54) using the eigendeformation $\boldsymbol{\epsilon}^*$. The energy functional now reads

$$\mathscr{E}(\mathbf{u}, \boldsymbol{\epsilon}^*) = \int_\Omega \psi_0(\boldsymbol{\epsilon}(\mathbf{u}) - \boldsymbol{\epsilon}^*) \mathrm{d}V + G_c \frac{|C_\kappa|}{2\kappa} - \mathscr{P}(\mathbf{u}) \tag{8.57}$$

where $|C_\kappa|$ is the volume of the approximate crack surface C_κ (Fig. 8.55), and κ is a length scale similar to b in a PFM. By limiting the eigendeformation to a binary state consisting of zero for intact material and $\boldsymbol{\epsilon}(\mathbf{u})$ for fractured material, Pandolfi and Ortiz (2012); Pandolfi et al. (2014) have introduced an eigen-erosion method for brittle fracture.

Thanks to this binary state one does not need to solve an extra PDE to determine the crack surfaces as in the PFM (see Sect. 8.5.3 for detail). Actually, whether a material point is fractured or not is simply based on a calculation of the particle energy as follows (Li et al. 2012)

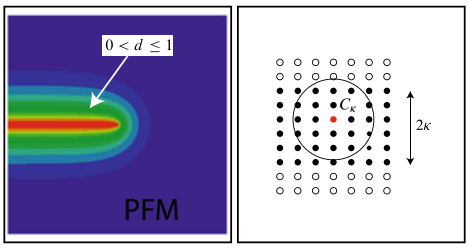


Fig. 8.55 Approximation of the crack surface by a continuous phase-field $d(\boldsymbol{x}, t)$ and a binary 0-1 eigendeformation (right). Black dots represent fractured (or eroded) particles

$$G_p = \frac{\beta\kappa}{\bar{m}_p} \sum_{q \in C_\kappa^p} m_q \psi_0(\boldsymbol{\epsilon}_q), \quad \bar{m}_p = \sum_q m_q \tag{8.58}$$

where β is a normalizing constant, C_κ^p denotes the κ—neighborhood of p, which is a circle/sphere centered at p with a radius of κ for 2D and 3D, respectively. Note that this κ—neighborhood of p excludes eroded particles.

The fracture criterion is then simply as

$$G_p \geq G_c \Rightarrow \text{p is eroded} \tag{8.59}$$

and when a particle is eroded, its stress is zero.

8.5.3 *Implementation of Variational Fracture Phase-Field Model*

Let us confine to second-order phase-field models in which the crack density function is generally written as (Wu 2017; Wu and Nguyen 2018)

$$\gamma(d; \nabla_0 d) = \frac{1}{c_\alpha}\left[\frac{1}{b}\alpha(d) + b\nabla_0 d \cdot \nabla_0 d\right] \tag{8.60}$$

in which concrete expressions for $\alpha(d)$, c_α and $\omega(d)$ are given in Table 8.4 for common models. The models are called second order because Eq. (8.60) involves the second spatial derivative of the damage field. We refer to Mandal et al. (2019b) for a comparative study of these different models.

As the MPM flowchart is more suitable for a staggered solver than a monolithic solver, we only present the staggered solver herein, and thus we focus on the damage sub-problem (the displacement sub-problem is solved using the usual MPM way).

Table 8.4 Common phase field models for brittle and cohesive fracture. TSL is short for traction-separation law, see e.g. Elices et al. (2002). Irwin's internal length is defined as $l_{\text{ch}} := E_0 G_c / f_{\text{t}}^2$, with E_0 being Young's modulus of the material; f_{t} and G_c being the failure strength and fracture energy, respectively. Notice that the models were presented in chronological order

Model	$\alpha(d)$	$\omega(d)$	Fracture type	Length-scale	Parameters
AT2	d^2	$(1-d)^2$	Brittle	$b = \frac{27}{256} l_{\text{ch}}$	$E_0, \nu_0, G_{\text{f}}, b$
AT1	d	$(1-d)^2$	Brittle	$b = \frac{3}{8} l_{\text{ch}}$	$E_0, \nu_0, G_{\text{f}}, b$
PF-CZM	$2d - d^2$	$\frac{(1-d)^p}{(1-d)^p + Q(d)}$	Brittle/cohesive	Numerical num.	E_0, ...+ TSL

We introduce a viscosity term to the fracture energy so that the damage sub-problem becomes a parabolic equation instead of an elliptic one. The reason for that is that we can advance the damage problem in time using an explicit solver. We refer to Kakouris and Triantafyllou (2017a, b); Cheon and Kim (2019) for a coupling of an explicit MPM with an implicit phase-field damage. In the computer graphics community, a similar idea has been recently proposed in Wolper et al. (2019) but for ductile fracture.

The weak form for the modified damage sub-problem is given by (note that we adopt a total Lagrangian for the damage equation)

$$\int_{\Omega_0} \left[\omega'(d)\bar{Y}\delta d + \frac{G_c}{c_\alpha}\left(\frac{\alpha'(d)}{b}\delta d + 2b\nabla_0 d\cdot\nabla_0\delta d\right) + \xi\dot{d}\delta d \right] \mathrm{d}V = 0 \tag{8.61a}$$

where $\bar{Y}(\boldsymbol{\epsilon})$ is the effective crack driving force, in the simplest case it is simply the elastic energy i.e., $\bar{Y} = \psi_0(\boldsymbol{\epsilon})$. Using the usual FE approximation for $\delta d = N_I \delta d_I$, one gets the following equation at node I

$$\int_{\Omega_0} \left[\omega'(d)\bar{Y}N_I + \frac{G_c}{c_\alpha}\left(\frac{\alpha'(d)}{b}N_I + 2b(\nabla_0 d\cdot\nabla_0 N_I)\right) + \xi\dot{d}N_I \right] \mathrm{d}V = 0 \tag{8.62a}$$

Then, we use the forward Euler method for $\dot{d}$ to arrive at the final equation as

$$m_I d_I^{t+\Delta t} = m_I d_I^t - \frac{\Delta t}{\xi}\int_{\Omega_0}\left[\omega'(d^t)\bar{Y}N_I + \frac{G_c}{c_\alpha}\left(\frac{\alpha'(d^t)}{b}N_I + 2b(\nabla_0 d\cdot\nabla_0 N_I)\right)\right]\mathrm{d}V \tag{8.63}$$

where m_I is the pseudo-mass at node I, which is given by (obtained using the well known row-sum method similar to the mass matrix)

$$m_I := \int_{\Omega_0} N_I \mathrm{d}V \tag{8.64}$$

In the MPM, Eq. (8.63) is solved on the Eulerian grid, and thus $N_I = \phi_I$. In GPIC, we solve this equation on the FE mesh, so $N_I = \phi^{\mathrm{FE}}$. Herein we only present the details for ULMPM.

Finally, the integral in Eq. (8.63) is evaluated using particles as integration points. This results in the final equation to update the grid damage:

$$d_I^{t+\Delta t} = d_I^t - \frac{\Delta t}{m_I\xi}\sum_p\left[\omega'(d^t)\bar{Y}N_I(\mathbf{X}_p) + \frac{G_c}{bc_\alpha}\left(\alpha'(d^t)N_I(\mathbf{X}_p) + 2b^2(\nabla_0 d\cdot\nabla_0 N_I(\mathbf{x}_p))\right)\right]V_p^0 \tag{8.65}$$

For implementation, we provide the complete ULMPM-PFM algorithm in Algorithm 16.

Algorithm 16 Solution procedure of explicit ULMPM-PFM.

while $t < t_f$ **do**
  Solving the displacement problem: 2 changes
    $\mathbf{f}_I^{\text{int},t} = -\sum_p \omega(d_p^t) V_p^t \boldsymbol{\sigma}_p^t \nabla \phi_I(\mathbf{x}_p^t)$
    In G2P step, also update the crack driving force $\bar{Y}$ for all particles
  end
  Solving the damage equation
    Update damage force 1: $F_{1I} = \sum_p \omega'(d^t) \bar{Y} N_I(\mathbf{X}_p)$
    Update damage force 2: $F_{2I} = \sum_p {}^{G_c}/_{bc_\alpha} \left(\alpha'(d^t) N_I(\mathbf{X}_p) + 2b^2 (\nabla_0 d \cdot \nabla_0 N_I(\mathbf{x}_p))\right)$
    Update damage force $F_I = F_{1I} + F_{2I}$
    Update grid damage: $d_I^{t+\Delta t} = d_I^t - \frac{\Delta t}{m_I \xi} F_I$
    Update particle damage: $d_p^{t+\Delta t} = \sum_I N_I(\mathbf{X}_p) d_I^{t+\Delta t}$
  end
  Advance time $t = t + \Delta t$
end while

We only implemented this phase field model in our `Julia` MPM code which is not parallelized; the code is slow for intensive phase field simulations. Thus, we do not present examples on this. We refer to Kakouris and Triantafyllou (2017a, b); Cheon and Kim (2019) for the performance of MPM-PFM.

8.5.4 Nonlocal Johnson-Cook Damage Models

The Johnson-Cook viscoplastic-damage model is widely used to model the deformation and fracture of metals. It is assumed that the inclusion of strain rate in the model helps to regularize the problems caused by strain localization. However, this is not true. We tested this hypothesis by simulating the fracture of a W700E asymmetrically notched specimen loaded in tension. We used the model described in Sect. 4.3. The results given Fig. 8.56 show that the result is mesh dependent and the crack pattern is a bit biased by the mesh orientation.

To mitigate the mesh-related issues of this local model, we present herein a nonlocal gradient enhanced formulation of the Johnson-Cook model described in Sect. 4.3. In this nonlocal model, the damage variable is given by de Vaucorbeil et al. (2022b)

$$D = \begin{cases} 0 & \text{when } 0 \leq \langle D_{\text{init}} \rangle < 1 \\ 10\left(\langle D_{\text{init}} \rangle - 1\right) & \text{when } \langle D_{\text{init}} \rangle \geq 1 \end{cases} \tag{8.66}$$

where $\langle D_{\text{init}} \rangle = \sum \langle \Delta D_{\text{init}} \rangle$ is the nonlocal damage initiation variable.

The gradient enhanced nonlocal damage formulation is obtained by adding a second order differential equation linking the nonlocal variable to the local variable

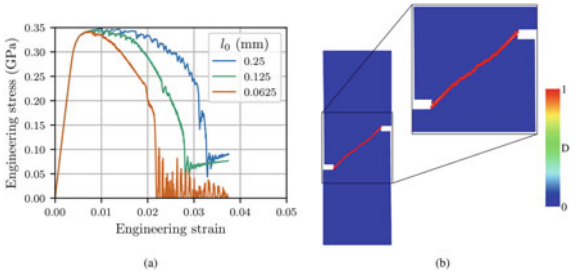


Fig. 8.56 Failure of a W700E asymmetrically notched specimen loaded in tension using a local damage model with strain rate for both the flow stress and the damage criteria taken into account: a. Stress-strain curves obtained for different background grid cell size illustrating the non-convergence of the onset of damage. Mesh bias is still there as illustrated in b. A regular square grid was used (de Vaucorbeil et al. 2022b)

(Peerlings et al. 2002). In a total Lagrangian setting, this differential equation and for our variable is given as::

$$\langle \Delta D^{\text{init}} \rangle - c_0 \nabla_0^2 \langle \Delta D^{\text{init}} \rangle = \Delta D^{\text{init}} \tag{8.67}$$

where ∇_0^2 is the Laplacian operator in the reference configuration, and c_0 equals $l_d^2/16$ in two dimensions and $l_d^2/18$ in three dimensions, with l_d being the nonlocal length scale used in the integral type nonlocal model.

In order to solve this equation, one first needs to obtain its weak form. Multiplying it by the test damage initiation increment function $\delta\langle \Delta D^{\text{init}} \rangle$ and integrating over the whole domain in the reference configuration Ω_0, one gets:

$$\int_{\Omega_0} \delta\langle \Delta D^{\text{init}} \rangle \left[\langle \Delta D^{\text{init}} \rangle - c_0 \nabla_0^2 \langle \Delta D^{\text{init}} \rangle - \Delta D^{\text{init}} \right] \mathrm{d}\Omega_0 = 0 \tag{8.68}$$

After integrating by parts the second term, applying the divergence theorem and incorporating the boundary condition $\overrightarrow{\nabla}_0 \langle \Delta D^{\text{init}} \rangle \cdot \mathbf{n} = 0$ on the boundary Γ_0 gives the weak form:

$$\begin{aligned} &\int_{\Omega_0} \delta\langle \Delta D^{\text{init}} \rangle \langle \Delta D^{\text{init}} \rangle \mathrm{d}\Omega_0 + \int_{\Omega_0} c_0 \nabla_0 \delta\langle \Delta D^{\text{init}} \rangle \cdot \overrightarrow{\nabla}_0 \langle \Delta D^{\text{init}} \rangle \mathrm{d}\Omega_0 \\ &\quad = \int_{\Omega_0} \delta\langle \Delta D^{\text{init}} \rangle \Delta D^{\text{init}} \mathrm{d}\Omega_0 \end{aligned} \tag{8.69}$$

In the spirit of the MPM, the particles are used as integration points, therefore Eq. (8.69) leads to the following discrete equation:

$$\sum_{p=1}^{N_p} \delta\langle \Delta D^{\text{init}}\rangle_p \langle \Delta D^{\text{init}}\rangle_p V_p^0 + \sum_{p=1}^{N_p} c_0 \nabla_0 \delta\langle \Delta D^{\text{init}}\rangle_p \cdot \nabla_0 \langle \Delta D^{\text{init}}\rangle_p V_p^0 = \sum_{p=1}^{N_p} \delta\langle \Delta D^{\text{init}}\rangle_p \Delta D_p^{\text{init}} V_p^0 \tag{8.70}$$

where V_p^0 is the volume of particle p in the reference configuration. Next, using the same grid weighting function used for the balance of momenta equation, $\delta\langle \Delta D^{\text{init}}\rangle_p$ and $\overrightarrow{\nabla}_0 \delta\langle \Delta D^{\text{init}}\rangle_p$ are both interpolated from the nodal values of the background mesh as:

$$\delta\langle \Delta D^{\text{init}}\rangle_p = \sum_{I=1}^{N_I} \Phi_I(\mathbf{X}_p)\delta\langle \Delta D^{\text{init}}\rangle_I \tag{8.71}$$

$$\nabla_0 \delta\langle \Delta D^{\text{init}}\rangle_p = \sum_{I=1}^{N_I} \nabla_0 \Phi_I(\mathbf{X}_p)\delta\langle \Delta D^{\text{init}}\rangle_I \tag{8.72}$$

where I designate the nodes of the background grid and N_I their number.

Substituting Eqs. (8.71) and (8.72) into Eq. (8.70) and rearranging gives:

$$\sum_{I=1}^{N_I} \delta\langle \Delta D^{\text{init}}\rangle_I \left[\sum_{p=1}^{N_p} \Phi_I(\mathbf{X}_p)\langle \Delta D^{\text{init}}\rangle_p V_p^0 + \sum_{p=1}^{N_p} c_0 \nabla_0 \Phi_I(\mathbf{X}_p) \cdot \nabla_0 \langle \Delta D^{\text{init}}\rangle_p V_p^0 - \sum_{p=1}^{N_p} \Phi_I(\mathbf{X}_p)\Delta D_p^{\text{init}} V_p^0 \right] = 0 \tag{8.73}$$

Since Eq. (8.73) is valid whatever the test function chosen, it must therefore hold that

$$\sum_{p=1}^{N_p} \Phi_I(\mathbf{X}_p)\langle \Delta D^{\text{init}}\rangle_p V_p^0 + \sum_{p=1}^{N_p} c_0 \nabla_0 \Phi_I(\mathbf{X}_p) \cdot \nabla_0 \langle \Delta D^{\text{init}}\rangle_p V_p^0 - \sum_{p=1}^{N_p} \Phi_I(\mathbf{X}_p)\Delta D_p^{\text{init}} V_p^0 = 0 \tag{8.74}$$

Considering:

$$\langle \Delta D^{\text{init}}\rangle_p = \sum_{J=1}^{N_I} \Phi_J(\mathbf{X}_p)\langle \Delta D^{\text{init}}\rangle_J \tag{8.75}$$

$$\nabla_0 \langle \Delta D^{\text{init}}\rangle_p = \sum_{J=1}^{N_I} \nabla_0 \Phi_J(\mathbf{X}_p)\langle \Delta D^{\text{init}}\rangle_J \tag{8.76}$$

Equation (8.74) becomes after factorisation:

$$\sum_{J=1}^{N_I} \left[\sum_{p=1}^{N_p} V_p^0 \left(\Phi_I(\mathbf{X}_p)\Phi_J(\mathbf{X}_p) + c_0 \overrightarrow{\nabla}_0 \Phi_I(\mathbf{X}_p) \cdot \overrightarrow{\nabla}_0 \Phi_J(\mathbf{X}_p) \right) \right] \langle \Delta D^{\text{init}}\rangle_J = \sum_{p=1}^{N_p} V_p^0 \Phi_I(\mathbf{X}_p)\Delta D_p^{\text{init}} \tag{8.77}$$

Collecting terms in Eq. (8.77) gives rise to the simple form:

$$\mathbf{K}^D\mathbf{D} = \mathbf{F}^D \tag{8.78}$$

where $\mathbf{D}$ is the nodal damage initiation increment vector i.e., $D_J = \langle \Delta D^{\text{init}} \rangle_J$, $\mathbf{K}^D$ is an $(N_I \times N_I)$ coefficient matrix whose $K^D_{I,J}$ component is defined as

$$K^D_{I,J} = \sum_{p=1}^{N_p} V_p^0 \left(\Phi_I(\mathbf{X}_p)\Phi_J(X_p) + c_0 \overrightarrow{\nabla}_0 \Phi_I(\mathbf{X}_p) \cdot \overrightarrow{\nabla}_0 \Phi_J(\mathbf{X}_p) \right) \tag{8.79}$$

and $\mathbf{F}^D$ is a force-like vector, which is given by

$$F^D_I = \sum_{p=1}^{N_p} V_p^0 \Phi_I(\mathbf{X}_p) \Delta D_p^{\text{init}} \tag{8.80}$$

It acts as the damage initiation driving force.

The integration of the gradient damage enhanced formulation into the TLMPM algorithm (see Algorithm 4) is performed as follows: (1) The matrix $\mathbf{K}^D$ is built during the initialisation part and is done only once, (2) During the G2P step, after the update of the stress tensors, the local damage initiation increment is calculated, then the vector $\mathbf{F}^D$ is updated and Eq. (8.78) is solved. The local damage update in Algorithm 9 is replaced by the new nonlocal counterpart in Algorithm 17.

Algorithm 17 Damage algorithm (nonlocal gradient formulation).

Inputs: $\Delta\varepsilon_p^{t+\Delta t}$ (incremental equivalent plastic strain), $\boldsymbol{\sigma}_{t+\Delta t}$, $\langle D^t_{\text{init}} \rangle$ (damage initiation variable)
Outputs: $\langle D^{t+\Delta t}_{\text{init}} \rangle$ (updated damage initiation variable), $D^{t+\Delta t}$ (updated damage)
$\sigma^* = -\hat{p}/\sigma_{\text{eq}}$ ▷ Compute stress triaxiality
$\varepsilon_f = \left[D_1 + D_2 \exp(D_3\sigma^*)\right][1 + D_4 \ln(\dot{\varepsilon}^*_p)]$ ▷ Strain at failure
Compute the local damage initiation increment: $\Delta D^{\text{init}} = \frac{\Delta\varepsilon_p^{t+\Delta t}}{\varepsilon_f}$
Update $\mathbf{F}^D$: $F^D_I = \sum_{p=1}^{N_p} V_p^0 \Phi_I(\mathbf{X}_p)\Delta D^{\text{init}}$
Inverse the nonlocal system: $\mathbf{D} = \left(\mathbf{K}^D\right)^{-1}\mathbf{F}^D$ ▷ $\mathbf{D}_J = \langle \Delta D^{\text{init}} \rangle_J$
Compute the damage initiation increment at particle p: $\langle \Delta D^{\text{init}} \rangle = \sum_{J=1}^{N_I} \Phi_J(\mathbf{X}_p)\langle \Delta D^{\text{init}} \rangle_J$
$\langle D^{t+\Delta t}_{\text{init}} \rangle = \langle D^t_{\text{init}} \rangle + \langle \Delta D_{\text{init}} \rangle$
if $\langle D^{t+\Delta t}_{\text{init}} \rangle \geq 1$ **then** ▷ Damage has initiated
    $D^{t+\Delta t} = 10\left(\langle D^{t+\Delta t}_{\text{init}} \rangle - 1\right)$
else ▷ Damage has not initiated
    $D^{t+\Delta t} = 0$
end if

Thus, in a computational cycle of the MPM, whereas the balance of momentum equation is solved explicitly, the damage equation, Eq. (8.78), is solved implicitly.

The good thing is that this equation is a linear one and is solved using fast and efficient linear solvers provided by the PETSC library.

Remark 46 Algorithmically, our formulation is similar to the phase-field fracture of Kakouris and Triantafyllou (2017b) which is also implemented in an MPM (ULMPM precisely). However a phase-field model is a regularised fracture mechanics formulation whereas ours is actually a purely damage mechanics formulation. We refer to Mandal et al. (2019a) for details.

8.6 Some Fracture Simulations

This section presents some fracture simulations using the TLMPM and the Johnson-Cook damage models. In Sect. 8.6.1 the necking and fracture of a smooth cylinder specimen made of Weldox steel alloys is studied using a local JC model. Next, common fracture tests are provided in Sects. 8.6.2, 8.6.3, solved with a nonlocal JC model demonstrating the mesh objectivity of the formulation. All these simulations are quasi-static. In Sect. 8.6.5 a dynamic fracture simulation is presented: the fracture of a Weldox steel alloys plate penetrated by a blunt bullet.

8.6.1 Tensile Test Specimen Experiencing Necking and Damage

As a simplest demonstration for problems exhibiting large plastic deformations and eventually fracture, we present 3D simulations of smooth cylindrical tensile samples made of ductile Weldox steels all the way to failure. The MPM results will be compared with experimental results given in Dey et al. (2004, 2006) and also with the FEM (using Abaqus Explicit). The ability to handle very large deformation and fracture is important for a large range of important engineering problems such as wear, material penetration etc. As we have previously demonstrated, the TLMPM is the most efficient and accurate MPM to date,[3] we thus use it for these simulations. As will be seen, the TLMPM is stable when material instabilities occur as experienced during necking, and it can simulate damage and fracture of ductile materials without requiring any algorithm changes, contrary to Total-Lagrangian Smoothed Particle Hydrodynamics (de Vaucorbeil and Hutchinson 2020). In this section, we use the local Johnson-Cook damage model.

The tensile specimens are 3D smooth cylinders of 30 mm in length and 6 mm in diameter made of three different Weldox steel alloys: W460E, W700E, and W900E. These materials were selected because material parameters for the widely used Johnson-Cook constitutive model (see Sect. 4.3) are available (Dey et al. 2004).

[3] We anticipate that GPIC might be slightly better for this problem.

Table 8.5 Material parameters for Weldox steels. ρ_0 is the reference bulk density, E Young modulus, ν Poisson's ratio, $c_0 = \sqrt{E/(2(1-2\nu)\rho_0)}$ is the bulk speed of sound, Γ_0 Grüneisen Gamma in the reference state

ρ_0 [kg/m^3]	E (GPa)	ν	c_0 [m/s]	S_α	Γ_0
7750	211	0.33	5166	1.5	0

Table 8.6 Material constants for the Johnson-Cook constitutive model and damage criterion as proposed by Dey et al. (2006)

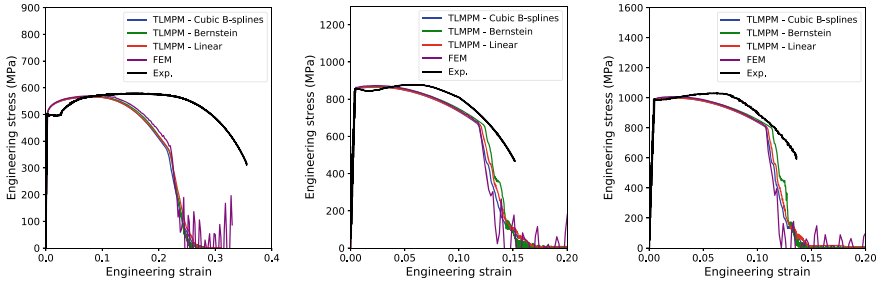
Material	Yield stress	Strain hardening			Damage			
	A (MPa)	B (MPa)	n	C	D_1	D_2	D_3	D_4
Weldox 460E	499	382	0.458	0	0.636	1.936	–2.969	–0.0140
Weldox 700E	859	329	0.579	0	0.361	4.768	–5.107	–0.0013
Weldox 900E	992	364	0.568	0	0.294	5.149	–5.583	0.0023

For these three alloys, their material parameters are directly taken from the literature and are listed in Tables 8.5 and 8.6. All simulations are supposed to be quasi-static. Therefore, the influence of the strain rate is not taken into account which is why $C = 0$ for all.

For each of the three materials, the TLMPM simulations are performed using each of the three different shape functions presented in Sect. 3: linear (hat functions), cubic B-splines, and quadratic Bernstein polynomials. For all these simulations, the specimens were discretized using a single particle for each cell of the background mesh, when they lie within the solid's boundary surface. The nodes of the background mesh coincident with the top and bottom faces of the cylinders were subjected to a velocity $\pm v$ of the form $v_{max}(1 - e^{-t})$ along the specimen's axis. A high velocity of $v_{max} = 1$ mm/ms was chosen to decrease the computational time and has no impact on the results as no strain rate effect was taken into account.

The results of these simulations are presented in Fig. 8.57, alongside results from FEM simulations (carried out by the authors) and experimental data published by Dey et al. (2006). One can see that the stress-strain curves as predicted by the TLMPM is in very good agreement with the FEM. Also, as expected, after damage initiation a steady decline of stress is observed, similarly to what happens in the finite element results, but with greater stability. Finally, the numerical simulations are in reasonable agreement with the experimental data. For all that, they do not quantitatively predict the strain at failure. This was expected since all the materials parameters are directly taken from the literature and no calibration was carried out to obtain a better match.

Figure 8.58 shows the typical evolution of both the equivalent stress and damage variable inside a tensile sample. It shows the formation and evolution of necking



(a) W460E (b) W700E (c) W900E

Fig. 8.57 Comparison of the stress-strain curves obtained with the TLMPM, FEM (de Vaucorbeil et al. 2020) and experimentally by Dey et al. (2006)

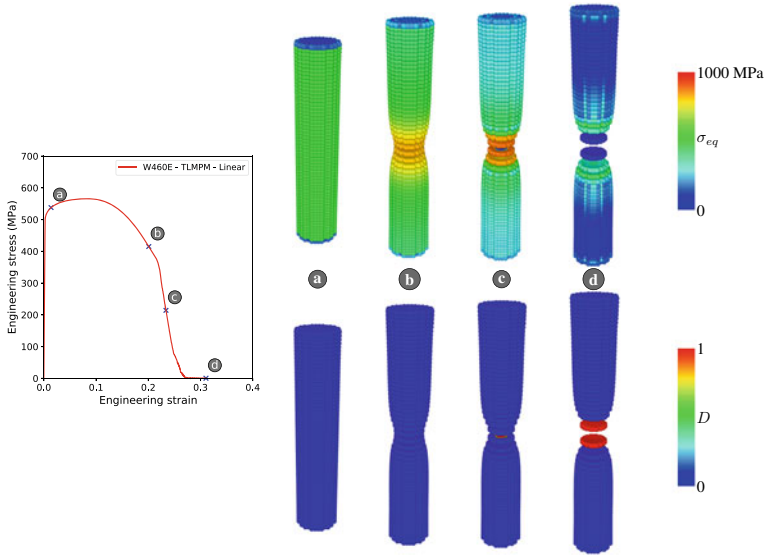


Fig. 8.58 Details of the evolution of the equivalent von Mises stress (σ_{eq}) and the damage (D) distribution during the tensile test of a W460E smooth cylinder using linear shape functions (de Vaucorbeil et al. 2022b). Visualization done with `Ovito` (Stukowski 2009)

in the specimen, as well as how the damage initiates and propagates inside the sample. These results prove that the simulation of ductile materials all the way to failure is possible using the TLMPM. Note that one should be careful when using the ULMPM for modeling fracture as numerical fracture is inherent in any particle methods adopting an UL formulation (Homel et al. 2016).

8.6.2 *Double Circular Notched Specimen*

This example consists of a a square specimen with asymmetrically placed two large circular notches loaded in tension (Fig. 8.59). More specifically, the left and upper edges of the specimen are pulled vertically upward while the lower and right edges are restrained. This is a problem without sharp discontinuity. This example is interesting owing to the curved nature of the obtained crack pattern joining the two notches in diagonal. The material is W700E. The hydrostatic pressure follows a linear EOS, see Eq. (4.11).

As expected, the crack has a curved trajectory and initiates around the notches (Fig. 8.60). However, the initiation does not happen exactly at the notch edge. This phenomenon is an artefact of the Johnson-Cook damage criterion and is not related to the enhanced gradient formulation. Indeed, the stress-triaxiality level is large at the centre of the specimen (Fig. 8.61a). Therefore, the failure strain ϵ_f given by Eq. (4.16) is higher at the centre. But, the plastic equivalent strain is higher near the edges (Fig. 8.61b). Owing to the damage initiation increment (Eq. (4.16)) being the ratio between the equivalent plastic strain increment and the stress triaxiality level, there is a competition between these two quantities along the expected crack path of this

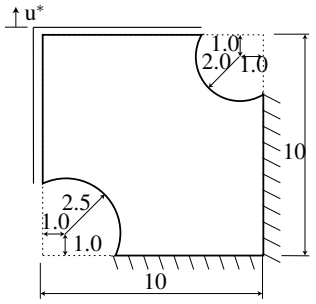
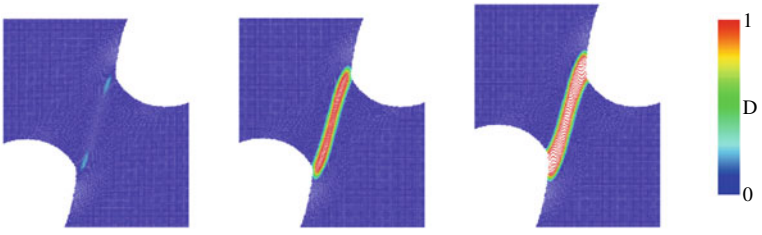
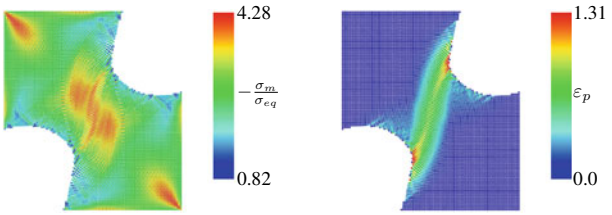


Fig. 8.59 Double circular notched specimen: geometry, loading and boundary conditions. The material is W700E (de Vaucorbeil et al. 2022b)



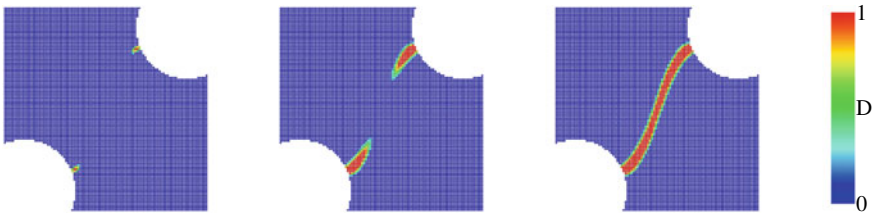
(a) $\varepsilon_p = 1.314$ (b) $\varepsilon_p = 1.968$ (c) $\varepsilon_p = 2.176$

Fig. 8.60 Double circular notched specimen: evolution of damage (on the deformed configuration). The results are shown for $l_d = 2.5$ mm with $l_0 = 0.10$ mm (de Vaucorbeil et al. 2022b)



(a) Triaxiality (b) ε_p

Fig. 8.61 Double circular notched specimen: hydrostatic pressure σ_m and plastic strain ε_p at damage initiation (Fig. 8.60a). The results are shown for $l_d = 2.5$ mm with $l_0 = 0.10$ mm (de Vaucorbeil et al. 2022b)



(a) $\varepsilon_p = 1.069$ (b) $\varepsilon_p = 1.583$ (c) $\varepsilon_p = 1.584$

Fig. 8.62 Double circular notched specimen: evolution of damage ignoring the effect of triaxiality in the computation of the equivalent strain at failure (i.e., $D_2 = 0$ in Eq. (4.16)). Results shown in the reference/initial configuration. The results are shown for $l_d = 2.5$ mm with $l_0 = 0.10$ mm (de Vaucorbeil et al. 2022b)

specimen. This results in damage initiating somewhere in between the centre and notch-edge (Fig. 8.60a).

This explanation can be verified by removing the effect of triaxiality from the failure strain ϵ_f in Eq. (4.16) such that it is constant. This is done by setting $D_2 = 0.0$. This results in a crack that do initiate at the edge of the notch as one can see in Fig. 8.62. The path itself remains qualitatively unchanged.

For the stress-strain response (with $D_2 = 4.768$) shown in Fig. 8.63a, one can see that no residual stress persists once the specimen has fully cracked. Moreover, the response is nearly identical for all the tested cell sizes, i.e., $l_0 = \{1.00\text{ mm},\ 0.50\text{ mm},\ 0.25\text{ mm}\}$. The spatial convergence is further confirmed from the value of engineering strains at damage initiation and final failure (Fig. 8.63b).

8.6.3 Compact Tension Specimen

This example concerns a standard mode-I ductile fracture problem where a notched specimen is pulled from two circular hinges located at the top and bottom, respectively, as shown in Fig. 8.64. This quasi-static fracture problem is popuplar as it is

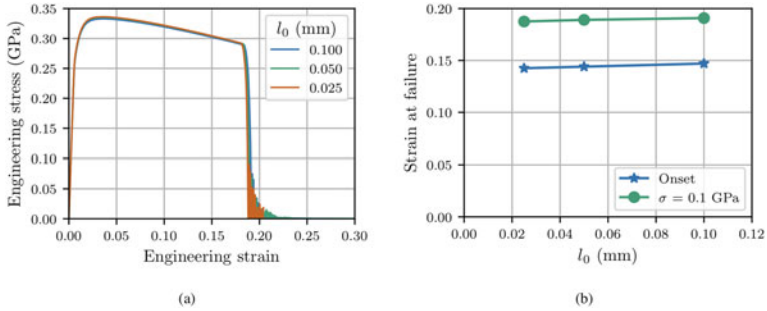


Fig. 8.63 Double circular notched specimen: load-deformation response. Linear weighting function with single particle per cell is considered. The nonlocal length scale $l_d = 2.5$ mm (de Vaucorbeil et al. 2022b)

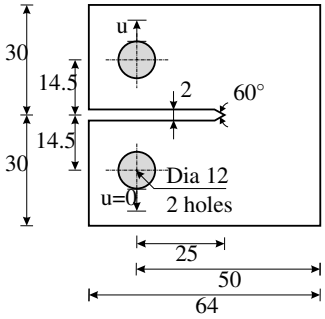


Fig. 8.64 Schematic of the CT specimen. The shaded areas correspond to rigid hinges being inserted in the holes. The hinges are explicit modeled and a hard contact between them and the specimen is adopted (de Vaucorbeil et al. 2022b)

common mechanical test. The material is W700E. The hydrostatic pressure follows a linear EOS.

Following the contact procedures developed in Sect. 8.2, a hard contact between the rigid hinges and holes has been implemented. The same vertical velocity as before is applied at the center of each hinge (i.e., $v(t) = \pm 2(1.0 - e^{-t})$ m/s). The force is transferred from the hinges to the specimen via unilateral contacts. The opposite displacement of the hinges generates a stress concentration at the notch tip, which results in localization of plastic deformation. The accumulated plastic strain causes damage initiation and propagation following Eq. (8.67). The propagation is horizontal in this case, as expected (Fig. 8.65).

The load-deformation response is shown in Fig. 8.66. The resistance of the specimen increases with applied strain until damage initiates. Then, the resistance of the specimen (force) plummets as damage propagates. It is worth noting that the load-deformation response converges with spatial refinement. Indeed, responses obtained with $l_0 \leq 0.25$ mm are virtually identical.

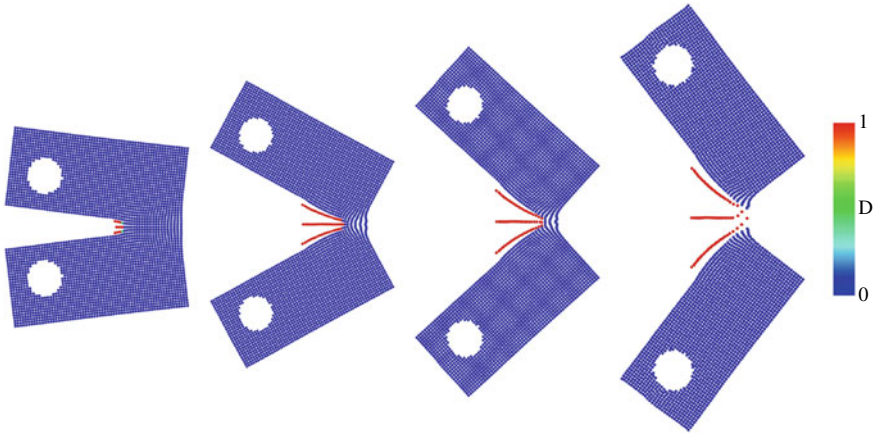


Fig. 8.65 Compact tension specimen: evolution of damage. The results are shown for $l_d = 2.5$ mm with $l_0 = 0.50$ mm. Linear weighting function with single particle per cell is considered (de Vaucorbeil et al. 2022b)

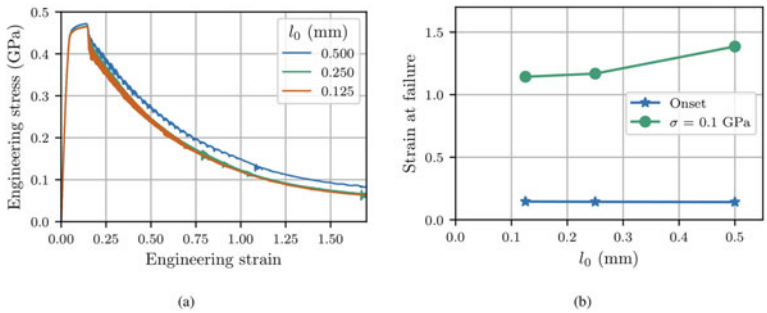


Fig. 8.66 Compact tension specimen: load-deformation response. Linear weighting function with single particle per cell is considered. The nonlocal length scale $l_d = 2.5$ mm (de Vaucorbeil et al. 2022b)

8.6.4 Machining Simulations

Machining is a controlled material removal process in which a piece of material is cut into a desired final shape and size. It is one of the most prominent industrial applications in the manufacturing field. Numerical simulations play an important role in improving this crucial process. During the material cutting process, severe deformations occur. Thus, accurate simulations of the process with mesh-based methods (i.e. finite element methods) is complicated owing to mesh distortion problems. Therefore, particle based methods like the MPM seem more appropriate for this kind of simulations.

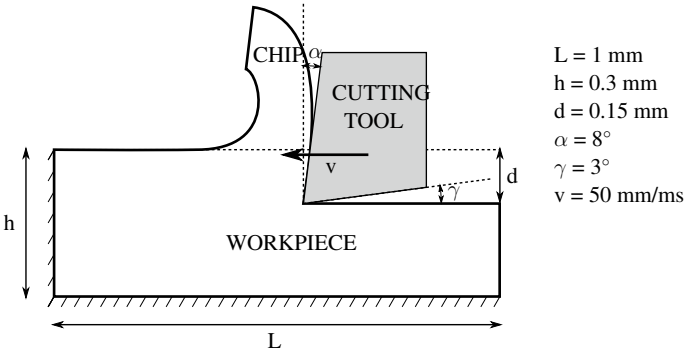


Fig. 8.67 Schematic of the 2D machining simulation setup

Table 8.7 Material parameters for the high strength steel used in the machining simulation

Material parameters		Flow stress params		EOS params		Damage params	
Density	7750 kg/m^3	A	980 MPa	c_0	5166 m/s	D_1	0.05
Young's modulus	211 GPa	B	2000 MPa	S_α	1.5	D_2	0.8
Poisson's ratio	0.33	C	0	Γ_0	2.17	D_3	–0.44
		n	0.83			D_4	0
						D_5	0

The total Lagrangian MPM is here used to simulate the cutting process of a workpiece which is a slab of 1 mm in length, and 0.3 mm in height (Fig. 8.67). The material of the workpiece is a high strength steel of which elasto-plastic and damage parameters, taken from Banerjee et al. (2015), are given in Table 8.7. The cutting depth and speeds are 0.15 mm and 50 mm/ms, respectively.

To save computational time, 2D simulations are considered and only the workpiece is discretized. The cutting tool is not modeled by particles, but rather by a force derived from Hertz' contact theory:

$$F_{tool} = \frac{\pi}{2} G \left(\frac{1+\nu}{1-\nu^2} \right) p^{3/2} \tag{8.81}$$

where G is the substrate's shear modulus, ν its Poisson's ration, and p the penetration distance between a particle and the cutting tool. The contact between the tool and the workpiece is supposed to be frictionless, therefore the direction of $\mathbf{F}_{tool}$ is normal to the tool's cutting surface.

The use of TLMPM is here motivated by the necessity to have material separation that is independent of the background grid size, i.e. described only by physic-based damage equations. This is because the ULMPM usually suffers from numerical frac-

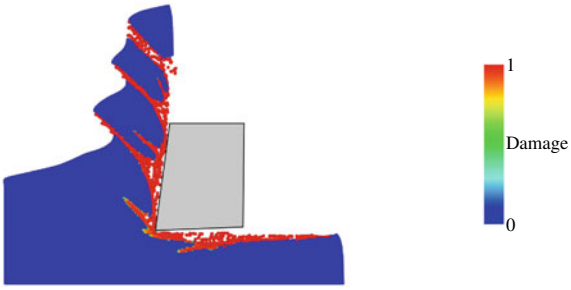


Fig. 8.68 Cutting process of a high strength steel (2D model)

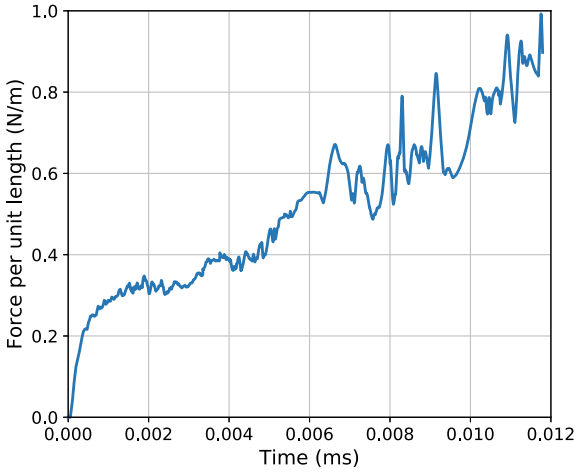


Fig. 8.69 Cutting force evolution profile

ture when particles are separated sufficiently far from each others. The results of these simulations showcases the abilities of the TLMPM implementation in `Karamelo` to simulate the formation and detachment of chips from the workpiece (See Figs. 8.68 and 8.69).

8.6.5 High Velocity Impact of a Bullet Into a Steel Plate

The simulation of impacts such as ballistic penetration is a great application for the TLMPM (explicit) as it involves large deformation, high strain rates and failure. Dey et al. (2006) have performed experiments of blunt projectile impacts onto Weldox steel plates over a large range of velocities. They also have simulated these impacts with FEM and found that when decreasing the mesh size the ballistic limit predicted decreased without ever reaching convergence.

Although Dey et al. (2006) have studied the response of three different alloys (Weldox 400E, 700E, and 900E), only one, the Weldox 700E, is studied here. This

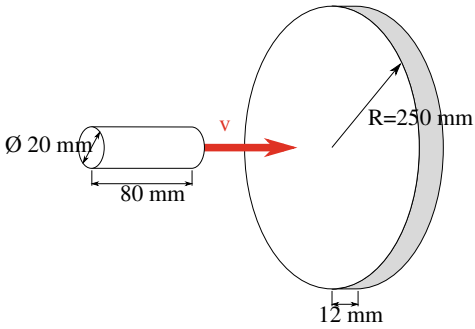


Fig. 8.70 Impact setup in 3D: a blunt projectile (cylinder) in launched at a cylindrical plate. The grey shade indicates fixed boundary conditions (de Vaucorbeil et al. 2022b)

is motivated by the fact that problems and challenges are the same for these three materials. The hydrostatic pressure follows a linear EOS.

The projectile is a steel cylinder 20 mm in diameter, 80 mm in length launched at a speed v ranging from 150 to 400 m/s towards a 12 mm thick plate which radius is 250 mm (see Fig. 8.70). The periphery of the plate is clamped. The problem being axisymmetrical, 2D axisymmetric simulations are performed here (see 2.7.2 for the axisymmetric TLMPM). The contact between the projectile and the plate is handled via the contact algorithm previously described in Sect. 8.2.

The projectile is modeled as a linearly elastic material with a modulus of 204 GPa, Poisson's ratio of 0.33, and density 7850 kg/m^3 (Dey et al. 2006). At first, the nonlocal radius is taken here as $l_d = 1.0$ mm. But its influence will be discussed later. The projectile and the target are discretized with 1 and 2×2 particle per cell, respectively.

Experimentally, the quantities of interest are the initial and residual velocities of the projectile. The residual velocity is the velocity of the projectile after penetration. If the projectile does not penetrate the target, this velocity is taken as zero. In order to compare the simulation and experimental results, the velocity of projectile is taken as the average velocity of all its particles.

Particle erosion. For an initial velocity of 250 m/s and a cell size of $l_0 = 0.5$ mm, the obtained velocity profile of the projectile is shown in Fig. 8.71. In this discussion time starts (i.e., $t = 0$) when the projectile first touches the target. Starting with an initial velocity of 250 m/s, one observes a rapid drop in the projectile's velocity owing to the absorption of a part of its kinetic energy by the target. At $t \approx 15$ μs (see Fig. 8.72a), the first signs of damage appear in the target, and by $t = 50$ μs, the target is fully damage and is thus perforated (see Fig. 8.72b). However, the velocity of the projectile keeps decreasing. This decrease happens step by step. These steps correspond to a contact between the projectile and the plug as seen in Fig. 8.72c. After each one of these contacts, the plug velocity increases significantly and becomes higher than that of the projectile. Then a gap forms, as seen in Fig. 8.72d. Since no external forces are considered, the plug's velocity should not decrease after having pulled away from the projectile. Thus, the projectile should not enter in contact with it after it has separated from the target. The hypothesis is that there is still a transmission of forces between

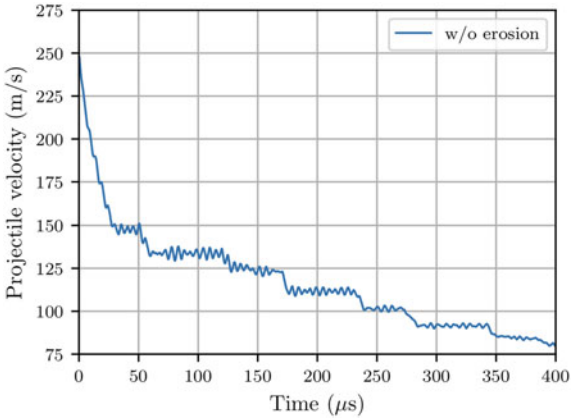


Fig. 8.71 Time evolution of the projectile velocity obtained without particle erosion. The initial velocity is 250 m/s. The cell size is $l_0 = 0.5$ mm (de Vaucorbeil et al. 2022b)

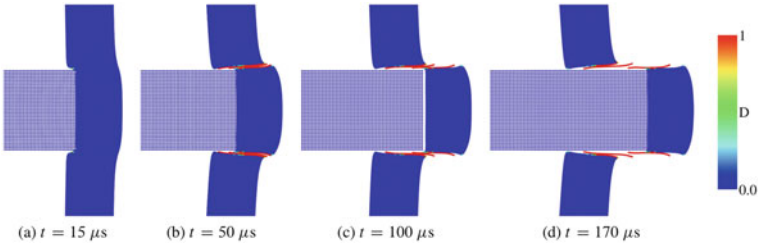


Fig. 8.72 Time evolution of impact between the projectile and the target for an initial velocity of 250 m/s (w/o particle erosion). The colors show the amount of damage present in each particles. For ease of visualisation, only a small part of the simulated domain focused around the impact area is shown (de Vaucorbeil et al. 2022b)

the target and the plug via the damaged particles. This can be due to the velocity of the damage particles still being taken into account in the computation of the nodes' velocity during the P2G step (Algorithm 4).

To test this hypothesis, fully damaged particles will not be taken into account during the mapping from particles to grid (P2G). Thus, the nodal masses, velocities, and forces will not be function of that of the fully damaged particles. And this particle erosion technique is similar to element deletion in FEM. The same simulation as presented above is then repeated (initial velocity of 250 m/s and a cell size of $l_0 = 0.5$ mm). The obtained projectile velocity profile (Fig. 8.73) shows that after $t \approx 60$ µs, the velocity is constant and does not go down step by step as in the case without particle erosion (Fig. 8.71). Moreover, from the time evolution of the impact (Fig. 8.74), one can see that compared to what happened without particle erosion, the plug separates more from both the target and the projectile.

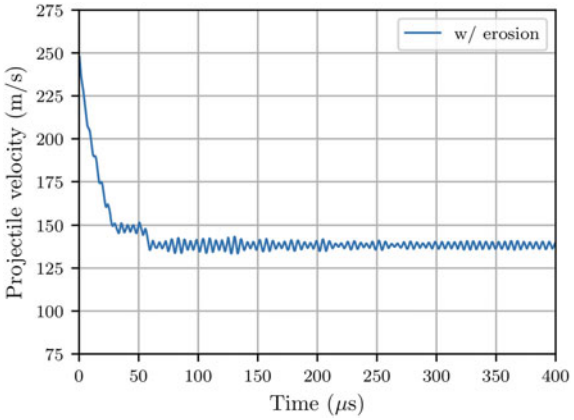


Fig. 8.73 Time evolution of the projectile velocity obtained with particle erosion. The initial velocity is 250 m/s. The cell size is $l_0 = 0.5$ mm (de Vaucorbeil et al. 2022b)

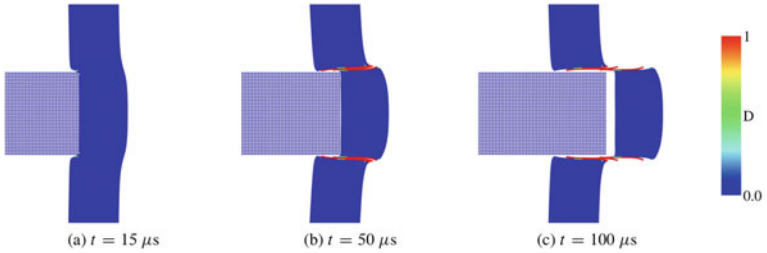


Fig. 8.74 Time evolution of impact between the projectile and the target for an initial velocity of 250 m/s (with particle erosion). The colors show the amount of damage present in each particles. For ease of visualisation, only a small part of the simulated domain focused around the impact area is shown (de Vaucorbeil et al. 2022b)

Since particle erosion solves the issue of the projectile velocity continuously dropping after penetration of the target, it will be used in subsequent simulations.

Validation against experiments. Before being able to simulate the residual velocities for different initial velocities, one has to ensure the convergence of the results. The convergence has been studied by decreasing the background cell size from 1.0 to 0.125 mm keeping the initial velocity constant at 250 m/s, with particle erosion activated. Figure 8.75 shows that the residual velocity converges as it is constant for cell sizes lower than 0.125 mm.

The validation of the nonlocal TLMPM method proposed here has been done by simulating the residual velocity as a function of the initial projectile velocity. The cell size is taken as $l_0 = 0.125$ mm. And the nonlocal radius is $l_d = 1.0$ mm. The

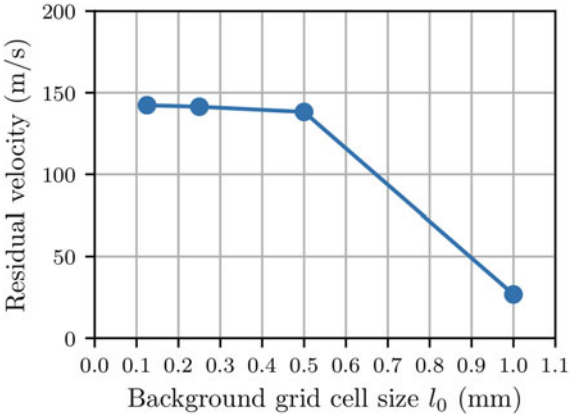


Fig. 8.75 Convergence of the residual velocity with respect to the cell sizes (de Vaucorbeil et al. 2022b)

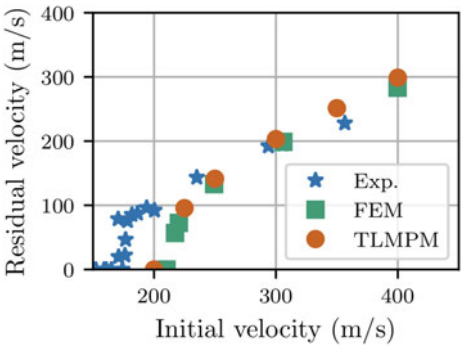


Fig. 8.76 Residual velocity versus initial velocity (de Vaucorbeil et al. 2022b)

obtained results are plotted in Fig. 8.76 alongside both the experimental and the FEM results published by Dey et al. (2006). One can see that very good agreement has been reached with the FEM and that good agreement is obtained with the experiments for velocities greater than 225 m/s.

References

Banerjee, A., Dhar, S., Acharyya, S., Datta, D., Nayak, N.: Determination of johnson cook material and failure model constants and numerical modelling of charpy impact test of armour steel. Mater. Sci. Eng., A **640**, 200–209 (2015). https://doi.org/10.1016/j.msea.2015.05.073. Jul

Bardenhagen, S.G., Kober, E.M.: The generalized interpolation material point method. Comput. Model. Eng. Sci. **5**(6), 477–495 (2004)

Bardenhagen, S.G., Brackbill, J.U., Sulsky, D.: The material-point method for granular materials. Comput. Methods Appl. Mech. Eng. **187**(3–4), 529–541 (2000)

Bardenhagen, S.G., Guilkey, J.E., Roessig, K.M., Brackbill, J.U., Witzel, W.M., Foster, J.C.: An improved contact algorithm for the material point method and application to stress propagation in granular material. Comput. Model. Eng. Sci. **2**(4), 509–522 (2001)

Belytschko, Ted, Neal, Mark O.: Contact-impact by the pinball algorithm with penalty and lagrangian methods. Int. J. Numer. Meth. Eng. **31**(3), 547–572 (1991). https://doi.org/10.1002/nme.1620310309

Bourdin, B., Francfort, G.A., Marigo, J.J.: Numerical experiments in revisited brittle fracture. J. Mech. Phys. Solids **48**(4), 797–826 (2000)

Budyn, E., Zi, G., Moës, N., Belytschko, T.: A method for multiple crack growth in brittle materials without remeshing. Int. J. Numer. Meth. Eng. **61**(10), 1741–1770 (2004)

Campbell, J., Vignjevic, R., Libersky, L.: A contact algorithm for smoothed particle hydrodynamics. Comput. Methods Appl. Mech. Eng. **184**(1), 49–65 (2000). https://doi.org/10.1016/s0045-7825(99)00442-9

Cheon, Y.-J., Kim, H.-G.: An adaptive material point method coupled with a phase-field fracture model for brittle materials. Int. J. Numer, Methods Eng (2019)

Coetzee, C.J.: The modelling of granular flow using the particle-in-cell method. Ph.D. thesis, University of Stellenbosch (2003)

de Vaucorbeil, A., Nguyen, C.P., Sinaie, S., Wu, J.Y.: Chapter Two - Material Point Method After 25 Years: Theory, Implementation, and Applications. Advances in Applied Mechanics, vol. 53, pp. 185–398. Elsevier (2020)

de Vaucorbeil, A., Hutchinson, C.R.: A new total-Lagrangian smooth particle hydrodynamics approximation for the simulation of damage and fracture of ductile materials. Int. J. Numer. Methods Eng. **121**, 2227–2245 (2020)

de Vaucorbeil, A., Nguyen, C.P.: Modeling contacts with a total lagrangian matertial point method. Comput. Methods Appl. Mech. Eng. **360**, 112783 (2021). https://doi.org/10.1016/j.cma.2019.112783

de Vaucorbeil, A., Nguyen, V.P.: Karamelo: an open source parallel C++ package for the matertial point method. Comput. Particle Mech. **8**, 767–789 (2021)

de Vaucorbeil, A., Nguyen, V.P., Hutchinson, C.R.: A Total-Lagrangian material point method for solid mechanics problems involving large deformations. Comput. Methods Appl. Mech. Eng. **360**, 112783 (2020). https://doi.org/10.1016/j.cma.2019.112783

de Vaucorbeil, A., Nguyen, V.P., Hutchinson, C.R., Barnett, M.R.: Total lagrangian material point method simulation of the scratching of high purity coppers. Int. J. Solids Struct. **239–240**, 111432 (2022)

de Vaucorbeil, A., Nguyen, V.P., Mandal, T.K.: Mesh objective simulations of large strain ductile fracture: a new nonlocal johnson-cook damage formulation for the total lagrangian material point method. Comput. Methods Appl. Mech. Eng. **389**, 114388 (2022)

Dey, S., Børvik, T., Hopperstad, O.S., Leinum, J.R., Langseth, M.: The effect of target strength on the perforation of steel plates using three different projectile nose shapes. Int. J. Impact Eng **30**(8), 1005–1038 (2004)

Dey, S., Børvik, T., Hopperstad, O.S., Langseth, M.: On the influence of fracture criterion in projectile impact of steel plates. Comput. Mater. Sci. **38**(1), 176–191 (2006)

Elices, M.G.G.V., Guinea, G.V., Gomez, J., Planas, J.: The cohesive zone model: advantages, limitations and challenges. Eng. Fract. Mech. **69**(2), 137–163 (2002)

Francfort, G.A., Marigo, J.J.: Revisiting brittle fracture as an energy minimization problem. J. Mech. Phys. Solids **46**(8), 1319–1342 (1998)

Geuzaine, C., Remacle, J.F.: Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. Int. J. Numer. Meth. Eng. **79**(11), 1309–1331 (2009)

Gilabert, F.A., Cantavella, V., Sánchez, E., Mallol, G.: Modelling fracture process in ceramic materials using the material point method. EPL (Europhys. Lett.) **96**(2), 24002 (2011)

Gray, J.P., Monaghan, J.J., Swift, R.P.: Sph elastic dynamics. Comput. Methods Appl. Mech. Eng. **190**(49–50), 6641–6662 (2001)

Griffith, A.A.: The phenomena of rupture and flow in solids. Philos. Trans. R. Soc. Londres **221**, 163–198 (1920)

Guo, Y., Nairn, J.A.: Calculation of j-integral and stress intensity factors using the material point method. Comput. Model. Eng. Sci. **6**, 295–308 (2004)

Homel, M.A., Herbold, E.B.: Field-gradient partitioning for fracture and frictional contact in the material point method. Int. J. Numer. Meth. Eng. **109**(7), 1013–1044 (2017)
Homel, M.A., Brannon, R.M., Guilkey, J.: Controlling the onset of numerical fracture in parallelized implementations of the material point method (MPM) with convective particle domain interpolation (CPDI) domain scaling. Int. J. Numer. Meth. Eng. **107**(1), 31–48 (2016)
Huang, P., Zhang, X., Ma, S., Huang, X.: Contact algorithms for the material point method in impact and penetration simulation. Int. J. Numer. Meth. Eng. **85**(4), 498–517 (2011)
Inglis, C.E.: Stresses in plates due to the presence of cracks and sharp corners. Trans. Inst. Naval Arch. **55**, 219–241 (1913)
Kakouris, E.G., Triantafyllou, S.P.: Phase-field material point method for brittle fracture. 2020. Int. J. Numer. Methods Eng. **112**(12), 1750–1776 (2017b)
Kakouris, E.G., Triantafyllou, S.P.: Material point method for crack propagation in anisotropic media: a phase field approach. Arch. Appl, Mech (2017)
Kuhn, C., Müller, R.: A continuum phase field model for fracture. Eng. Fract. Mech. **77**(18), 3625–3634 (2010)
Lemiale, V., Nairn, J., Hurmane, A.: Material point method simulation of equal channel angular pressing involving large plastic strain and contact through sharp corners. Comput. Model. Eng. Sci. **70**(1), 41–66 (2010)
Leroch, S., Varga, M., Eder, S.J., Vernes, A., Rodriguez Ripoll, M., Ganzenmüller, G.: Smooth particle hydrodynamics simulation of damage induced by a spherical indenter scratching a viscoplastic material. Int. J. Solids Struct. **81**(Supplement C), 188–202 (2016)
Li, B., Kidane, A., Ravichandran, G., Ortiz, M.: Verification and validation of the optimal transportation meshfree (OTM) simulation of terminal ballistics. Int. J. Impact Eng **42**, 25–36 (2012)
Li, X., Sovilla, B., Jiang, C., Gaume, J.: Three-dimensional and real-scale modeling of flow regimes in dense snow avalanches. Landslides **18**(10), 3393–3406 (2021)
Liang, Y., Benedek, T., Zhang, X., Liu, Y.: Material point method with enriched shape function for crack problems. Comput. Methods Appl. Mech. Eng. **322**, 541–562 (2017)
Lin, L., Blackman, G.S., Matheson, R.R.: Quantitative characterization of scratch and mar behavior of polymer coatings. Mater. Sci. Eng., A **317**(1–2), 163–170 (2001). https://doi.org/10.1016/s0921-5093(01)01159-5
Mandal, T.K., Nguyen, V.P., Wu, J.-Y.: Length scale and mesh bias sensitivity of phase-field models for brittle and cohesive fracture. Eng. Fract. Mech. **217**(106532) (2019b)
Mandal, T.K., Nguyen, V.P., Heidarpour, A.: Phase field and gradient enhanced damage models for quasi-brittle failure: a numerical comparative study. Eng. Fract. Mech., 207(48–67), 2019a
Mandal, T.K., Nguyen, V.P., Wu, J.-Y.: A length scale insensitive anisotropic phase field fracture model for hyperelastic composites. Int. J. Mech. Sci. **188**, 105941 (2020)
Moës, N., Dolbow, J., Belytschko, T.: A finite element method for crack growth without remeshing. Int. J. Numer. Meth. Eng. **46**(1), 133–150 (1999)
Müller, M., Chentanez, N., Kim, T.-Y., Macklin, M.: Air meshes for robust collision handling. ACM Trans. Graph. (TOG) **34**(4), 133 (2015)
Nairn, J.A.: Material point method calculations with explicit cracks. Comput. Model. Eng. Sci. **4**(6), 649–663 (2003)
Nairn, J.A.: Material point method simulations of transverse fracture in wood with realistic morphologies. Holzforschung **61**(4), 375–381 (2007)
Nairn, J.A.: Numerical implementation of imperfect interfaces. Comput. Mater. Sci. **40**(4), 525–536 (2007)
Nairn, J.A., Bardenhagen, S.G., Smith, G.D.: Generalized contact and improved frictional heating in the material point method. Comput Particle Mech. **5**(3), 285–296 (2018)
Nguyen, C.T., Nguyen, V.P., de Vaucorbeil, A., Mandal, T.K., WU, J.Y.: Jive: An open source, research-oriented C++ library for solving partial differential equations. Adv. Eng. Softw. **150**(102925) (2020)

Nguyen, V.P., de Vaucorbeil, A., Nguyen-Thanh, C., Mandal, T.K.: A generalized particle in cell method for explicit solid dynamics. Comput. Methods Appl. Mech. Eng. **360**, 112783 (2021). https://doi.org/10.1016/j.cma.2019.112783

Oliver, J., Hartmann, S., Cante, J.C., Weyler, R., Hernández, J.A.: A contact domain method for large deformation frictional contact problems. part 1: Theoretical basis. Comput. Methods Appl. Mech. Eng. **198**(33–36), 2591–2606 (2009)

Pandolfi, A., Li, B., Ortiz, M.: Modeling Fracture by Material-Point Erosion, pp. 3–16. Cham (2014)

Pandolfi, A., Ortiz, M.: An eigenerosion approach to brittle fracture. Int. J. Numer. Meth. Eng. **92**(8), 694–714 (2012)

Peerlings, R.H.J., de Borst, R., Brekelmans, W.A.M., Geers, M.G.D.: Localisation issures in local and nonlocal continuum approaches to fracture. European J. Mech. A/Solids **21**, 7207–7229 (2002)

Randles, P.W., Libersky, L.D.: Smoothed particle hydrodynamics: some recent improvements and applications. Comput. Methods Appl. Mech. Eng. **139**(1–4), 375–408 (1996). ISSN 0045-7825

Rice, J.R.: A path independent integral and the approximate analysis of strain cncentrations by notches and cracks. J. Appl. Mech.-T. ASME **35**, 379–386 (1968)

Schmidt, B., Fraternali, F., Ortiz, M.: Eigenfracture: an eigendeformation approach to variational fracture. Multiscale Model Simul. **7**(3), 1237–1266 (2009)

Steffen, M., Kirby, R.M., Berzins, M.: Analysis and reduction of quadrature errors in the material point method (MPM). Int. J. Numer. Meth. Eng. **76**(6), 922–948 (2008)

Stukowski, A.: Visualization and analysis of atomistic simulation data with ovito-the open visualization tool. Modell. Simul. Mater. Sci. Eng. **18**(1), 015012 (2009)

Sulsky, D., Brackbill, J.U.: A numerical method for suspension flow. J. Comput. Phys. **96**(2), 339–368 (1991)

Sulsky, D., Schreyer, H.L.: Axisymmetric form of the material point method with applications to upsetting and Taylor impact problems. Comput. Methods Appl. Mech. Eng. **139**, 409–429 (1996)

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

Sutula, D., Kerfriden, P., van Dam, T., Bordas, S.P.A.: Minimum energy multiple crack propagation. In: XFEM Computer Implementation and Applications. Engineering Fracture Mechanics, Part III (2017)

Tan, H., Nairn, J.A.: Hierarchical, adaptive, material point method for dynamic energy release rate calculations. Comput. Methods Appl. Mech. Eng. **191**(19–20), 2123–2137 (2002)

Trucano, T.G., Grady, D.E.: Study of intermediate velocity penetration of steel spheres into deep aluminum targets. Technical report, Sandia National Labs., Albuquerque, NM (USA) (1985)

Villumsen, M.F., Fauerholdt, T.G.: Simulation of Metal Cutting Using Smooth Particle Hydrodynamics. LS-DYNA Anwenderforum, Bamberg, vol. 30 (2008). http://refhub.elsevier.com/S0020-7683(15)00487-4/sbref0045

Wang, H., Wereszczak, A.A., Lance, M.J.: Effect of grain size on dynamic scratch response in alumina. In: Mechanical Properties and Performance of Engineering Ceramics II: Ceramic Engineering and Science Proceedings, vol. 27, Issue 2, pp. 767–779. Wiley, Inc. (2006) 10.1002/9780470291313.ch72

Wang, Jian: Chan, Dave: Frictional contact algorithms in SPH for the simulation of soil-structure interaction. Int. J. Numer. Anal. Meth. Geomech. **38**(7), 747–770 (2014). https://doi.org/10.1002/nag.2233

Wang, B., Karuppiah, V., Lu, H., Komanduri, R., Roy, S.: Two-dimensional mixed mode crack simulation using the material point method. Mech. Adv. Mater. Struct. **12**(6), 471–484 (2005)

Wolper, J., Fang, Y., Li, M., Lu, J., Gao, M., Jiang, C.: Chenfanfu: Cd-mpm: continuum damage material point methods for dynamic fracture animation. ACM Trans. Graph. (TOG) **38**(4), 119 (2019)

Wriggers, P.: Computational Contact Mechanics, 2nd edn. Springer (2006)

Wriggers, P., Schröder, J., Schwarz, A.: A finite element method for contact using a third medium. Comput. Mech. **52**(4), 837–847 (2013)

Wu, J.Y., Nguyen, V.P., Nguyen, C.T., Sutula, D., Sinaie, S., Bordas, S.: Phase field modeling of fracture. In: Advances in Applied Mechancis: Fracture Mechanics: Recent Developments and Trends, vol. 53:submitted (2019)

Wu, J.Y.: A unified phase-field theory for the mechanics of damage and quasi-brittle failure in solids. J. Mech. Phys. Solids **103**, 72–99 (2017)

Wu, J.Y., Nguyen, V.P.: A length scale insensitive phase-field damage model for brittle fracture. J. Mech. Phys. Solids **119**, 20–42 (2018)

Wu, J.-Y., Huang, Y., Zhou, H., Nguyen, V.P.: Three-dimensional phase-field modeling of mode i + ii/iii failure in solids. Comput. Methods Appl. Mech. Eng. **373**, 113537 (2021)

York, A.R.: Development of modifications to the material point method for the simulation of thin membranes, compressible fluids, and their interactions. Ph.D. thesis, The University of New Mexico, Albuquerque (1997)

Chapter 9
Stability, Accuracy and Recent Improvements

Even though the MPM, as it has been introduced in this book up to this point, has been used with great success in solving many challenging engineering problems (see Chap. 1 for applications of the MPM), its convergence rate of (precisely that of the updated Lagrangian variants) are poor in the case of simple academic problems. The issues are: (1) the MPM does not converge quadratically (the best rate we can hope for) and (2) for very fine grid resolutions, the method does not converge at all.

In order to develop better MPMs, a deeper understanding of the stability and accuracy of the method is necessary. This will be achieved in this chapter through a detailed mathematical analysis of the MPM.

The chapter begins with an analysis of energy and momenta conservation (Sect. 9.1). To quantify the convergence rate of the MPM, the method of manufactured solutions is often used. An introduction to the method of manufactured solutions is thus given in Sect. 9.2. Also discussed are error norms and how to compute the convergence rate. Next, we discuss the improved MPM (Sect. 9.3)–which is a recent development in the MPM community–in which the moving least square is used to improve the particle to grid mapping. Then, the Affine Particle in Cell (APIC)–a recent development from the computer graphics community (Sect. 9.4) is introduced. Section 9.5 presents various tests to study the convergence rate of different MPM variants. Finally, we discuss methods to mitigate volumetric locking in the MPM (Sect. 9.6).

Like most meshfree/particle methods, analysis of the method is not an easy task. This difficulty stems from the fact that there are more than one type of error in the MPM: interpolation error, temporal error, quadrature error, particle to grid mapping error etc. which are all inter-related. The lack of an analysis framework for the MPM, as found in FE methods, makes it difficult to explain unexpected numerical artifacts often seen during simulations. Only a few works were published on this difficult but important topic e.g. Wallstedt and Guilkey (2008); Tran et al. (2010); Steffen et al. (2008a, b, 2010); Gritton and Berzins (2017); Hammerquist and Nairn (2017); Berzins (2018).

V. P. Nguyen et al., *The Material Point Method*, Scientific Computation,
https://doi.org/10.1007/978-3-031-24070-6_9

9.1 Energy and Momenta Conservation

The stability and accuracy of the MPM is dictated by its ability, or lack of, conserving the energy and momentum. Here, we first show how the MPM conserves linear momentum by design (Sect. 9.1.1), but not the angular momentum (Sect. 9.1.2). Finally, light is shed onto the origins of energy dissipations (Sect. 9.1.3).

9.1.1 Linear Momentum Conservation

As the objective of the MPM algorithm is to mimic the conservation properties of the continuum, both the total linear and angula momenta should be conserved. As far as the linear momentum is concerned, the good news is that the algorithm does so by design.

At the beginning of the time step n, the total linear momentum on the particles is:

$$L^t = \sum_p m_p v_p^t \tag{9.1}$$

After the particle to grid step, the total linear momentum on the grid is given by:

$$\sum_I m_I^t \mathbf{v}_I^t = \sum_I \sum_p m_p \phi_I(\mathbf{x}_p^t) \mathbf{v}_p^t \tag{9.2}$$

$$= \sum_p m_p \mathbf{v}_p^t \sum_I \phi_I(\mathbf{x}_p^t) \tag{9.3}$$

Owing to the partition of unity, $\sum_I \phi_I(\mathbf{x}_p^t) = 1$, therefore:

$$\sum_I m_I^t \mathbf{v}_I^t = \sum_p m_p \mathbf{v}_p^t = L^t \tag{9.4}$$

This shows that the particle to grid step conserves exactly the total linear momentum. What about the grid to particle step? To analyze this last step, one needs to distinguish between the cases where the PIC velocity update is used, and where the FLIP update is used. In both cases, the total linear momentum at the end of the time step is:

$$L^{t+\Delta t} = \sum_p m_p v_p^{t+\Delta t} \tag{9.5}$$

For both types of velocity update, $L^{t+\Delta t}$ is expressed as the function of the variables on the grid at the end of the momenta update step:

1. **PIC**

$$\begin{aligned}\sum_p m_p \mathbf{v}_p^{t+\Delta t} &= \sum_p m_p \sum_I \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t} \\ &= \sum_I \mathbf{v}_I^{t+\Delta t} \sum_p \phi_I(\mathbf{x}_p^t) m_p \\ &= \sum_I m_I^t \mathbf{v}_I^{t+\Delta t}\end{aligned} \tag{9.6}$$

2. **FLIP**

$$\begin{aligned}\sum_p m_p \mathbf{v}_p^{t+\Delta t} &= \sum_p m_p \mathbf{v}_p^t + \sum_p m_p \sum_I \phi_I(\mathbf{x}_p^t)(\mathbf{v}_I^{t+\Delta t} - \mathbf{v}_I^t) \\ &= \sum_p m_p \mathbf{v}_p^t + \sum_I (\mathbf{v}_I^{t+\Delta t} - \mathbf{v}_I^t) \sum_p \phi_I(\mathbf{x}_p^t) m_p \\ &= \sum_p m_p \mathbf{v}_p^t + \sum_I m_I^t \mathbf{v}_I^{t+\Delta t} - \sum_I m_I^t \mathbf{v}_I^t \\ &= \sum_I m_I^t \mathbf{v}_I^{t+\Delta t}\end{aligned} \tag{9.7}$$

Therefore, whatever the velocity update formulation used (PIC, or FLIP, or linear combination of both), the grid to particule step also conserves exactly the total linear momentum. This proves that the total linear momentum is always conserved in the MPM. Note that the same is true for TLMPM, as one can easily see by changing $\phi_I(\mathbf{x}_p^t)$ to $\phi_I(\mathbf{X}_p)$.

9.1.2 Angular Momentum Conservation

The story about the angular momentum conservation in MPM is quite different from that of the linear momentum. Indeed, nothing in the design of the method enforces its conservation as we shall see.

At the beginning of the time step n, the total angular momentum with respect to the origin is:

$$\mathbf{J}_p^t = \sum_p \mathbf{x}_p^t \times m_p v_p^t \tag{9.8}$$

Similarly to Sect. 9.1.1, the total angular momentum of the grid nodes with respect to the origin after the particle-to-grid projection $\mathbf{J}_h^t$ is:

$$\mathbf{J}_h^t = \sum_I \mathbf{x}_I \times m_I^t \mathbf{v}_I^t = \sum_I \mathbf{x}_I \times \sum_p m_p \phi_I(\mathbf{x}_p^t)\mathbf{v}_p^t$$
$$= -\sum_p m_p \mathbf{v}_p^t \times \sum_I \phi_I(\mathbf{x}_p^t)\mathbf{x}_I \tag{9.9}$$

Noting that $\sum_I \phi_I(\mathbf{x}_p^t)\mathbf{x}_I = \mathbf{x}_p$, the previous equation yields:

$$\sum_I \mathbf{x}_I \times m_I^t \mathbf{v}_I^t = -\sum_p m_p \mathbf{v}_p^t \times \mathbf{x}_p = J_p^t \tag{9.10}$$

The particle to grid step therefore conserves exactly the total angular momentum. How about the grid to particle step? Once again, distinction is made for the treatment of this step between PIC and FLIP particle update.

1. **PIC**: Remembering that the particle velocities are updated as $\mathbf{v}_p^{t+\Delta t} = \sum_J \phi_J(\mathbf{x}_p^t)\mathbf{v}_J^{t+\Delta t}$, the total angular momentum at the end of the time step is:

$$\begin{aligned}
\mathbf{J}_p^{t+\Delta t} &= \sum_p \mathbf{x}_p^{t+\Delta t} \times m_p \mathbf{v}_p^{t+\Delta t} \\
&= \sum_p \mathbf{x}_p^{t+\Delta t} \times m_p \sum_J \phi_J(\mathbf{x}_p^t)\mathbf{v}_J^{t+\Delta t} \\
&= \sum_p \left(\sum_I \phi_I(\mathbf{x}_p^t)\mathbf{x}_I^{t+\Delta t} \right) \times m_p \sum_J \phi_J(\mathbf{x}_p^t)\mathbf{v}_J^{t+\Delta t} \\
&= \sum_I \mathbf{x}_I^{t+\Delta t} \times \left(\sum_J \sum_p m_p \phi_I(\mathbf{x}_p^t)\phi_J(\mathbf{x}_p^t)\mathbf{v}_J^{t+\Delta t} \right) \\
&= \sum_I \mathbf{x}_I^{t+\Delta t} \times \sum_J M_{IJ}\mathbf{v}_J^{t+\Delta t}
\end{aligned} \tag{9.11}$$

   where $M_{IJ} = \sum_p m_p \phi_I(\mathbf{x}_p^t)\phi_J(\mathbf{x}_p^t)$ is the full mass matrix. Owing to the difference between the full and lumped mass matrices, once can readily see that the total angular momentum is not conserved during this step.
   Equation (9.12) can be expended to reveal how the angular momentum evolves over time:

$$\begin{aligned}
\mathbf{J}_p^{t+\Delta t} &= \sum_I \mathbf{x}_I^{t+\Delta t} \times \sum_J \bar{M}_{IJ}\mathbf{v}_J^{t+\Delta t} - \sum_I \mathbf{x}_I^{t+\Delta t} \times \sum_J (\bar{M}_{IJ} - M_{IJ})\mathbf{v}_J^{t+\Delta t} \\
&= \mathbf{J}_h^t + \Delta \mathbf{J}_h - \sum_I \mathbf{x}_I^{t+\Delta t} \times \sum_J (\bar{M}_{IJ} - M_{IJ})\mathbf{v}_J^{t+\Delta t} \\
&= \mathbf{J}_p^t + \Delta \mathbf{J}_h - \sum_I \mathbf{x}_I^{t+\Delta t} \times \sum_J (\bar{M}_{IJ} - M_{IJ})\mathbf{v}_J^{t+\Delta t}
\end{aligned} \tag{9.12}$$

where $\Delta \mathbf{J}_h$ is the change of angular momentum during the grid update step, and $\bar{M}_{IJ}$ is the lumped mass matrix. In the absence of external forces, we have $\Delta \mathbf{J}_h = 0$ and the change of angular momentum in a given time step is $-\sum_I \mathbf{x}_I^{t+\Delta t} \times \sum_J (\bar{M}_{IJ} - M_{IJ}) \mathbf{v}_J^{t+\Delta t}$,

2. **FLIP**: With FLIP, the particles velocities are updated as: $\mathbf{v}_p^{t+\Delta t} = \mathbf{v}_p^t + \sum_J \phi_J(\mathbf{x}_p^t) \Delta \mathbf{v}_j$, where $= \Delta \mathbf{v}_j \mathbf{v}_J^{t+\Delta t} - \mathbf{v}_J^t$. Therefore, the total angular momentum at the end of the step is (Love and Sulsky 2006b):

$$
\begin{aligned}
\mathbf{J}_p^{t+\Delta t} &= \sum_p \mathbf{x}_p^{t+\Delta t} \times m_p \mathbf{v}_p^{t+\Delta t} \\
&= \sum_p \left(\sum_I \phi_I(\mathbf{x}_p^t) \mathbf{x}_I^{t+\Delta t} \right) \times m_p \left[\mathbf{v}_p^t + \sum_J \phi_J(\mathbf{x}_p^t) \Delta \mathbf{v}_j \right] \\
&= \sum_I \mathbf{x}_I^{t+\Delta t} \times \left[\sum_p \phi_I(\mathbf{x}_p^t) \left(m_p \mathbf{v}_p^t + m_p \sum_J \phi_J(\mathbf{x}_p^t) \Delta \mathbf{v}_j \right) \right] \\
&= \sum_I \mathbf{x}_I^{t+\Delta t} \times \left[\sum_J \bar{M}_{IJ} \mathbf{v}_J^t + \sum_J \sum_p \phi_I(\mathbf{x}_p^t) \phi_J(\mathbf{x}_p^t) m_p \Delta \mathbf{v}_j \right] \qquad (9.13) \\
&= \sum_I \mathbf{x}_I^{t+\Delta t} \times \left[\sum_J \bar{M}_{IJ} (\mathbf{v}_J^{t+\Delta t} - \Delta \mathbf{v}_j) + \sum_J \sum_p \phi_I(\mathbf{x}_p^t) \phi_J(\mathbf{x}_p^t) m_p \Delta \mathbf{v}_j \right] \\
&= \mathbf{J}_h^t + \Delta \mathbf{J}_h - \sum_I \mathbf{x}_I^{t+\Delta t} \times \sum_J (\bar{M}_{IJ} - M_{IJ}) \Delta \mathbf{v}_J \\
&= \mathbf{J}_p^t + \Delta \mathbf{J}_h - \sum_I \mathbf{x}_I^{t+\Delta t} \times \sum_J (\bar{M}_{IJ} - M_{IJ}) \Delta \mathbf{v}_J
\end{aligned}
$$

Similar to what happens when using the PIC velocity update, using the FLIP update, the total angular momentum is not conserved. Moreover, in the absence of external forces, $\Delta \mathbf{J}_h = \mathbf{0}$, and the change of angular momentum in a given time step is $-\sum_I \mathbf{x}_I^{t+\Delta t} \times \sum_{I,J} (\bar{M}_{IJ} - M_{IJ}) \Delta \mathbf{v}_I$.

The change of angular momentum on the particles is respectively proportional to $\mathbf{v}_J^{t+\Delta t}$ and $\Delta \mathbf{v}$ when using PIC and FLIP velocity update. It is therefore obvious that the magnitude of the change is lower with FLIP than PIC. In other words, FLIP preserves better the angular momentum than PIC.

9.1.3 Total Energy Conservation

In the MPM, not only energy conservation is not explicitly enforced but the energy is known to be dissipated. Dissipation comes from different origins. First, there is the use of a lumped mass matrix (Sect. 2.5.1). Then, the kind of formulation used, i.e., USF, USL, or MUSL. As shown in Figs. 6.9 and 6.10, USF, USL with or without

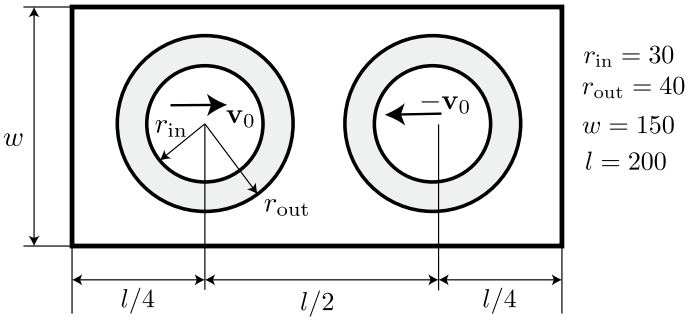
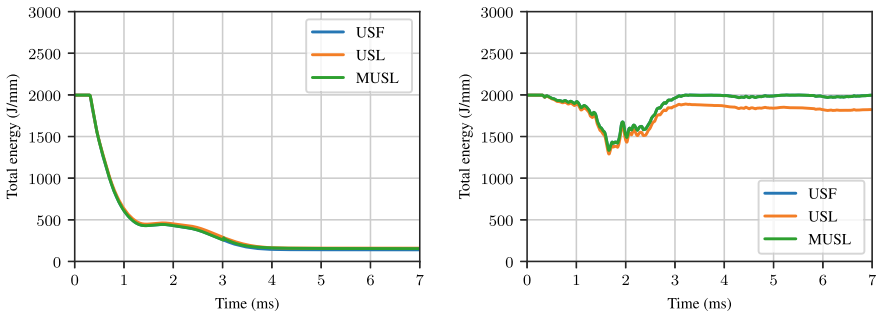


Fig. 9.1 Impact of two elastic bodies: problem description. Dimensions are in millimeters (de Vaucorbeil and Nguyen 2021)

double mapping do not have the same impact on the energy conservation. Finally, the total energy is dramatically affected by the use of either PIC or FLIP for the velocity update. In particular, PIC is known to be way more dissipative than FLIP.

To illustrate how the different formulations and velocity updates affect the total energy, the example of the impact between two compressible Neo-Hookean rings is used. The two rings are hollow elastic cylinders, under the assumption of plane strain (Fig. 9.1). The setup used here is the one used by Huang et al. (2011). The material is a compressible Neo-Hookean with bulk modulus $K = 121.7$ MPa, shear modulus $G = 26.1$ MPa and density $\rho = 1010 \times 10^{-12}$ kg/mm^3. The magnitude of the rings initial velocity is $v_0 = 30$ m/s. The cell size is $h = 0.625$ mm or a grid of 640×320 cells is used. The total number of particles is 45, 000 and the shape functions used are cubic B-splines.

The first thing to notice when looking at the energy evolution profiles (Fig. 9.2) is that the results for both USF and MUSL are virtually identical. Second, as expected, PIC is highly dissipative. Third, more energy is dissipated when using USL than



(a) PIC (b) FLIP

Fig. 9.2 Evolution of the total energy during the impact of two compressible Neo-Hookean rings obtained with ULMPM using cubic B-splines and 4 particles per cell

when using USF (and MUSL). Finally, and more importantly, with USF, the level of energy are similar between the beginning and the end of the simulation.

Following the analysis published by Bardenhagen (2002), a qualitative analysis of the energy dissipation of the MPM is made. Distinction will be made between USF and USL (single mapping) as well as PIC and FLIP, hence generating four different cases. A special note is added at the end of the section to explain how the case of double mapping is energetically equivalent to USF.

We are interested in the change of total energy in the system during a single time step and in the absence of heat transfer into the system. In the MPM, since the grid is only used as a computational pad to provide kinematic updates, the system of interest is the ensemble of particles (or material points). The total energy change on the particles over a time step, $\Delta E_{\text{particles}}$ is the sum of the change in kinetic and strain energies, $\Delta KE_{\text{particles}}$ and $\Delta SE_{\text{particles}}$, respectively. In the absence of external work, therefore, $\Delta E_{\text{particles}}$ should be zero (case in which the energy would be conserved). In reality, an error ΔE_{error} is made such that:

$$\Delta E_{\text{particles}} = \Delta E_{\text{error}} = \Delta KE_{\text{particles}} + \Delta SE_{\text{particles}} \tag{9.14}$$

Bardenhagen (2002) splits this error into two different contributions: an interpolation error and an algorithmic error:

$$\Delta E_{\text{error}} = -\Delta E_{\text{interpolation}} - \Delta E_{\text{algorithm}} \tag{9.15}$$

Since the background grid acts as a pad for all computations, the computed change of kinetic energy on the grid during one time step should translate into the exact same change of kinetic energy on the particles during this same time step. If not, it means that errors have been made during the interpolation back and forth from the particles to the grid. This interpolation error is thus expressed as:

$$\Delta E_{\text{interpolation}} = \Delta KE_{\text{nodes}} - \Delta KE_{\text{particles}} \tag{9.16}$$

The other part of the total error is the error made by the algorithm simply expressed as:

$$\Delta E_{\text{algorithm}} = -\Delta E_{\text{error}} - \Delta E_{\text{interpolation}} = -\Delta KE_{\text{nodes}} - \Delta SE_{\text{particles}} \tag{9.17}$$

where ΔKE_{nodes} is the change of kinetic energy on the grid. It is expressed as follows:

$$\Delta KE_{\text{nodes}} = \frac{1}{2}\sum_I m_I^t \mathbf{v}_I^{t+\Delta t} \cdot \mathbf{v}_I^{t+\Delta t} - \frac{1}{2}\sum_I m_I^t \mathbf{v}_I^t \cdot \mathbf{v}_I^t = \sum_I m_I^t \Delta \mathbf{v}_I \cdot \left(\mathbf{v}_I^t + \frac{1}{2}\Delta \mathbf{v}_I\right) \tag{9.18}$$

and $\Delta KE_{\text{particles}}$ is the change of kinetic energy on the particles, which is given by

$$\Delta KE_{\text{particles}} = \frac{1}{2}\sum_p m_p^t \mathbf{v}_p^{t+\Delta t}\cdot\mathbf{v}_p^{t+\Delta t} - \frac{1}{2}\sum_p m_p^t \mathbf{v}_p^t\cdot\mathbf{v}_p^t = \sum_p m_p^t \Delta\mathbf{v}_p\cdot\left(\mathbf{v}_p^t + \frac{1}{2}\Delta\mathbf{v}_p\right) \tag{9.19}$$

Using the PIC velocity update, $\mathbf{v}_p^{t+\Delta t} = \sum_I \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t}$, Eq. (9.19) becomes:

$$\Delta KE_{\text{particles}} = \frac{1}{2}\sum_I\sum_J\sum_p m_p^t \phi_I(\mathbf{x}_p^t)\phi_J(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t}\cdot\mathbf{v}_J^{t+\Delta t} - \frac{1}{2}\sum_p m_p^t\mathbf{v}_p^t\cdot\mathbf{v}_p^t \tag{9.20}$$

$$= \frac{1}{2}\sum_{I,J}\mathbf{v}_I^{t+\Delta t}\cdot M_{IJ}\mathbf{v}_J^{t+\Delta t} - \frac{1}{2}\sum_p m_p^t\mathbf{v}_p^t\cdot\mathbf{v}_p^t \tag{9.21}$$

Therefore, using the results from Love and Sulsky (2006b), substituting Eq. (9.21) into Eq. (9.16) yields:

$$\Delta E_{\text{interpolation}} = \frac{1}{2}\left[\sum_{I,J}\mathbf{v}_I^{t+\Delta t}\cdot(\bar{M}_{IJ}-M_{IJ})\mathbf{v}_J^{t+\Delta t} + \sum_{I,J}\mathbf{v}_I^t\cdot(\bar{M}_{IJ}-M_{IJ})\mathbf{v}_J^t + \sum_p m_p\left\|\mathbf{v}_p^t - \sum_I\phi_I(\mathbf{x}_p^t)\mathbf{v}_I^t\right\|^2\right] \tag{9.22}$$

Using the FLIP velocity update, $\Delta\mathbf{v}_p = \sum_I \phi_I(\mathbf{x}_p^t)\Delta\mathbf{v}_I$, Eq. (9.19) becomes:

$$\Delta E_{\text{interpolation}} = \sum_I m_I^t\Delta\mathbf{v}_I\cdot\left(\mathbf{v}_I^t+\frac{1}{2}\Delta\mathbf{v}_I\right) - \sum_p m_p\sum_I\phi_I(\mathbf{x}_p^t)\Delta\mathbf{v}_I\cdot\left(\mathbf{v}_p^t+\frac{1}{2}\sum_J\phi_J(\mathbf{x}_p^t)\Delta\mathbf{v}_J\right)$$

$$= \sum_I m_I^t\Delta\mathbf{v}_I\cdot\left(\mathbf{v}_I^t+\frac{1}{2}\Delta\mathbf{v}_I\right) - \sum_I\Delta\mathbf{v}_I\cdot\sum_p\phi_I(\mathbf{x}_p^t)m_p\mathbf{v}_p^t - \frac{1}{2}\sum_{I,J}M_{IJ}\Delta\mathbf{v}_I\cdot\Delta\mathbf{v}_J \tag{9.23}$$

$$= \frac{1}{2}\sum_I m_I^t\Delta\mathbf{v}_I\cdot\Delta\mathbf{v}_I - \frac{1}{2}\sum_{I,J}M_{IJ}\Delta\mathbf{v}_I\cdot\Delta\mathbf{v}_J$$

$$= \frac{1}{2}\sum_{I,J}\Delta\mathbf{v}_I\cdot(\bar{M}_{IJ}-M_{IJ})\Delta\mathbf{v}_J \tag{9.24}$$

The symmetric matrix $(\bar{M}_{IJ} - M_{IJ})$ being positive-semi-definite (Love and Sulsky 2006b), the interpolation error is always greater or equal to zero, i.e., $\Delta E_{\text{interpolation}} \geq 0$ with both PIC and FLIP. Moreover, it is obvious that the magnitude of the interpolation error is greater when using PIC than FLIP.

The conservation of energy (see Eq. (2.20)) states that $\rho^{De}/_{Dt} = \mathbf{D}:\boldsymbol{\sigma}$. Using this equation and the transport theorem, the rate of change in strain energy is:

$$\frac{dSE}{dt} = \int_\Omega \mathbf{D}:\boldsymbol{\sigma}\, dv \tag{9.25}$$

where Ω is the current configuration of the solid. The discretization of Eq. (9.25) gives:

$$\frac{dSE}{dt} = \sum_p \mathbf{D}_p : \boldsymbol{\sigma}_p V_p \tag{9.26}$$

Noting that the strain increment $\Delta\boldsymbol{\varepsilon}_p = \mathbf{D}_p \Delta t$ and substituting the stress by its first order Taylor approximation gives for infinitesimal deformations (volume remains unchanged):

$$\Delta SE_{\text{particles}} = \sum_p \boldsymbol{\varepsilon}_p : \frac{\boldsymbol{\sigma}_p^{t+\Delta t} + \boldsymbol{\sigma}_p^t}{2} V_p + \mathcal{O}(\Delta t)^2 \tag{9.27}$$

Since no external work is considered, $\mathbf{f}_I^{\text{int}} = \mathbf{0}$ and the change of node velocity is

$$\Delta \mathbf{v}_I^t = -\frac{1}{m_I^t} \sum_p V_p^t \boldsymbol{\sigma}_p \nabla \phi_I(\mathbf{x}_p^t) \Delta t \tag{9.28}$$

Coming back to the change in kinetic energy on the grid, it is easy to show that Eq. (9.18) can be re-written as

$$\Delta KE_{\text{nodes}} = \frac{1}{4} \sum_I m_I^t \Delta \mathbf{v}_I \cdot \left(\mathbf{v}_I^t + \mathbf{v}_I^{t+\Delta t}\right) + \frac{1}{4} \sum_I \left(\mathbf{v}_I^t + \mathbf{v}_I^{t+\Delta t}\right) \cdot m_I^t \Delta \mathbf{v}_I \tag{9.29}$$

Substituting Eq. (9.28) into Eq. (9.29) yields:

$$\begin{aligned}
\Delta KE_{\text{nodes}} &= -\frac{1}{4} \sum_I \sum_p V_p^t \boldsymbol{\sigma}_p \nabla \phi_I(\mathbf{x}_p^t) \Delta t \cdot \left(\mathbf{v}_I^t + \mathbf{v}_I^{t+\Delta t}\right) - \frac{1}{4} \left(\mathbf{v}_I^t + \mathbf{v}_I^{t+\Delta t}\right) \cdot \sum_I \sum_p V_p^t \boldsymbol{\sigma}_p \nabla \phi_I(\mathbf{x}_p^t) \Delta t \\
&= -\frac{1}{2} \sum_p V_p^t \boldsymbol{\sigma}_p \frac{1}{2} \sum_I \left(\nabla \phi_I(\mathbf{x}_p^t) \mathbf{v}_I^t + \mathbf{v}_I^t \nabla \phi_I(\mathbf{x}_p^t)\right) \Delta t \\
&\quad - \frac{1}{2} \sum_p V_p^t \boldsymbol{\sigma}_p \frac{1}{2} \sum_I \left(\nabla \phi_I(\mathbf{x}_p^t) \mathbf{v}_I^{t+\Delta t} + \mathbf{v}_I^{t+\Delta t} \nabla \phi_I(\mathbf{x}_p^t)\right) \Delta t \\
&= -\frac{1}{2} \sum_p V_p^t \boldsymbol{\sigma}_p \left(\Delta \boldsymbol{\varepsilon}_p^t + \Delta \boldsymbol{\varepsilon}_p^{t+\Delta t}\right)
\end{aligned} \tag{9.30}$$

The algorithmic error is finally given by combining Eqs. (9.30) and (9.27):

$$\Delta E_{\text{algorithm}} = \frac{1}{2} \sum_p V_p^t \boldsymbol{\sigma}_p : \left(\Delta \boldsymbol{\varepsilon}_p^{t+\Delta t} + \Delta \boldsymbol{\varepsilon}_p^t\right) - \sum_p V_p^t \frac{\boldsymbol{\sigma}_p^{t+\Delta t} + \boldsymbol{\sigma}_p^t}{2} : \Delta \boldsymbol{\varepsilon}_p \tag{9.31}$$

This energy contribution is of the order of Δt^2, but unlike the interpolation error, it can be of either sign. Moreover, it is strongly dependent on when the stress state is updated, as we shall see next.

Update Stress First (USF). The solution procedure for the USF algorithm implies that the stresses and the strains are calculated directly from the nodes velocities $\mathbf{v}_I^t$ obtained from the initial interpolation from the particles to the grid (P2G step) as shown in Algorithm 13. Since the stress is updated right at the beginning of the step, $\boldsymbol{\sigma}_p = \boldsymbol{\sigma}_p^{t+\Delta t}$ while $\Delta\boldsymbol{\varepsilon}_p = \Delta\boldsymbol{\varepsilon}_p^t$. Therefore, the grid velocity increment is

$$\Delta\mathbf{v}_I^t = -\frac{1}{m_I^t}\sum_p V_p^t \boldsymbol{\sigma}_p^{t+\Delta t}\nabla\phi_I(\mathbf{x}_p^t)\Delta t \tag{9.32}$$

The change in kinetic energy is:

$$\Delta KE_{\text{nodes}} = -\frac{1}{2}\sum_p V_p^t \boldsymbol{\sigma}_p^{t+\Delta t}\left(\Delta\boldsymbol{\varepsilon}_p^t + \Delta\boldsymbol{\varepsilon}_p^{t+\Delta t}\right) \tag{9.33}$$

and the change in strain energy:

$$\Delta SE_{\text{particles}} = \sum_p \boldsymbol{\varepsilon}_p^t : \frac{\boldsymbol{\sigma}_p^{t+\Delta t} + \boldsymbol{\sigma}_p^t}{2} V_p \tag{9.34}$$

Therefore, the algorithm error becomes:

$$\Delta E_{\text{algorithm}} = \frac{1}{2}\sum_p V_p^t \left(\boldsymbol{\sigma}_p^{t+\Delta t} : \Delta\boldsymbol{\varepsilon}_p^{t+\Delta t} - \boldsymbol{\sigma}_p^t : \Delta\boldsymbol{\varepsilon}_p^t\right) \tag{9.35}$$

In practice, the velocity gradient is used to update the stress tensor:

$$\boldsymbol{\sigma}_p^{t+\Delta t} = \boldsymbol{\sigma}_p^t + \mathbf{L}_p : \Delta\boldsymbol{\varepsilon}_p^t \tag{9.36}$$

Substituting Eq. (9.36) into Eq. (9.35) gives:

$$\Delta E_{\text{algorithm}} = \frac{1}{2}\sum_p V_p^t \left(\boldsymbol{\sigma}_p^{t+\Delta t} : (\Delta\boldsymbol{\varepsilon}_p^{t+\Delta t} - \Delta\boldsymbol{\varepsilon}_p^t) + \Delta\boldsymbol{\varepsilon}_p^t : \mathbf{L}_p : \Delta\boldsymbol{\varepsilon}_p^t\right) \tag{9.37}$$

Noting that

$$\Delta\boldsymbol{\varepsilon}_p^{t+\Delta t} = \frac{1}{2}\sum_I \left(\nabla\phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t} + \mathbf{v}_I^{t+\Delta t}\nabla\phi_I(\mathbf{x}_p^t)\right)$$

and $\Delta\boldsymbol{\varepsilon}_p^t = \frac{1}{2}\sum_I \left(\nabla\phi_I(\mathbf{x}_p^t)\mathbf{v}_I^t + \mathbf{v}_I^t\nabla\phi_I(\mathbf{x}_p^t)\right)$, substituting Eq. (9.32) into Eq. (9.37), one gets:

$$\Delta E_{\text{algorithm}} = -\frac{1}{2}\sum_I m_I^t \Delta\mathbf{v}_I \cdot \Delta\mathbf{v}_I + \frac{1}{2}\sum_p V_p^t \Delta\boldsymbol{\varepsilon}_p^t : \mathbf{L}_p : \Delta\boldsymbol{\varepsilon}_p^t \tag{9.38}$$

Finally, by substituting Eq. (9.38), and respectively Eq. (9.22) or Eq. (9.24) into Eq. (9.15) the total error increment is found to be:

$$\begin{aligned}\Delta E_{\text{error}} = -\frac{1}{2}\Bigg[&\sum_{I,J}\mathbf{v}_I^{t+\Delta t}\cdot(\bar{M}_{IJ}-M_{IJ})\mathbf{v}_J^{t+\Delta t}+\sum_{I,J}\mathbf{v}_I^{t}\cdot(\bar{M}_{IJ}-M_{IJ})\mathbf{v}_J^{t}\\&+\sum_p m_p||\mathbf{v}_p^t-\sum_I\phi_I(\mathbf{x}_p^t)\mathbf{v}_I^t||^2\Bigg]\\&+\frac{1}{2}\sum_{I,J}\Delta\mathbf{v}_I\cdot\bar{M}_{IJ}\Delta\mathbf{v}_J-\frac{1}{2}\sum_p V_p^t\Delta\boldsymbol{\varepsilon}_p^t:\mathbf{L}_p:\Delta\boldsymbol{\varepsilon}_p^t\end{aligned} \tag{9.39}$$

for PIC, or

$$\Delta E_{\text{error}} = \frac{1}{2}\sum_{I,J}\Delta\mathbf{v}_I\cdot M_{IJ}\Delta\mathbf{v}_J-\frac{1}{2}\sum_p V_p^t\Delta\boldsymbol{\varepsilon}_p^t:\mathbf{L}_p:\Delta\boldsymbol{\varepsilon}_p^t \tag{9.40}$$

for FLIP.

When using the FLIP velocity update, ΔE_{error} as written in Eq. (9.40) suggests of a trade-off between an incremental change in kinetic energy and an incremental change in strain energy as pointed out by Bardenhagen (2002). Therefore, ΔE_{error} can be expected to be low. This is unfortunately not the case when using the PIC velocity update where the incremental change in strain energy cannot counterbalance the other terms.

Update Stress Last (USL). Update stress last is the common MPM formulation. With this formulation, the stresses and strains are calculated at the end of the time step (see Algorithm 1) calculated from the updated nodes' velocities. Therefore the strain increments are $\Delta\boldsymbol{\varepsilon}_p = \Delta\boldsymbol{\varepsilon}_p^{t+\Delta t}$, and the stress $\boldsymbol{\sigma}_p^t$ is used to compute the grid velocity increment. Therefore, during the whole step $\boldsymbol{\sigma}_p = \boldsymbol{\sigma}_p^t$ and:

$$\Delta\mathbf{v}_I^t = -\frac{1}{m_I^t}\sum_p V_p^t\boldsymbol{\sigma}_p^t\nabla\phi_I(\mathbf{x}_p^t)\Delta t \tag{9.41}$$

The change in kinetic energy is:

$$\Delta KE_{\text{nodes}} = -\frac{1}{2}\sum_p V_p^t\boldsymbol{\sigma}_p^t\left(\Delta\boldsymbol{\varepsilon}_p^t+\Delta\boldsymbol{\varepsilon}_p^{t+\Delta t}\right) \tag{9.42}$$

and the change in strain energy:

$$\Delta SE_{\text{particles}} = \sum_p\boldsymbol{\varepsilon}_p^{t+\Delta t}:\frac{\boldsymbol{\sigma}_p^{t+\Delta t}+\boldsymbol{\sigma}_p^t}{2}V_p \tag{9.43}$$

Therefore, the algorithm error becomes:

$$\Delta E_{\text{algorithm}} = \frac{1}{2}\sum_p V_p^t \left(\boldsymbol{\sigma}_p^t : \Delta\boldsymbol{\varepsilon}_p^t - \boldsymbol{\sigma}_p^{t+\Delta t} : \Delta\boldsymbol{\varepsilon}_p^{t+\Delta t}\right) \tag{9.44}$$

The algorithm error has exactly the opposite sign as for USF (see Eq. (9.35)). This highlights the dramatic effects of the moment at which the stresses are updated. Starting with exactly the same conditions at the beginning of a time step, with one formulation the energy error due to the algorithm would be positive in one case, and negative in the other. However, it should be kept in mind that the algorithm error is just one part of the whole error.

The equation used in this case to update the stress tensor is:

$$\boldsymbol{\sigma}_p^{t+\Delta t} = \boldsymbol{\sigma}_p^t + \mathbf{L}_p : \Delta\boldsymbol{\varepsilon}_p^{t+\Delta t} \tag{9.45}$$

Substituting Eq. (9.45) into Eq. (9.44) gives:

$$\Delta E_{\text{algorithm}} = \frac{1}{2}\sum_p V_p^t \left(\boldsymbol{\sigma}_p^t : (\Delta\boldsymbol{\varepsilon}_p^t - \Delta\boldsymbol{\varepsilon}_p^{t+\Delta t}) + \Delta\boldsymbol{\varepsilon}_p^{t+\Delta t} : \mathbf{L}_p : \Delta\boldsymbol{\varepsilon}_p^{t+\Delta t}\right) \tag{9.46}$$

Noting that

$$\Delta\boldsymbol{\varepsilon}_p^{t+\Delta t} = \frac{1}{2}\sum_I \left(\nabla\phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t} + \mathbf{v}_I^{t+\Delta t}\nabla\phi_I(\mathbf{x}_p^t)\right)$$

and $\Delta\boldsymbol{\varepsilon}_p^t = \frac{1}{2}\sum_I \left(\nabla\phi_I(\mathbf{x}_p^t)\mathbf{v}_I^t + \mathbf{v}_I^t\nabla\phi_I(\mathbf{x}_p^t)\right)$, substituting Eq. (9.41) into Eq. (9.46), one gets:

$$\Delta E_{\text{algorithm}} = \frac{1}{2}\sum_I m_I^t \Delta\mathbf{v}_I \cdot \Delta\mathbf{v}_I - \frac{1}{2}\sum_p V_p^t \Delta\boldsymbol{\varepsilon}_p^{t+\Delta t} : \mathbf{L}_p : \Delta\boldsymbol{\varepsilon}_p^{t+\Delta t} \tag{9.47}$$

Finally, by substituting Eq. (9.47), and respectively Eq. (9.22) or Eq. (9.24) into Eq. (9.15) the total error increment is found to be

$$\begin{aligned}\Delta E_{\text{error}} = -\frac{1}{2}\Bigg[&\sum_{I,J}\mathbf{v}_I^{t+\Delta t}\cdot(\bar{M}_{IJ} - M_{IJ})\mathbf{v}_J^{t+\Delta t} + \sum_{I,J}\mathbf{v}_I^t\cdot(\bar{M}_{IJ} - M_{IJ})\mathbf{v}_J^t \\ &+ \sum_p m_p\left\|\mathbf{v}_p^t - \sum_I \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^t\right\|^2\Bigg] \\ &- \frac{1}{2}\sum_{I,J}\Delta\mathbf{v}_I\cdot\bar{M}_{IJ}\Delta\mathbf{v}_J + \frac{1}{2}\sum_p V_p^t\Delta\boldsymbol{\varepsilon}_p^{t+\Delta t} : \mathbf{L}_p : \Delta\boldsymbol{\varepsilon}_p^{t+\Delta t}\end{aligned} \tag{9.48}$$

when using the PIC update, and

$$\Delta E_{\text{error}} = -\sum_{I,J} \Delta \mathbf{v}_I \cdot (\bar{M}_{IJ} - M_{IJ}) \Delta \mathbf{v}_J - \frac{1}{2} \sum_{I,J} \Delta \mathbf{v}_I \cdot M_{IJ} \Delta \mathbf{v}_J + \frac{1}{2} \sum_p V_p^t \Delta \boldsymbol{\varepsilon}_p^{t+\Delta t} : \mathbf{L}_p : \Delta \boldsymbol{\varepsilon}_p^{t+\Delta t} \tag{9.49}$$

when using the FLIP update.

In practice, its has been observed that when using the FLIP velocity update with USF, the change in strain energy over time can be negligible Fig. 9.2b. Hence, $\Delta E_{\text{error}} \approx 0$, and from Eq. (9.40) it can be deduced that $\frac{1}{2} \sum_{I,J} \Delta \mathbf{v}_I \cdot M_{IJ} \Delta \mathbf{v}_J \approx \frac{1}{2} \sum_p V_p^t \Delta \boldsymbol{\varepsilon}_p^t : \mathbf{L}_p : \Delta \boldsymbol{\varepsilon}_p^t$. Assuming that this is true, the MPM algorithm using the update stress last formulation would be strictly dissipative for both PIC and FLIP velocity update schemes. This explains why the numerical investigations presented earlier (Fig. 9.2b) have shown that USL is more energy dissipative than USF.

Modified Update Stress Last (MUSL). In the modified update stress last formulation, the stresses are calculated at the end of the time step, after the re-mapping of node velocities using the updated particles velocities (double mapping) as shown in Algorithm 2. To the exception of the very first time step, this double mapping is equivalent to mapping the node velocities at the beginning of the next time step and calculating the stresses right after, but using the shape functions evaluated at time t and not $t + \Delta t$.

The difference between MUSL and USF resides in the computation of the internal forces. Their difference for a given grid node is:

$$\begin{aligned}
\left(f_I^{\text{int},t}\right)_{USF} - \left(f_I^{\text{int},t}\right)_{MUSL} &= -\sum_p V_p^t \left[\left(\boldsymbol{\sigma}_p^{t+\Delta t}\right)_{USF} - \left(\boldsymbol{\sigma}_p^t\right)_{MUSL} \right] \nabla \phi_I(\mathbf{x}_p^t) \\
&= -\sum_p V_p^t \left[\sum_J \nabla \phi_J(\mathbf{x}_p^t) \mathbf{v}_J^t - \sum_J \nabla \phi_J(\mathbf{x}_p^{t-\Delta t}) \mathbf{v}_J^t \right] : \Delta \boldsymbol{\varepsilon}_p \nabla \phi_I(\mathbf{x}_p^t) \\
&= -\sum_p V_p^t \sum_J \left[\nabla \phi_J(\mathbf{x}_p^t) - \nabla \phi_J(\mathbf{x}_p^{t-\Delta t}) \right] \mathbf{v}_J^t : \Delta \boldsymbol{\varepsilon}_p \nabla \phi_I(\mathbf{x}_p^t)
\end{aligned} \tag{9.50}$$

Since $\nabla \phi_J(\mathbf{x}_p^{t-\Delta t}) = \nabla \phi_J(\mathbf{x}_p^t) + \mathcal{O}(\Delta t)$, Eq. (9.50) becomes:

$$\left(f_I^{\text{int},t}\right)_{USF} - \left(f_I^{\text{int},t}\right)_{MUSL} = \mathcal{O}(\Delta t) \tag{9.51}$$

Therefore, the internal forces obtained by either the USF or the MUSL formulation are equivalent to the first order. And this explains why the energy profiles obtained with either formulations cannot be distinguished in Fig. 9.2.

9.2 The Method of Manufactured Solutions (MMS)

The method of manufactured solutions (MMS) provides a framework to verify nonlinear codes. In the MMS, the solution (i.e., the displacement field in solid mechanics) of the model equations is assumed *a priori* i.e., being manufactured. Given the assumed prescribed displacements, the constitutive model can be evaluated to determine the corresponding stress field. Next, the divergence of the stress and the acceleration are evaluated, and the required body forces to achieve these solutions are analytically determined from the equations of motion. Note that boundary conditions (e.g. for the velocities) and initial conditions (e.g. for the velocities and stresses) are also determined from the manufactured solutions. One then execute the code (an MPM code in our context) with this body force vector and boundary/initial conditions for different grid resolutions. The numerical solutions are then compared with the manufactured ones (Knupp and Salari 2003); an error is computed for each grid resolution. From this, a convergence rate can be obtained. An excellent report on the MMS can be found at http://prod.sandia.gov/techlib/access-control.cgi/2000/001444.pdf.

Wallstedt and Guilkey (2008) was the first to use the MMS to check the convergence of the MPM. Since then, the MMS has been used more. For example, a suite of code verification tests for solid mechanics problems is presented in Kamojjala et al. (2015) for rate-independent constitutive models.

For readers unfamiliar with the method, we present a step-by-step derivation of the body force for a 1D problem in Sect. 9.2.1 and a 2D problem in Sect. 9.2.2. This is followed by a discussion on error norms in Sect. 9.2.4. Section 9.2.5 provides a procedure to obtain the so-called convergence curve and convergence rate for a chosen manufactured solution.

9.2.1 An One Dimensional Manufactured Solution

To demonstrate the MMS , in what follows we present the method in one dimension. It should be noted that the solutions are typically manufactured in the total Lagrangian form i.e., with respect to the initial configuration in the MMS as it is more convenient.

Now, we consider a one dimensional bar of length L. Let denote by X the material coordinates i.e., coordinates in the reference configuration. Now, we introduce a dimensionless coordinate $\bar{X} = {}^{X}/_{L}$. The manufactured displacement is assumed to be

$$u(\bar{X}, t) = G \sin(\pi \bar{X}) \sin\left(\frac{c\pi t}{L}\right) \tag{9.52}$$

where G is the maximum amplitude of the displacement; $c = \sqrt{E/\rho}$ and E denotes the Young modulus. Now, to simplify the subsequent derivation, we assume that

$L = 1$ m, and thus $\bar{X} = X$. The period is thus given by $T = {}^{2\pi}/_{c\pi}$; and the time domain is $0 \leq t \leq T$ i.e., one period of oscillation is considered. As can be seen, the manufactured displacements are expressed in terms of sine and cosine functions i.e., smooth functions are commonly used, see e.g. Wallstedt and Guilkey (2008) even though they are not representative of general material deformations.

The velocity and acceleration are thus given by

$$\begin{aligned} v(X,t) &= \pi c G \sin(\pi X)\cos(c\pi t) \\ a(X,t) &= -\pi^2 c^2 G \sin(\pi X)\sin(c\pi t) = -\pi^2 c^2 u(X,t) \end{aligned} \tag{9.53}$$

where Eq. (9.52) was used.

One has to choose a constitutive model so that the stress can be determined analytically. We use a Neo-Hookean material where the 1st PK stress P is given by

$$P = \lambda \ln(J) F^{-1} + \mu F^{-1}(FF - 1) \tag{9.54}$$

with λ, μ being the Lamé constants and $J = F$ is the Jacobian of the deformation. Using the displacement given in Eq. (9.52), the deformation gradient F is written as

$$F(X,t) = 1 + \frac{\partial u}{\partial X} = 1 + \pi G \cos(\pi X)\sin(c\pi t) \tag{9.55}$$

which results in the following expression for the spatial derivative of F

$$\frac{\partial F}{\partial X} = -\pi^2 G \sin(\pi X)\sin(c\pi t) = -\pi^2 u(X,t) \tag{9.56}$$

where use was made of Eq. (9.52). The divergence of the stress, appearing in the linear momentum balance equation, is thus given by

$$\frac{\partial P}{\partial X} = \frac{\partial F}{\partial X}\left[\frac{\lambda}{F^2}(1 - \ln(F)) + \mu\left(1 + \frac{1}{F^2}\right)\right] \tag{9.57}$$

From the momentum equation given as follows

$$\rho_0 a(X,t) = \frac{\partial P}{\partial X} + \rho_0 b(X,t) \tag{9.58}$$

one can solve for the body force

$$b(X,t) = \frac{\pi^2 u(X,t)}{\rho_0}\left[\frac{\lambda}{F^2}(1 - \ln(F)) + \mu\left(1 + \frac{1}{F^2}\right) - E\right] \tag{9.59}$$

Besides, initial conditions are given by

$$\begin{aligned} v(X,0) &= \pi c G \sin(\pi X) \\ \sigma(X,0) &= \frac{1}{J}\left[\lambda \ln(F(X,0)) + \mu(F(X,0)F(X,0) - 1)\right] = 0 \end{aligned} \tag{9.60}$$

and boundary conditions are written as

$$v(0,t) = 0, \quad v(1,t) = 0 \tag{9.61}$$

9.2.2 A Two Dimensional MMS

Herein, we present an extension of the previous MMS to three dimensions. The solid is a unit cube occupying the domain delimited by $0 \leq X_1 \leq 1$, $0 \leq X_2 \leq 1$, and $0 \leq X_3 \leq 1$. The starting point are the following prescribed displacements

$$\mathbf{u}(\mathbf{X},t) = G \sin(\pi \mathbf{X}) \sin\left(\sqrt{\frac{E}{\rho_0}}\pi t + \boldsymbol{\phi}\right) \tag{9.62}$$

where G is the maximum amplitude of displacement, E denotes Young's modulus, and $\boldsymbol{\phi} = [0, \pi, \pi]$, an arbitrary phase angle. $\mathbf{X}$ denotes the material coordinates in the reference configuration. One period of oscillation is considered, i.e. the time domain is $0 \leq t \leq T$, with $T = {}^{2\pi}/_{\pi\sqrt{E/\rho_0}}$. Note that the assumed displacements where the horizontal displacement is a function of X and the vertical displacement depends on Y leads to a deformation without shearing and thus a diagonal deformation gradient $\mathbf{F}$ which allows to determine the body force analytically by hand calculations. It should be noted that the two components of the manufactured displacement field cannot be chosen independently but should be physically reasonable as shown in Fig. 9.3.

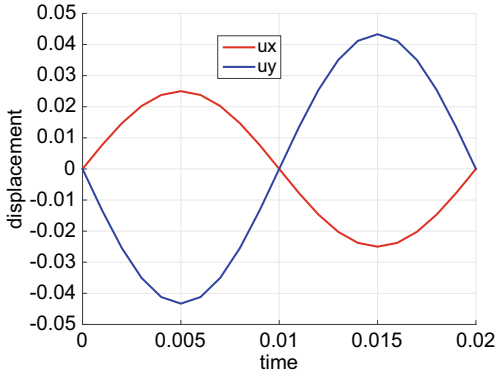


Fig. 9.3 Evolution of the displacements in time at point (1/6, 1/3)

The goal now is to determine the associated body forces. They are calculated by inverting the momentum equation:

$$\rho_0 \mathbf{a}(\mathbf{X}, t) = \nabla_0 \mathbf{P}(\mathbf{X}, t) + \rho_0 \mathbf{b}(\mathbf{X}, t) \tag{9.63}$$

which depends on the acceleration $\mathbf{a}(\mathbf{X}, t)$ and the divergence of the stress $\nabla_0 \mathbf{P}(\mathbf{X}, t)$.

First, let's determine the acceleration. This is done by taking the second derivative of the displacement, which gives:

$$\mathbf{a}(\mathbf{X}, t) = -\pi^2 \frac{E}{\rho_0} G \sin(\pi \mathbf{X}) \sin\left(\sqrt{\frac{E}{\rho_0}} \pi t + \boldsymbol{\phi}\right) = -\pi^2 \frac{E}{\rho_0} \mathbf{u} \tag{9.64}$$

Second, let's determine $\nabla_0 \mathbf{P}(\mathbf{X}, t)$. Owing to the use of the Neo-Hookean model, $\mathbf{P}$ depends only on the deformation matrix $\mathbf{F}$ which is calculated by taking the spatial derivative of the displacement with respect to the initial position:

$$F_{ii}(\mathbf{X}, t) = 1 + \pi G \cos(\pi X_i) \sin\left(\sqrt{\frac{E}{\rho_0}} \pi t + \phi_i\right) \tag{9.65}$$

Then, by taking the spatial derivative of $\mathbf{F}$, one gets:

$$\nabla_0 \mathbf{F}(\mathbf{X}, t) = \begin{bmatrix} -\pi^2 G \sin(\pi X_1) \sin\left(\sqrt{\frac{E}{\rho_0}} \pi t + \phi_1\right) \\ -\pi^2 G \sin(\pi X_2) \sin\left(\sqrt{\frac{E}{\rho_0}} \pi t + \phi_2\right) \\ -\pi^2 G \sin(\pi X_3) \sin\left(\sqrt{\frac{E}{\rho_0}} \pi t + \phi_3\right) \end{bmatrix} = -\pi^2 \mathbf{u}(\mathbf{X}, t) \tag{9.66}$$

Eventually, the divergence of the stress is given by, using Eq. (4.6)

$$\nabla_0 \mathbf{P}(\mathbf{X}, t) = \begin{bmatrix} \pi^2 u_1 \left(\frac{\lambda}{F_{11}^2} (1 - \ln(F_{11} F_{22} F_{33})) + \mu \left(1 + \frac{1}{F_{11}^2}\right)\right) \\ \pi^2 u_2 \left(\frac{\lambda}{F_{22}^2} (1 - \ln(F_{11} F_{22} F_{33})) + \mu \left(1 + \frac{1}{F_{22}^2}\right)\right) \\ \pi^2 u_3 \left(\frac{\lambda}{F_{33}^2} (1 - \ln(F_{11} F_{22} F_{33})) + \mu \left(1 + \frac{1}{F_{33}^2}\right)\right) \end{bmatrix} \tag{9.67}$$

Finally, one can invert Eq. (9.63) by substituting for both the acceleration and the divergence of the stress using Eqs. (9.64) and (9.67). we can solve for the body force:

$$\mathbf{b}(\mathbf{X},t)=\begin{bmatrix}\dfrac{\pi^2 u_1}{\rho_0}\left(\dfrac{\lambda}{F_{11}^2}\left(1-\ln(F_{11}F_{22}F_{33})\right)+\mu\left(1+\dfrac{1}{F_{11}^2}\right)-E\right)\\ \dfrac{\pi^2 u_2}{\rho_0}\left(\dfrac{\lambda}{F_{22}^2}\left(1-\ln(F_{11}F_{22}F_{33})\right)+\mu\left(1+\dfrac{1}{F_{22}^2}\right)-E\right)\\ \dfrac{\pi^2 u_3}{\rho_0}\left(\dfrac{\lambda}{F_{33}^2}\left(1-\ln(F_{11}F_{22}F_{33})\right)+\mu\left(1+\dfrac{1}{F_{33}^2}\right)-E\right)\end{bmatrix} \tag{9.68}$$

which is a generalization of the 1D body force given in Eq. (9.59).

We also impose the following boundary conditions, obtained using Eq. (9.62), onto the nodes of the background mesh:

$$\mathbf{v}(X_i=0,t)=\mathbf{v}(X_i=1,t)=0 \tag{9.69}$$

as well as the following initial conditions:

$$\mathbf{v}(\mathbf{X},0)=\pi\sqrt{\frac{E}{\rho_0}}G\sin(\pi\mathbf{X})\cos(\boldsymbol{\phi}) \tag{9.70}$$

$$\mathbf{P}(\mathbf{X},0)=\mathbf{0} \tag{9.71}$$

9.2.3 Generalized Vortex Problem

The generalized vortex problem is an example of MMS involving simple shear with superimposed rotation which is a complicated problem to simulate using MPM and its advanced (improved) variants (Brannon et al. 2011; Kamojjala et al. 2015; Wang et al. 2019). This problem features a 2D ring, cf. Figure 9.4a, which is locally subjected to a pure circular motion. The local displacement is purely angular and varies with the radial coordinate. Therefore, the material is only subjected to shear and circular motion, see Fig. 9.4b.

The ring center is at the origin ($x=y=0$) and its inner and outer radii are R_i and R_o, respectively. The current position $\mathbf{x}$ of any given point on the ring is thus given as:

$$\mathbf{x}(t)=\mathbf{Q}(t)\cdot\mathbf{X} \tag{9.72}$$

with $\mathbf{Q}$ being a standard rotation matrix in 2D:

$$\mathbf{Q}(t)=\begin{bmatrix}\cos\alpha & -\sin\alpha\\ \sin\alpha & \cos\alpha\end{bmatrix} \tag{9.73}$$

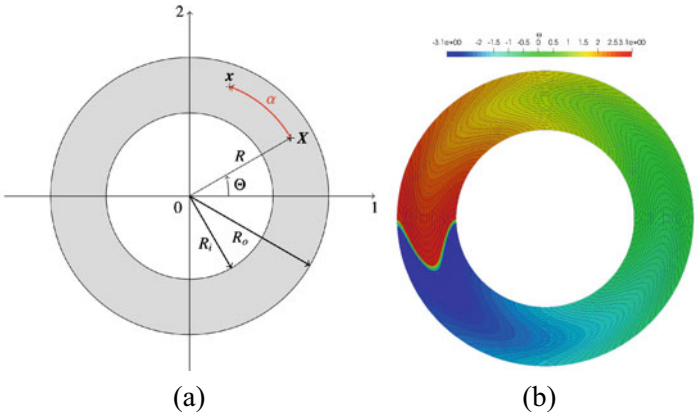


Fig. 9.4 Generalized vortex problem. Geometry and variable definition (**a**), and deformed shape (**b**) (de Vaucorbeil et al. 2020)

where α is the rotation angle which varies only with time and radial coordinate R ($R = \sqrt{X^2 + Y^2}$). It is given as:

$$\alpha(t, R) = g(t)h(R) \tag{9.74}$$

where $g(t)$ controls the amplitude of the deformation with time and $h(R)$ controls the relative radial variation of the rotation; $g(t)$ is taken as periodic, and $h(R)$ is chosen such that the outer and inner radii do not move: $h(R_i) = h(R_o) = 0$. Zero traction on the boundaries is insured by having: $h'(R_i) = h'(R_o) = 0$ as well. Following Brannon et al. (2011), they are given by:

$$h(R) = 1 - 8\left(\frac{R - \bar{R}}{R_i - R_0}\right)^2 + 16\left(\frac{R - \bar{R}}{R_i - R_0}\right)^4, \quad g(t) = G \sin\left(\frac{\pi t}{T_0}\right) \tag{9.75}$$

with $\bar{R} = (R_0 + R_i)/2$ and T_0 is the period.

Using the procedure shown in Sect. 9.2.1 the body forces necessary to obtain this vortex motion are expressed as (see de Vaucorbeil et al. (2020) for detailed derivation):

$$\begin{aligned} b_1(R, t) &= b_R(R, t)\cos\Theta - b_\Theta(R, t)\sin\Theta \\ b_2(R, t) &= b_\Theta(R, t)\cos\Theta + b_R(R, t)\sin\Theta \end{aligned} \tag{9.76}$$

where:

$$
\begin{aligned}
b_R(R,t) &= \left[\frac{\mu}{\rho_0}\left(3g(t)h'(R) + Rg(t)h''(R)\right) - Rg''(t)h(R)\right]\sin\alpha \\
&\quad + \left[\frac{\mu}{\rho_0}R(g(t)h'(R))^2 - R(g'(t)h(R))^2\right]\cos\alpha \\
b_\Theta(R,t) &= \left[-\frac{\mu}{\rho_0}\left(3g(t)h'(R) + Rg(t)h''(R)\right) + Rg''(t)h(R)\right]\cos\alpha \\
&\quad + \left[\frac{\mu}{\rho_0}R(g(t)h'(R))^2 + R(g'(t)h(R))^2\right]\sin\alpha
\end{aligned}
\tag{9.77}
$$

where $h'(R)$ denotes the first derivative of h with respect to R and $h''(R)$ is the second derivative. Furthermore, the initial velocity and stress are identically zero. And the boundary conditions are that nodes on the ring perimeters (inner and outer) are fixed (as $h(R_i) = h(R_o) = 0$).

Now, one has a well defined problem, which can be solved using an MPM code. The obtained numerical displacement, denoted by $\mathbf{u}^h(\mathbf{X}, t)$, is then compared it with the manufactured (exact) displacement in Eq. (9.52) to assess the performance of the MPM. One needs a norm for this comparison, which is discussed in what follows.

9.2.4 Norms

To assess the accuracy of the numerical solution, one needs a measure of the error made compared to the analytical solution (i.e., the manufactured solutions). However, what error measure to be adopted is confusing as different authors used different definitions. Even worse, some researchers adopted different error measures in different related works. It seems to the authors that their aim was to get convergence forcefully. Therefore, de Vaucorbeil et al. (2020) presented two different errors, the first is un-normalized while the second is. The first error is based on the "distance" measure between the numerical and analytical solution and integrated in space. This measure is a function of time, and at time step t^n it is defined as:

$$
e(t^n) = \sqrt{\frac{\sum_{p=1}^{n_p} V_p^{t^n} \left|\left|\mathbf{u}_p^h(t^n) - \mathbf{u}^{\text{exact}}(\mathbf{X}_p, t^n)\right|\right|^2}{V_{\text{tot}}}} \tag{9.78}
$$

where $\mathbf{u}_p^h(t^n) = \mathbf{x}_p^{t^n} - \mathbf{X}_p$ is the numerical displacement at particle p, $\mathbf{u}^{\text{exact}}(\mathbf{X}_p, t^n)$ is the manufactured displacement at p (e.g. the one given in Eq. (9.52)), n_p is the total number of particles in the solid and V_{tot} its total volume. In the above equation, $||\mathbf{a}||$ denotes the Euclidean norm of vector $\mathbf{a}$. However, to not have to compare the error at every time step, the first error measure is taken as the overall maximum of the error function:

$$
e_1 := \max_n(e(t^n)) \tag{9.79}
$$

This measure of error is not normalized and has the dimension of length. It is therefore dependent on the maximum amplitude of the displacement. One can normalize the error to make it dimensionless. The normalized error measure is defined as

$$e_2 := \sqrt{\frac{\sum_{t^n=0}^{t_f} \sum_{p=1}^{n_p} V_p^{t^n} \left|\left|\mathbf{u}_p^h(t^n) - \mathbf{u}^{\text{exact}}(\mathbf{X}_p, t^n)\right|\right|^2}{\sum_{t^n=0}^{T} \sum_{p=1}^{n_p} V_p^{t^n} \left|\left|\mathbf{u}^{\text{exact}}(\mathbf{X}_p, t^n)\right|\right|^2}} \tag{9.80}$$

For the TLMPM, as the spatial integration is carried over the initial configuration, one just simply replaces $V_p^{t^n}$ by V_p^0 in Eqs. (9.79) and (9.80).

9.2.5 Convergence Rate

It is well known that the error of the numerical solution measured in some norms is related to the element size h by the following equation (Strang and Fix 1973; Ciarlet and Lions 1991)

$$e(h) \approx Ch^{k(p)} \tag{9.81}$$

where C is a positive constant, $k(p)$ is a positive integer and p denotes the order of the basis functions: $p = 1$ for linear elements, $p = 2$ for quadratic elements etc. Eq. (9.81) furnishes a practical way to check the convergence of a numerical method by taking the logarithm of both sides of the above equation

$$e(h) \approx Ch^k \rightarrow \log(E(h)) \approx \log(C) + k \log(h) \tag{9.82}$$

which indicates that, on a log-log plot the relation between the error and the mesh size is a line with a slope k; k is the *convergence rate*.

For time dependent problems, to manifest the expected theoretical (or optimal) convergence rates in space, the time step and spatial mesh size must be selected such that the space discretization error dominates the time discretization error. Consequently, rather small time steps must be taken. We refer to Strang and Fix (1973); Ciarlet and Lions (1991) for a formal treatment of the mathematical analysis of the FEM which can serve as a basis for the analysis of the MPM.

A procedure to verify the convergence rate of a dynamic MPM code is as follows

- Manufacturing a solution and determining the corresponding body forces and initial/boundary conditions;
- Running an MPM code with body forces, initial/boundary conditions determined from the first step. Repeating this step for different mesh sizes from large to very small;

Table 9.1 An example convergence data produced by a second-order accurate (in space) method

Mesh size h	Error e	Ratio
0.1250000	7.281254249028449e-06	
0.0625000	3.777609954416918e-06	1.93
0.0312500	1.916379665584172e-06	1.97

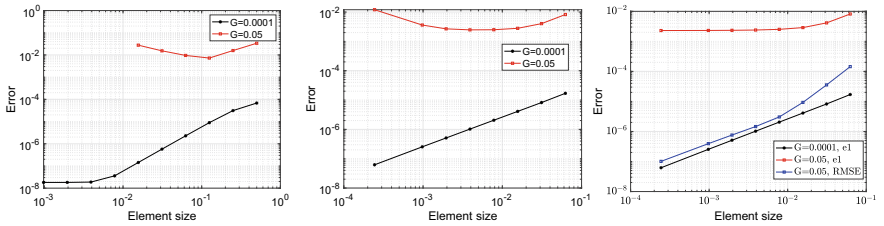
- Defining a proper error norm and compute these norms for every considered meshes;
- Plotting the errors and the mesh sizes h on a log-log scale.

A convergence data is given in Table 9.1 for demonstration. Three meshes are used with a starting mesh size h equals 0.125, and the second mesh size was halved from the first mesh and so on. A convergence curve is obtained by plotting this data and the convergence rate is the slope of the convergence curve in the log-log space and this rate can be conveniently determined in Matlab using the `polyfit` command. Alternatively this rate can be obtained by computing the ratios between successive errors.

The data shown in Table 9.1 demonstrate that to get highly accurate results, one must adopt very fine grids i.e., tiny h. However, in practice we don't have the patience or supercomputing resources to take h extremely small: we want to have good results with h as large as possible! That is why a high-order method i.e., $k \geq 2$ is in favor over a first-order one.

9.2.6 Convergence Rate of the MPM

Convergence tests, see Fig. 9.5 and Sect. 9.5 for details, reveal that MPMs do not converge at an optimal rate (i.e., the convergence rate for L_2 error of the displacement is not two). There are many reasons for this undesired behavior. It is quite clear that quadrature error is one of them as particles are arbitrarily positioned on the grid. Methods to improve this error include MPM variants with smooth weighting functions (GIMP/CPDI/BSMPM) and the standard MPM with the Gaussian quadrature (that is particle data are reconstructed at the quadrature points and they are used for evaluating the weak form integrals similar to the FEM). Wallstedt and Guilkey (2007) have shown that the grid momenta calculation, with linear shape functions, is not able to provide an exact projection of a linear velocity field for arbitrary particle positions. This led Sulsky and Gong (2016) to develop the improved MPM (iMPM) where moving least square (MLS) method is adopted to reconstruct the particle data (velocity, density and stress) at the grid nodes and cell centers.



(a) MPM, MUSL (b) uGIMP, USL (c) cpGIMP, USL

Fig. 9.5 Axis-align unit segment test: convergence of various MPM variants using the unnormalized error measure e_1 given in Eq. (9.79) (de Vaucorbeil et al. 2020)

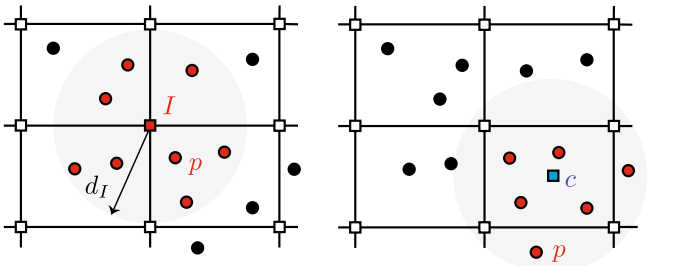
9.3 Moving Least Square MPM

To have a high-order MPM, one needs to fulfill the following conditions: (1) high order grid basis functions such as cubic B-splines, (2) a proper projection of the particle velocities to the grid and (3) accurate integration of the weak form integrals (the nodal mass and forces). Lack of any of these three conditions would result in a first-order MPM for large deformation problems where the particle positions are arbitrary with respect to the grid. Furthermore, the stress update of the constitutive equations must also be high order to have an overall high order accurate method. Herein we focus on the last two conditions as cubic B-splines have been discussed. The device to fulfill these last conditions is moving least square approximation (Gong 2015; Sulsky and Gong 2016). The ideas are as follows.

- Reconstruction of particle momenta using the MLS and project this reconstructed momenta to the grid nodes. This will solve the velocity projection issue of MPM (including GIMP, CPDI);
- Integration of the mass matrix and nodal forces using one-point quadrature rule located at the element centers. This will improve the quadrature error of MPM. Note that CPDI does not subject to this error with the expense of a particle FE mesh. Again the MLS is used to construct the data at the element centers. Precisely the particle stresses and densities are approximated using the MLS and they are computed at the element centers.
- Standard FE shape functions i.e., the hat functions are still employed in the calculation of mass matrix and internal/external forces.

Graphical illustration of the two MLS approximations involved in the above two iterms is given in Fig. 9.6. The last point warrants a further discussion. Using hat functions facilitate the enforcement of boundary conditions. Furthermore the ultimate goal is to have second-order accuracy and hat functions are sufficient for that.

This section introduces the moving least square MPM or iMPM as Gong and Sulsky called it (Gong 2015). It is structured as follows. In Sect. 9.3.1 we briefly recall least square and moving least square approximations. Then in Sect. 9.3.2, we



(a) Velocity projection (b) Density/stress projection

Fig. 9.6 Moving least square approximations **a** to project particle velocities to the grid and **b** to construct cell-centered densities and stresses (de Vaucorbeil et al. 2020)

discuss how the velocity projection is done. After that, one point quadrature is given in Sect. 9.3.3. We conclude the section with implementation details in Sects. 9.3.4 and 9.3.5.

9.3.1 Least Square Approximations

Data approximation from a scattered set of points is required in many applications e.g. computer graphics and visualization, image processing, regression models, supervised learning, and solving partial differential equations using finite element and meshfree methods.

Before presenting the moving least square method that is used in the iMPM, we recall some methods in data fitting. For the sake of simplicity, we first confine to one dimensional case. Considering the well known data fitting problem where we want to find a function $u^h(x)$ fitting the data points (x_I, u_I), $I = 1, 2, \ldots, n$ (Fig. 9.7). The data points can be some experimental measurements such as depth and pore pressure, temperature distribution in time etc.

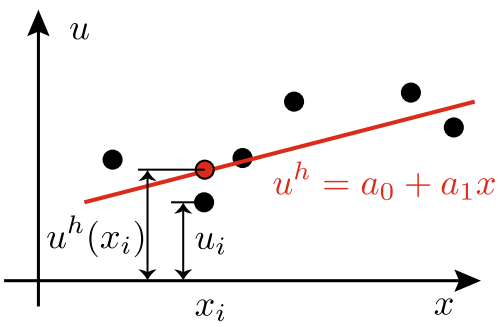


Fig. 9.7 Data fitting in one dimension

Assuming that the approximation function $u^h(x)$ is a polynomial of order m:

$$u^h(x) = a_0 + a_1 x + a_2 x^2 \ldots + a_m x^m \tag{9.83}$$

where the coefficients a_I are to be determined such that we have a best fit to the data points (x_I, u_I). It is more convenient to write the above equation in the following compact form

$$u^h(x) = \mathbf{p}^{\mathrm{T}}(x)\mathbf{a}, \quad \mathbf{a} = \left[a_0\ a_1\ \ldots\ a_m\right]^{\mathrm{T}}, \quad \mathbf{p} = \left[1\ x\ x^2\ \ldots\ x^m\right]^{\mathrm{T}} \tag{9.84}$$

Least square data fitting. The coefficients a_I can be determined by minimizing J– the sum of the square of the difference between u_I and $u^h(x_I)$ i.e., the difference between the given data and the sought-for fitting functions:

$$J = \sum_{I=1}^{n} \left(u^h(x_I) - u_I\right)^2 = \sum_{I=1}^{n} \left(\mathbf{p}^{\mathrm{T}}(x_I)\mathbf{a} - u_I\right)^2 \tag{9.85}$$

J is also referred to as summed square of errors/residuals or summed square of offsets. which indicates that positive or negative error have the same value (data point is above or below the fitting curve) and greater errors are more heavily weighted.

Differentiating J with respect to $a_0, a_1, \ldots, a_m$ are given by

$$\begin{bmatrix} \dfrac{\partial J}{\partial a_0} \\ \dfrac{\partial J}{\partial a_1} \\ \vdots \\ \dfrac{\partial J}{\partial a_m} \end{bmatrix} = \begin{bmatrix} \sum_I 2\left(\mathbf{p}^{\mathrm{T}}(x_I)\mathbf{a} - u_I\right) 1 \\ \sum_I 2\left(\mathbf{p}^{\mathrm{T}}(x_I)\mathbf{a} - u_I\right) x_I \\ \vdots \\ \sum_I 2\left(\mathbf{p}^{\mathrm{T}}(x_I)\mathbf{a} - u_I\right) x_I^m \end{bmatrix} \tag{9.86}$$

And thus, from $\partial J/\partial \mathbf{a} = \mathbf{0}$, one obtains the following system of linear equations to solve for $\mathbf{a}$

$$\left(\sum_{I=1}^{n} \mathbf{p}(x_I)\mathbf{p}^{\mathrm{T}}(x_I)\right)\mathbf{a} = \sum_{I=1}^{n} \mathbf{p}(x_I)u_I \tag{9.87}$$

Examples. We provide some examples to clearly demmonstrate the method. For example, if we intend to use a linear function to fit the data, then $\mathbf{p} = [1, x]^{\mathrm{T}}$. In this case Eq. (9.87) becomes

$$\left(\sum_{I=1}^{n} \begin{bmatrix} 1 & x_I \\ x_I & x_I^2 \end{bmatrix}\right) \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \sum_{I=1}^{n} u_I \begin{bmatrix} 1 \\ x_I \end{bmatrix} \tag{9.88}$$

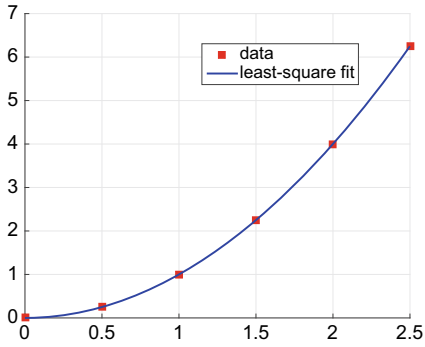


Fig. 9.8 Data fitting in one dimension: second order least square fitting

which can be solved for a_0 and a_1 if the square matrix is invertible which is the case if $n \geq 2$ or more generally $n \geq m$.

In case that a quadratic polynomial is used i.e., $\mathbf{p} = [1, x, x^2]^{\mathrm{T}}$, Eq. (9.87) becomes

$$\left(\sum_{i=1}^{n} \begin{bmatrix} 1 & x_I & x_I^2 \\ x_I & x_I^2 & x_I^3 \\ x_I^2 & x_I^3 & x_I^4 \end{bmatrix} \right) \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \sum_{I=1}^{n} u_I \begin{bmatrix} 1 \\ x_I \\ x_I^2 \end{bmatrix} \tag{9.89}$$

which can be solved for a_0, a_1 and a_2. An example of this second order least square fitting is given in Fig. 9.8.[1]

Weighted least square data fitting. Usually the data (x_I, u_I) have different uncertainties associated with them. It is, therefore, natural to give more weight to measurements of which you are more certain. According to the weighted least square method, the coefficients a_I can be determined by minimizing the weighted sum of the square of the difference between u_I and $u^h(x_i)$:

$$J = \sum_{I=1}^{n} w_I \left(u^h(x_I) - u_I \right)^2 = \sum_{I=1}^{n} w_I \left(\mathbf{p}^{\mathrm{T}}(x_I)\mathbf{a} - u_I \right)^2 \tag{9.90}$$

where w_I denotes a weight at point x_I. The coefficients $\mathbf{a}$ in this case are given by

$$\mathbf{Aa} = \mathbf{Bu} \tag{9.91}$$

where

$$\mathbf{A} = \sum_{I=1}^{n} w_I \mathbf{p}(x_i)\mathbf{p}^{\mathrm{T}}(x_i), \quad \mathbf{B} = \left[w_1 \mathbf{p}(x_1)\ w_2 \mathbf{p}(x_2)\ \ldots\ w_n \mathbf{p}(x_n) \right] \tag{9.92}$$

and $\mathbf{u} = \left[u_1\ u_2\ \ldots\ u_n \right]^{\mathrm{T}}$.

[1] The M-file for this is **leastSquareExample.m**.

Moving least square approximation. This method was introduced in Shepard (1968) in the late 1960s for constructing smooth approximations to fit a specified cloud of points. It was then extended in Lancaster (1981) for general surface generation problems. The most famous application of MLS approximation is probably within the diffuse element method (DEM) and element-free Galerkin (EFG) method. In a MLS scheme the fitting is done locally i.e., the fitting is different from points to points. For a point $\bar{x}$ the approximation now reads

$$u^h(x, \bar{x}) = \mathbf{p}^{\mathrm{T}}(x)\mathbf{a}(\bar{x}) \tag{9.93}$$

where the coefficients $\mathbf{a}$ are no longer constant but varies in space. Following the same procedure of the weighted least square, we obtain the MLS approximation which is written as

$$\begin{aligned} u^h(x) &= \mathbf{p}^{\mathrm{T}}(x)\mathbf{A}^{-1}(x)\mathbf{B}(x)\mathbf{u} \\ &= \mathbf{p}^{\mathrm{T}}(x)\mathbf{A}^{-1}(x)w(x - x_I)\mathbf{p}(x_I)u_I \end{aligned} \tag{9.94}$$

where the second equation allows us to define the so-called MLS basis function $\Phi_I(x)$ given by

$$\Phi_I(x) = \mathbf{p}^{\mathrm{T}}(x)\mathbf{A}^{-1}(x)w(x - x_I)\mathbf{p}(x_I) \tag{9.95}$$

where $w(x - x_I)$ denotes the weight function (or kernel function). The kernel functions have compact support, thus MLS approximation is local. The support size is defined by the so called dilatation parameter or smoothing length. It is critical to solution accuracy, stability and plays the role of the element size in the finite element method. The need to build up and invert the moment matrix $\mathbf{A}$ at a large number of points is the major drawback of the MLS, because of the computational cost and the possibility that the matrix inversion fails.

Shepard approximation. If $\mathbf{p} = [1]$ then one has

$$A = \sum_I w(x - x_I), \quad B_I = w(x - x_I) \tag{9.96}$$

Therefore, the MLS approximation becomes the Shepard approximation which is given by

$$u^h(x) = \frac{\sum_{I=1}^n w(x - x_I)}{\sum_{I=1}^n w(x - x_I)} u_I \tag{9.97}$$

which can reproduce exactly a constant function i.e., if $u_I = c$ then $u^h(x) = c$.

If we recall the projection of the particle velocity to the grid nodes in the standard MPM

$$v_I = \frac{\sum_p N_I(x_p)m_p v_p}{\sum_p N_I(x_p)m_p} = \frac{\sum_p N_I(x_p)m_p}{\sum_p N_I(x_p)m_p} v_p \tag{9.98}$$

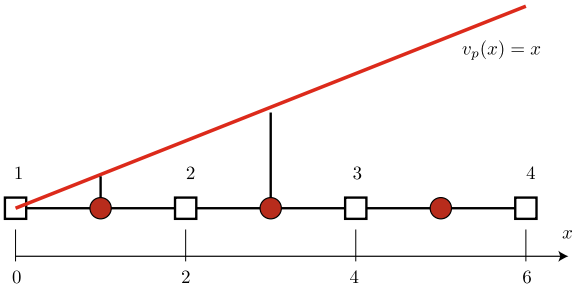


Fig. 9.9 Error in projecting particle velocities to grid nodes. With particles located at cell centers, the internal nodes have correct projected velocities while the boundary nodes do not (de Vaucorbeil et al. 2020)

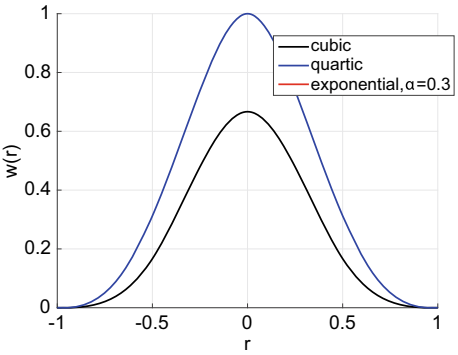


Fig. 9.10 Visualization of some commonly used weight functions

which is similar to the Shepard approximation. That is why only a constant particle velocity field can be exactly projected to the grid using the MPM. Therefore, the standard MPM is not able to provide an exact projection of a linear velocity field for arbitrary particle positions. To demonstrate this, we consider a grid of three equally spaced cells ($h = 2$) with 3 particles located at the cell centers. Assume that all the particles have a mass of unity and the particle velocity field is linear i.e., $v(x) = x$, cf. Figure 9.9. Obviously, Eq. (9.98) results in $v_1 = 1$ (correct value is 0) and $v_4 = 5$ (correct value is 6). The situation is getting worse with off-center particles.

Weight functions. Some commonly-used weight functions are (see Fig. 9.10)

- the cubic spline weight function

$$w(r) = \begin{cases} 2/3 - 4r^2 + 4r^3 & r \leq 1/2 \\ 4/3 - 4r + 4r^2 - 4/3r^3 & 1/2 < r \leq 1 \\ 0 & r > 1 \end{cases} \tag{9.99}$$

- the quartic spline weight function

$$w(r) = \begin{cases} 1 - 6r^2 + 8r^3 - 3r^4 & r \leq 1 \\ 0 & r > 1 \end{cases} \tag{9.100}$$

- the exponential weight function

$$w(r) = \begin{cases} e^{-(r/\alpha)^2} & r \leq 1 \\ 0 & r > 1 \end{cases} \tag{9.101}$$

where α is a constant.

with

$$r = \frac{|x_I - x|}{d_I} \tag{9.102}$$

where d_I is the support size of node I.

Extension of the MLS to higher dimensions is straightforward. For example, in 2D the linear basis is $\mathbf{p} = [1 \; x \; y]^{\mathrm{T}}$ and thus the moment matrix is given by

$$\mathbf{A} = \sum_{I=1}^{n} w(\mathbf{x} - \mathbf{x}_I) \begin{bmatrix} 1 & x_I & y_I \\ x_I & x_I^2 & x_I y_I \\ y_I & x_I y_I & y_I^2 \end{bmatrix} \tag{9.103}$$

and due to the compact support of the sum $\sum_I$ operates only over the points within the circle centered at $\mathbf{x}$ of radius being the smooth length. The argument to the weight functions is now written as $r = ||\mathbf{x}_I - \mathbf{x}|| / d_I$ where $||\cdot||$ is the usual Euclidean norm. Note that we are using a circular domain of influence.

Examples. We test the MLS approximations with constant and linear basis (i.e., a Shepard approximation and a MLS with $\mathbf{p} = [1 \; x]^{\mathrm{T}}$). A unit interval is discretized by 11 equally spaced grid nodes (mesh size is thus $h = 1/10$). Fifteen sampling points are randomly distributed on the interval, denoted by x_p with $p = 1, 2, \ldots, 15$. A linear function $u(x) = 1 + x$ is considered. Thus, at the sampling point x_p we have $u_p = 1 + x_p$. The purpose is to use MLS approximation to determine the functions at the grid nodes i.e., u_I with $I = 1, 2, \ldots, 11$. We use the cubic spline given in Eq. (9.99) with $d_I = 2h$. The M-file for this example is **mlsExamples1D.m**. The result show in Fig. 9.11 indicates that a the MLS with linear basis is able to exactly reproduce a linear function but the Shepard approximation is not.

Next we study the convergence property of the MLS. To this end, we consider the function $\sin(\pi x)$ on a unit interval. This interval is discretized by 20, 40, 80 and 160 elements. For each element, two material points are randomly distributed so that we have a distribution of material points that mimic a real MPM simulation. Results are depicted in Fig. 9.12 which confirms the second order accuracy of the linear MLS. The M-file for this test is **mlsConvergenceTest1D.m**.

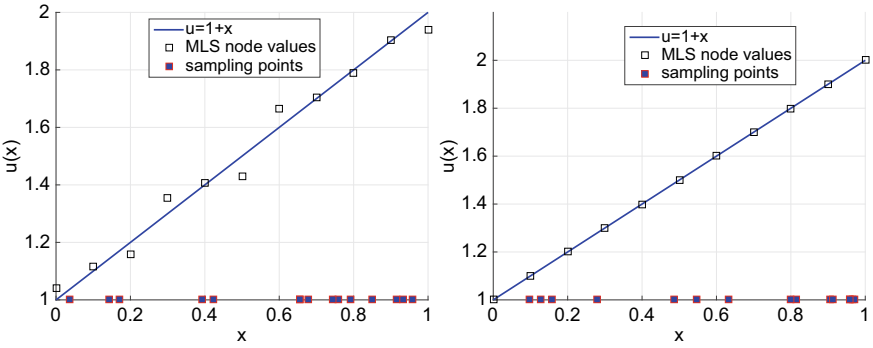


Fig. 9.11 MLS approximation of a linear function with Shepard function (left) and linear function $\mathbf{p} = [1 \;\; x]^{\mathrm{T}}$ (right)

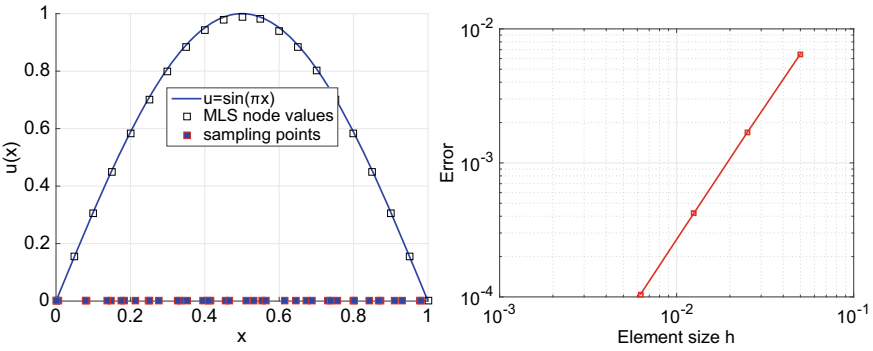


Fig. 9.12 MLS approximation of $\sin(\pi x)$ with linear basis $\mathbf{p} = [1 \; x]^{\mathrm{T}}$. Second order convergence verified with sampling points unequally spaced within the grid elements

9.3.2 *Velocity Projection*

From the particle velocities one can obtain the grid node velocities as follows

$$\mathbf{v}_I = \sum_p \Phi_I(\mathbf{x}_p)\mathbf{v}_p \tag{9.104}$$

where the sum is over the particles within the domain of influence of node I.

9.3.3 *One Point Quadrature*

The nodal mass and nodal forces (internal and external) are numerically computed using only one quadrature point–the element center $\mathbf{x}_c$. Therefore, we have

$$
\begin{aligned}
m_I &\approx \sum_{c=1}^{nc} \rho_c N_I(\mathbf{x}_c) V_c \\
\mathbf{f}_I^{\text{int}} &\approx -\sum_{c=1}^{nc} \boldsymbol{\sigma}_c \nabla N_I(\mathbf{x}_c) V_c
\end{aligned}
\tag{9.105}
$$

where V_c denotes the volume of the element, $V_c = \Delta x \Delta y$ for 2D; and nc is the number of elements surrounding node I which, for interior nodes, is $nc = 2$ in 1D, $nc = 4$ in 2D and $nc = 8$ in 3D. It should be emphasized that using one single quadrature point might not be sufficient. Either hourglass control can be added or full quadrature can be employed. In any case the quadrature error still exists as the boundary of the material domain is not accurately represented in the MPM. In this regard, the CPDI is the best among various MPM formulations.

The density and stresses at the element center are computed using a MLS approximation:

$$
\begin{aligned}
\rho_c &= \sum_p \Phi_p(\mathbf{x}_c) \rho_p \\
\boldsymbol{\sigma}_c &= \sum_p \Phi_p(\mathbf{x}_c) \boldsymbol{\sigma}_p
\end{aligned}
\tag{9.106}
$$

where the sum is over the particles within the domain of influence of the element center $\mathbf{x}_c$.

Remark 47 Using one-point quadrature rule improves the numerical integration error and eliminates the cell-crossing error (as the weighting gradients are always evaluated at the cell centers). Yet it is not accurate for cells on the solid boundary. A mixed quadrature was presented in Song et al. (2019) where for fully filled cells, one-point quadrature is used (similar to iMPM) and for partially filled cells, material point based integration is used. Partially filled cells are cells intersecting with the solid boundaries.

Remark 48 Mixed quadrature was used for the first time, to the best of our knowledge, in the geo-technical engineering community (Beuth et al. 2011). The motivation was due to the fact that particle based quadrature resulted in oscillation in the stress field. Even though this oscillation was due to cell crossing (Bardenhagen and Kober 2004) which can be removed using C^1 weighting functions, constructing such smooth functions is not easy with an unstructured grid.

9.3.4 Implementation

In this section we present the implementation of the MLS-MPM. The complete algorithm is given in Algorithm 18 which requires some modifications of a standard

MPM code only. First, we introduce two data structures to store the cell-centered density and stress. Second, in the step 'particles to grid', we proceed in two sub-steps: (1) computation of nodal mass and forces by looping over the elements and then over the centers and (2) computation of the nodal velocities by sweeping over the grid nodes. Third, in the update particle step, we construct the cell-centered density and stress from the newly updated particle density and stress. There is one change to the initialization step as well–one needs to initialize the cell-centered density and stresses in addition to conventional particle data.

Algorithm 18 Moving least square MPM: steps from t to $t + \Delta t$.

Mapping from particles to nodes
  Compute nodal mass $m_I^t = \sum_c N_I(\mathbf{x}_c^t)\rho_c^t V_c$
  Compute internal force $\mathbf{f}_I^{\text{int},t} = -\sum_c V_c \boldsymbol{\sigma}_c^t \nabla N_I(\mathbf{x}_c^t)$
  Compute nodal force $\mathbf{f}_I^t = \mathbf{f}_I^{\text{ext},t} + \mathbf{f}_I^{\text{int},t}$
  Compute nodal velocities $\mathbf{v}_I^t = \sum_p \Phi_I(\mathbf{x}_p)\mathbf{v}_p^t$
end
Update velocities $\mathbf{v}_I^{t+\Delta t} = \mathbf{v}_I^t + \left(\mathbf{f}_I^t/m_I^t\right)\Delta t$
Fix Dirichlet nodes
Update particles + construct cell-centered density/stress
  Update particle velocities $\mathbf{v}_p^{t+\Delta t} = \mathbf{v}_p^t + \Delta t \sum_I N_I(\mathbf{x}_p^t)\mathbf{f}_I^t/m_I^t$
  Update particle positions $\mathbf{x}_p^{t+\Delta t} = \mathbf{x}_p^t + \Delta t \sum_I N_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t}$
  Compute velocity gradient $\mathbf{L}_p^{t+\Delta t} = \sum_I \nabla N_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t}$
  Updated gradient deformation tensor $\mathbf{F}_p^{t+\Delta t} = (\mathbf{I} + \mathbf{L}_p^{t+\Delta t}\Delta t)\mathbf{F}_p^t$
  Update volume $V_p^{t+\Delta t} = \det \mathbf{F}_p^{t+\Delta t} V_p^0$.
  Update stresses $\boldsymbol{\sigma}_p^{t+\Delta t} = \boldsymbol{\sigma}_p^t + \Delta\boldsymbol{\sigma}_p$
  Update density $\rho_p^{t+\Delta t} = \rho_p^0/J$
  Construct cell-centered density $\rho_c^{t+\Delta t} = \sum_p \Phi_p(\mathbf{x}_c)\rho_p^{t+\Delta t}$
  Construct cell-centered stress $\boldsymbol{\sigma}_c^{t+\Delta t} = \sum_p \Phi_p(\mathbf{x}_c)\boldsymbol{\sigma}_p^{t+\Delta t}$
end

In order to make it general the MLS approximation for the velocity can be different from the MLS approximation used for the density/stress. Matlab code for MLS approximation of the node velocities and cell-centered density/stress are given in Listings 9.1 and 9.2, respectively. As can be observed the first involves a loop over the grid nodes while the latter is achieved by sweeping over the elements (of the background grid). It should be emphasized that not all nodes and elements are used in the reconstruction of the grid velocities and cell-centered density/stress. This is because there are inactive nodes and elements (these concepts have been introduced in Chap. 5). Only active nodes/elements, demonstrated in Fig. 9.13, are involved in the MLS reconstruction. The referred figure also points out the quadrature error of MPM–the domain boundary is not exactly captured.

Listing 9.1 MLS approximation for grid node velocities

```
shape = 'circle' ;          % shape of domain of influence
dmax1 = 2.0 ;               % smoothing length=dmax*nodal spacing (velo)
dmax2 = 2.0 ;               % smoothing length for cell-centered quantities
form  = 'cubic_spline' ;    % using cubic spline weight function
for ii=1:length(activeNodes)
  i         = activeNodes(ii);
  pt        = nodes(i);
  index     = defineSupport(xp,pt,di1);
  phi       = mlsLinearBasis1D(pt,index,xp,di1,form);
  nvelo0(i) = nvelo0(i) + dot(phi,vp(index));
end
```

Listing 9.2 MLS approximation for cell-centered density/stress

```
cellDensity = zeros(elemCount,1);  % cell-centerd density
cellStress  = zeros(elemCount,1);  % cell-centerd stress
% project particle density/stress to grid centers (MLS)
cellDensity(:) = 0;
cellStress(:)  = 0;
for ie=1:length(activeElems)
  e             = activeElems(ie);
  esctr         = elements(e,:);
  enode         = nodes(esctr);
  xc            = mean(enode);
  index         = defineSupport(xp,xc,di2);
  phi           = mlsLinearBasis1D(xc,index,xp,di2,form);
  cellDensity(e) = cellDensity(e) + dot(phi,rhop(index));
  cellStress(e)  = cellStress(e)  + dot(phi,s(index));
end
```

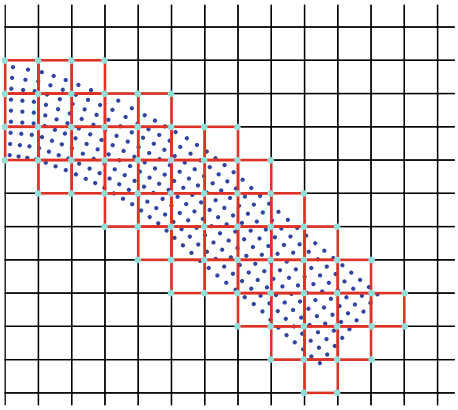


Fig. 9.13 Active nodes and elements: blue dots denote the particles (material points), cyan dots represent the active nodes and red squares the active elements. This is a simulation of the vibration of a soft cantilever beam

9.3.5 Improved Implementation

As MLS approximation involves inverse of the moment matrix $\mathbf{A}$ which is a 3×3 matrix for a 2D linear basis. This matrix inversion is repeated for all grid nodes and cell centers for every time step. After having worked out a working implementation of the MLS MPM, we have to improve it now to reduce the computational cost. As it is not efficient to run intensive computations with a Matlab code we did not attempt to fully optimize our code. The aim is to (1) have a sufficiently fast MLS MPM code to test its performance and (2) to illustrate some useful Matlab tools for speeding up a numerical code. To this end, we employ a two-step process. In the first step, the Matlab profiler is used to detect where the bottle neck of the code is and in the second step the Matlab Coder is used to convert this time consuming function to a MEX file.

Listing 9.3 Using a profiler to measure where a program spends time

```
profile on
impmMMS2D
profile viewer
```

Listing 9.3 demonstrates how to use the Matlab profiler to measure where the MPM (with MLS) program spends time. And as we have expected, the MLS approximation is the culprit. For a test it constitutes 60% run time.

Another technique to speed up the MLS is the improved MLS, developed by Liew et al. (2005) where orthogonal basis are used leading to a diagonal moment matrix which is trivial for inverting. Tran et al. (2019) employed the improved MLS in the MLS MPM. We have not coded this in our Matlab code.

9.4 The Affine Particle in Cell (APIC)

9.4.1 The Gradient Enhancement Technique

As demonstrated in Sect. 9.3.1, the standard MPM cannot exactly project a linear velocity field from the particle to the grid. Wallstedt and Guilkey (2007) presented analyses similar to what we have done in Sect. 9.3.1 for both MPM and GIMP. They found that, while the error does not decrease monotonically with increasing PPC (particles per cell), the error displays a downward trend. The GIMP basis functions perform better in general.

To minimize the error of the velocity projection from particle to grid, Wallstedt and Guilkey (2007) suggested to enhance the particle's velocities using the already available velocity gradient. In 1D, the particle to grid projection was therefore defined as, see Fig. 9.14 for an illustration

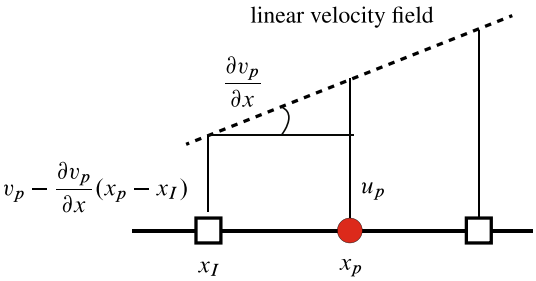


Fig. 9.14 Projection of a linear velocity field using the velocity gradient

$$m_I v_I = \sum_p m_p \phi_I(x_p) \left[v_p - \frac{\partial v_p}{\partial x}(x_p - x_I) \right] \tag{9.107}$$

Extension to 2D is straightforward and Wallstedt and Guilkey (2007) provided the formula but did not present 2D analyses

$$m_I v_{xI} = \sum_p m_p \phi_I(x_p) \left[v_{xp} - \frac{\partial v_{xp}}{\partial x}(x_p - x_I) - \frac{\partial v_{xp}}{\partial y}(y_p - y_I) \right] \tag{9.108}$$

$$m_I v_{yI} = \sum_p m_p \phi_I(y_p) \left[v_{yp} - \frac{\partial v_{yp}}{\partial x}(x_p - x_I) - \frac{\partial v_{yp}}{\partial y}(y_p - y_I) \right] \tag{9.109}$$

Written compactly, the above becomes

$$m_I \mathbf{v}_I = \sum_p m_p \phi_I(\mathbf{x}_p) \left[\mathbf{v}_p + \mathbf{L}(\mathbf{x}_I - \mathbf{x}_p) \right] \tag{9.110}$$

where $\mathbf{L}$ is the velocity gradient.

Pursuing a similar idea, Jiang et al. (2015) in the computer graphics community presented the Affine Particle-In-Cell (APIC) method. The APIC particle velocity to grid projection is given by

$$m_I^t \mathbf{v}_I^t = \sum_p m_p \phi_I(\mathbf{x}_p^t) \left[\mathbf{v}_p^t + \mathbf{B}_p^{t+\Delta t} (\mathbf{W}_p^t)^{-1} (\mathbf{x}_I - \mathbf{x}_p^t) \right] \tag{9.111}$$

where

$$\mathbf{B}_p^{t+\Delta t} = \sum_I \phi_I(\mathbf{x}_p^t) \mathbf{v}_I^{t+\Delta t} (\mathbf{x}_I - \mathbf{x}_p^t)^{\mathrm{T}}, \quad \mathbf{W}_p^t = \sum_I \phi_I(\mathbf{x}_p^t) (\mathbf{x}_I - \mathbf{x}_p^t)(\mathbf{x}_I - \mathbf{x}_p^t)^{\mathrm{T}} \tag{9.112}$$

Jiang et al. (2015) proved that APIC conserves angular momentum. They used this APIC with quadratic and cubic splines.

Remark 49 Without proof, Jiang et al. (2015) stated that $\mathbf{W}_p = {}^1/_4 h^2 \mathbf{I}$ for quadratic splines and $\mathbf{W}_p = {}^1/_3 h^2 \mathbf{I}$ for quadratic splines. Actually, they are just approximations. We have numerically verified that these simplified expressions for $\mathbf{W}_p$ are incorrect for hat functions, cubic B-splines and Bernstein functions. However, $\mathbf{W}_p$ is always a diagonal matrix.

9.4.2 Derivation

To obtain Eq. (9.111) we follow the derivation proposed by Müller et al. (2004). Assuming that the velocity field is $\mathbf{v} = [u \ v \ w]^\mathrm{T}$ and using a first order Taylor approximation, the x-component of the velocity at node I can be approximated as

$$\tilde{u}_I = u_p + \nabla u|_{\mathbf{x}_p} \cdot \mathbf{x}_{Ip} \tag{9.113}$$

where $\nabla u|_{\mathbf{x}_p} = [u_{,x} \ u_{,y} \ u_{,z}]^\mathrm{T}$ at particle p, and $\mathbf{x}_{Ip} = \mathbf{x}_I - \mathbf{x}_p$. The error of this approximation is measured as the sum of the squared differences between the approximated value $\tilde{u}_I$ and their known value u_I weighted by the shape function:

$$e = \sum_I \phi_I(\mathbf{x}_p)(\tilde{u}_I - u_I)^2 \tag{9.114}$$

Substituting Eq. (9.113) into Eq. (9.114) and expending yields $e = \sum_I \phi_p(\mathbf{x}_I)(u_p + u_{,x}\, x_{Ip} + u_{,y}\, y_{Ip} + u_{,z}\, z_{Ip} - u_I)^2$, where x_{Ip}, y_{Ip}, and z_{Ip} are the x, y and z-components of $\mathbf{x}_{Ip}$, respectively. Knowing the positions and velocities of the particles and the nodes, let's find $u_{,x}$, $u_{,y}$, and $u_{,z}$ that minimize the error e. Taking the respective derivative of e with respect to $u_{,x}$, $u_{,y}$, and $u_{,z}$, and setting them to 0 yields a system of three equations with three unknowns (Müller et al. 2004)

$$\left(\sum_I \phi_I(\mathbf{x}_p)\mathbf{x}_{Ip}\mathbf{x}_{Ip}^\mathrm{T}\right) \nabla u|_{\mathbf{x}_p} = \sum_I (u_p - u_i)\mathbf{x}_{Ip}\phi_I(\mathbf{x}_p) \tag{9.115}$$

Applying the same derivation to the y and z-components of the velocity (v and w) and adding the resulting two equations to the system given in Eq. (9.115) gives:

$$\begin{aligned}\left(\frac{\partial \mathbf{v}}{\partial \mathbf{x}}\bigg|_{\mathbf{x}_p}\right)^\mathrm{T} &= \mathbf{W}_p^{-1} \sum_I \phi_I(\mathbf{x}_p)\mathbf{x}_{Ip}(\mathbf{v}_p - \mathbf{v}_I)^\mathrm{T} \\ &= \mathbf{W}_p^{-1}\left(\sum_I \phi_I(\mathbf{x}_p)(\mathbf{x}_p - \mathbf{x}_I)\right)(\mathbf{v}_p)^\mathrm{T} - \mathbf{W}_p^{-1}\sum_I \phi_I(\mathbf{x}_p)(\mathbf{x}_p - \mathbf{x}_I)(\mathbf{v}_I)^\mathrm{T}\end{aligned} \tag{9.116}$$

where $\mathbf{W}_p$ given in Eq. (9.112).

Noting that $\sum_I \phi_p(\mathbf{x}_I)\mathbf{x}_I$ is the Shepard approximation of x_p, we get $\sum_I \phi_p(\mathbf{x}_I)(\mathbf{x}_p - \mathbf{x}_I) = 0$. Moreover, since $\mathbf{W}_p$ is symmetrical, Eq. (9.116) becomes:

$$\left.\frac{\partial \mathbf{v}}{\partial \mathbf{x}}\right|_{\mathbf{x}_p} = \sum_I \phi_I(\mathbf{x}_p)\mathbf{v}_I(\mathbf{x}_I - \mathbf{x}_p)^{\mathrm{T}}\mathbf{W}_p^{-1} = \mathbf{B}_p\mathbf{W}_p^{-1} \tag{9.117}$$

with $\mathbf{B}_p$ given in Eq. (9.112).

Coming back to the particle to grid projection, in APIC, it is therefore obtained as:

$$m_i\mathbf{v}_I = \sum_p m_p\phi_I(\mathbf{x}_p)\left[\mathbf{v}_p + \mathbf{B}_p\mathbf{W}_p^{-1}(\mathbf{x}_I - \mathbf{x}_p)\right] \tag{9.118}$$

Assuming a uniform velocity at the grid $\mathbf{c}$, $\mathbf{B}_p$ becomes identically zeros. To see that, assume a 2D case, then this matrix is written by

$$\mathbf{B}_p = \mathbf{c}\sum_I \phi_I(\mathbf{x}_p)(\mathbf{x}_{Ip})^{\mathrm{T}} = \mathbf{c}\sum_I \phi_I(\mathbf{x}_p)[x_I - x_p \;\; y_I - y_p] \tag{9.119}$$

We need to prove $\sum_I \phi_I(\mathbf{x}_p)(x_I - x_p) = \sum_I \phi_I(\mathbf{x}_p)(y_I - y_p) = 0$. This can be obtained using the PUM of the basis functions

$$\sum_I \phi_I(\mathbf{x}_p) = 1 \rightarrow \sum_I \phi_I(\mathbf{x}_p)x_p = x_p \tag{9.120}$$

And the following isoparametric

$$\sum_I \phi_I(\mathbf{x}_p)x_I = x_p \tag{9.121}$$

Then, one can obtain the following

$$\sum_I \phi_I(\mathbf{x}_p)(x_I - x_p) = \sum_I \phi_I(\mathbf{x}_p)x_I - \sum_I \phi_I(\mathbf{x}_p)x_p = x_p - \sum_I \phi_I(\mathbf{x}_p)x_p = 0 \tag{9.122}$$

9.4.3 Implementation

We present in Algorithm 19 a complete algorithm for the APIC. To avoid repetition, we have omitted some common steps with the MPM. We use the USL as APIC does not need to adopt the MUSL with $\alpha = 0$. Furthermore, the particle velocity update follows a PIC scheme i.e., $\alpha = 0$ in Eq. (2.54).

Algorithm 19 Solution procedure of explicit APIC (USL).

while $t < t_f$ **do**
  Reset grid quantities: $m_I^t = 0$, $(m\mathbf{v})_I^t = \mathbf{0}$, $\mathbf{f}_I^{\text{ext},t} = \mathbf{0}$, $\mathbf{f}_I^{\text{int},t} = \mathbf{0}$
  Mapping from particles to nodes (P2G)
    Compute nodal mass $m_I^t = \sum_p \phi_I(\mathbf{x}_p^t) m_p$
    Compute nodal momentum $(m\mathbf{v})_I^t = \sum_p m_p \phi_I(\mathbf{x}_p) \left[\mathbf{v}_p^t + \mathbf{L}_p^t(\mathbf{x}_I - \mathbf{x}_p^t)\right]$
    Compute internal force $\mathbf{f}_I^{\text{int},t} = -\sum_p V_p^t \boldsymbol{\sigma}_p^t \nabla\phi_I(\mathbf{x}_p^t)$
  end
  Update the momenta $(m\mathbf{v})_I^{t+\Delta t} = (m\mathbf{v})_I^t + \mathbf{f}_I^t \Delta t$
  Fix Dirichlet nodes I e.g. $(m\mathbf{v})_I^t = \mathbf{0}$ and $(m\mathbf{v})_I^{t+\Delta t} = \mathbf{0}$
  Update particles (G2P)
    Get nodal velocities $\mathbf{v}_I^t = (m\mathbf{v})_I^t / m_I^t$ and $\mathbf{v}_I^{t+\Delta t} = (m\mathbf{v})_I^{t+\Delta t} / m_I^t$
    Update particle velocities $\mathbf{v}_p^{t+\Delta t} = \sum_I \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t}$
    Compute $\mathbf{x}_{Ip} = \mathbf{x}_I - \mathbf{x}_p^t$
    Compute $\mathbf{W}_p^t = \sum_I \phi_I(\mathbf{x}_p^t)\mathbf{x}_{Ip}\mathbf{x}_{Ip}^{\mathrm{T}}$
    Compute $\mathbf{B}_p = \sum_I \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t}(\mathbf{x}_{Ip})^{\mathrm{T}}$
    Update particle positions $\mathbf{x}_p^{t+\Delta t} = \mathbf{x}_p^t + \Delta t \sum_I \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t}$
    Compute velocity gradient $\mathbf{L}_p^{t+\Delta t} = \mathbf{B}_p \mathbf{W}_p^{-1}$
    Updated gradient deformation tensor $\mathbf{F}_p^{t+\Delta t} = (\mathbf{I} + \mathbf{L}_p^{t+\Delta t}\Delta t)\mathbf{F}_p^t$
    Update volume $V_p^{t+\Delta t} = \det \mathbf{F}_p^{t+\Delta t} V_p^0$
    Compute the rate of deformation matrix $\mathbf{D}_p^{t+\Delta t} = 0.5(\mathbf{L}_p^{t+\Delta t} + (\mathbf{L}_p^{t+\Delta t})^{\mathrm{T}})$
    Compute the strain increment $\Delta\boldsymbol{\epsilon}_p = \Delta t \mathbf{D}_p^{t+\Delta t}$
  end
  Advance time $t = t + \Delta t$
  Error calculation: if needed (e.g. for convergence tests)
end while

9.4.4 Momenta Conservation

Linear momentum conservation. Similar to all other MPM variants, APIC does conserve exactly the linear momentum. The proof is as follows.

The total linear momentum of the grid nodes after the particle-to-grid projection is:

$$\sum_I m_I^t \mathbf{v}_I^t = \sum_I \sum_p m_p \phi_I(\mathbf{x}_p^t) \left[\mathbf{v}_p^t + \mathbf{B}_p^{t-\Delta t} \left(\mathbf{W}_p^{t-\Delta t}\right)^{-1} (\mathbf{x}_I - \mathbf{x}_p^t)\right] \tag{9.123}$$
$$= \sum_p m_p \left[\mathbf{v}_p^t \sum_I \phi_I(\mathbf{x}_p^t) + \mathbf{B}_p^{t-\Delta t} \left(\mathbf{W}_p^{t-\Delta t}\right)^{-1} (\sum_I \phi_I(\mathbf{x}_p^t)\mathbf{x}_I - \mathbf{x}_p^t \sum_I \phi_I(\mathbf{x}_p^t))\right]$$

Because of the partition of unity, $\sum_I \phi_I(\mathbf{x}_p^t) = 1$, and recalling that $\mathbf{x}_p = \sum_I \phi_I(\mathbf{x}_p^t)\mathbf{x}_I$, Eq. (9.123) becomes:

$$\sum_I m_I^t \mathbf{v}_I^t = \sum_p m_p \mathbf{v}_p^t \tag{9.124}$$

Therefore, the particle to grid step conserves exactly the linear momentum.

The total linear momentum of the particles expressed after the G2P (grid-to-particle) step is similar to the standard MPM using PIC velocity updates:

$$\begin{aligned}\sum_p m_p \mathbf{v}_p^{t+\Delta t} &= \sum_p m_p \sum_I \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t} \\ &= \sum_I \mathbf{v}_I^{t+\Delta t} \sum_p \phi_I(\mathbf{x}_p^t) m_p \\ &= \sum_I m_I^t \mathbf{v}_I^{t+\Delta t}\end{aligned} \tag{9.125}$$

Hence, the G2P also conserves exactly the linear momentum.

Angular momentum conservation. With APIC, the angular momentum that is lost at the end of the timestep during the transfer from particles to grid (G2P) is cleverly added through the affine approximation to the transfer of momentum from the particles to grid, but at the beginning of the time step. In other words, the loss of angular momentum at the end of the timestep is compensated for in advance. In the end of the day, this completely corrects this loss of momentum without altering the conservation of linear momentum. However, this has implications on the conservation of energy.

The angular momentum on the grid after the transfer from the particles to the grid is:

$$\mathbf{J}_h^t = \sum_I \mathbf{x}_I \times m_i v_i^t \tag{9.126}$$

$$= \sum_p \sum_I \mathbf{x}_I \times m_p \phi_p(\mathbf{x}_I)\left[\mathbf{v}_p + \mathbf{B}_p^t \mathbf{W}_p^{-1}(\mathbf{x}_I - \mathbf{x}_p^t)\right] \tag{9.127}$$

$$= \sum_p \sum_I \mathbf{x}_I \times m_p \phi_p(\mathbf{x}_I)\mathbf{v}_p + \sum_p \sum_I \mathbf{x}_I \times m_p \phi_p(\mathbf{x}_I)\mathbf{B}_p^t \mathbf{W}_p^{-1}\mathbf{x}_I - \sum_p \left[\sum_I \phi_p(\mathbf{x}_I)\mathbf{x}_I\right] \times m_p \mathbf{B}_p^t \mathbf{W}_p^{-1}\mathbf{x}_p^t$$

Note that in the following the permutation tensor $\boldsymbol{\epsilon}$ is going to be used. Moreover, to make these portions easier to read, the following convention is also adopted: for any matrix $\mathbf{A}$, the contraction $\mathbf{A} : \boldsymbol{\epsilon}$ means the same thing as $A_{\alpha\beta}\epsilon_{\alpha\beta\gamma}$. The manipulation $\mathbf{u} \times \mathbf{v} = (\mathbf{v}\mathbf{u}^{\mathrm{T}})^{\mathrm{T}} : \boldsymbol{\epsilon}$ is used to transition from a cross product into the permutation tensor.

Substituting Eqs. (9.9) and (9.10) and using the partition of unity ($\sum_I \phi_p(\mathbf{x}_I^t)\mathbf{x}_I = \mathbf{x}_p^t$), the above equation becomes:

$$
\begin{aligned}
\mathbf{J}_h^t &= \sum_p \mathbf{x}_p^t \times m_p \mathbf{v}_p^t + \sum_p m_p \sum_I \phi_p(\mathbf{x}_I)\mathbf{x}_I \times \left(\mathbf{B}_p^t \mathbf{W}_p^{-1} \mathbf{x}_I\right) - \sum_p m_p \mathbf{x}_p^t \times \left(\mathbf{B}_p^t \mathbf{W}_p^{-1} \mathbf{x}_p^t\right) \\
&= \sum_p \mathbf{x}_p^t \times m_p \mathbf{v}_p^t + \sum_p m_p \left(\sum_I \phi_p(\mathbf{x}_I)\mathbf{B}_p^t \mathbf{W}_p^{-1} \mathbf{x}_I (\mathbf{x}_I^t)^{\mathrm{T}}\right)^{\mathrm{T}} : \boldsymbol{\epsilon} - \sum_p m_p \left(\mathbf{B}_p^t \mathbf{W}_p^{-1} \mathbf{x}_p^t (\mathbf{x}_p^t)^{\mathrm{T}}\right)^{\mathrm{T}} : \boldsymbol{\epsilon} \\
&= \sum_p \mathbf{x}_p^t \times m_p \mathbf{v}_p^t + \sum_p m_p \left(\sum_I \phi_p(\mathbf{x}_I)\mathbf{B}_p^t \mathbf{W}_p^{-1} \mathbf{x}_I (\mathbf{x}_I^t)^{\mathrm{T}} - \mathbf{B}_p^t \mathbf{W}_p^{-1} \mathbf{x}_p^t (\mathbf{x}_p^t)^{\mathrm{T}}\right)^{\mathrm{T}} : \boldsymbol{\epsilon} \\
&= \sum_p \mathbf{x}_p^t \times m_p \mathbf{v}_p^t + \sum_p m_p \left(\mathbf{B}_p^t \mathbf{W}_p^{-1} \left(\sum_I \phi_p(\mathbf{x}_I)\mathbf{x}_I (\mathbf{x}_I^t)^{\mathrm{T}} - \mathbf{x}_p^t (\mathbf{x}_p^t)^{\mathrm{T}}\right)\right)^{\mathrm{T}} : \boldsymbol{\epsilon} \\
&= \sum_p \mathbf{x}_p^t \times m_p \mathbf{v}_p^t + \sum_p m_p \left(\mathbf{B}_p^t \mathbf{W}_p^{-1} \mathbf{W}_p\right)^{\mathrm{T}} : \boldsymbol{\epsilon} \\
&= \sum_p \mathbf{x}_p^t \times m_p \mathbf{v}_p^t + \sum_p m_p (\mathbf{B}_p^t)^{\mathrm{T}} : \boldsymbol{\epsilon} \\
&= \mathbf{J}_p^t + \sum_p m_p (\mathbf{B}_p^t)^{\mathrm{T}} : \boldsymbol{\epsilon}
\end{aligned}
\tag{9.128}
$$

One can see that the angular momentum is not preserved during that step due to the presence of the extra term $\sum_p m_p (\mathbf{B}_p^t)^{\mathrm{T}} : \boldsymbol{\epsilon}$.

The angular momentum on the particles at the end of the time step, i.e., after the grid to particle transfer (G2P), is:

$$
\mathbf{J}_p^{t+\Delta t} = \sum_p \mathbf{x}_p^{t+\Delta t} \times m_p \mathbf{v}_p^{t+\Delta t} \tag{9.129}
$$

Using the fact that $\sum_p \Delta t \mathbf{v}_p^{t+\Delta t} \times m_p \mathbf{v}_p^{t+\Delta t} = \mathbf{0}$

$$
\begin{aligned}
\mathbf{J}_p^{t+\Delta t} &= \sum_p \mathbf{x}_p^{t+\Delta t} \times m_p \mathbf{v}_p^{t+\Delta t} - \sum_p \Delta t \mathbf{v}_p^{t+\Delta t} \times m_p \mathbf{v}_p^{t+\Delta t} \\
&= \sum_p \mathbf{x}_p^{t+\Delta t} \times m_p \mathbf{v}_p^{t+\Delta t} - \sum_p (\mathbf{x}_p^{t+\Delta t} - \mathbf{x}_p^t) \times m_p \mathbf{v}_p^{t+\Delta t} \\
&= \sum_p \mathbf{x}_p^t \times m_p \mathbf{v}_p^{t+\Delta t} \\
&= \sum_I \mathbf{x}_I \times m_I \mathbf{v}_I^{t+\Delta t} - \sum_p m_p (\mathbf{B}_p^{t+\Delta t})^{\mathrm{T}} : \boldsymbol{\epsilon} \\
&= \mathbf{J}_h^{t+\Delta t} - \sum_p m_p (\mathbf{B}_p^{t+\Delta t})^{\mathrm{T}} : \boldsymbol{\epsilon}
\end{aligned}
$$

the change of angular momentum on the particles between the beginning and the end of the time step is:

$$\mathbf{J}_p^{t+\Delta t} - \mathbf{J}_p^t = \mathbf{J}_h^{t+\Delta t} - \sum_p m_p (\mathbf{B}_p^{t+\Delta t})^{\mathrm{T}} : \boldsymbol{\epsilon} - \mathbf{J}_h^t + \sum_p m_p (\mathbf{B}_p^t)^{\mathrm{T}} : \boldsymbol{\epsilon}$$
$$= \mathbf{J}_h^{t+\Delta t} - \mathbf{J}_h^t + \sum_p m_p (\mathbf{B}_p^t - \mathbf{B}_p^{t+\Delta t})^{\mathrm{T}} : \boldsymbol{\epsilon} \tag{9.130}$$

The change of angular momentum on the particles during a single time step is not exactly equal to the change on the grid: it differs by a quantity $\Delta \mathbf{J}_p - \Delta \mathbf{J}_h = \sum_p m_p (\mathbf{B}_p^t - \mathbf{B}_p^{t+\Delta t})^{\mathrm{T}} : \boldsymbol{\epsilon}$ that scales as $\mathcal{O}(\Delta t)$. Even though the norm of this quantity is small, the angular momentum is still not exactly preserved on the particles. However, it is interesting to check how the angular momentum changes (or lack thereof) on the grid between the end of a time step, and the beginning of a new one. To avoid any confusing, let's adopt a slightly different notation: $\mathbf{J}_h^{N,b}$ will be the angular momentum on the grid at the beginning of the time step N just after the particle to grid transfer, and $\mathbf{J}_h^{N,e}$ will be the same angular momentum but at the end of the time step N just before the grid to particle transfer, $\mathbf{J}_p^N$ will be the angular momentum on the particles at the beginning of time step N right before the particle to grid transfer but also the angular momentum on the particles at the end of the time step $N + 1$.

Equation (9.128) gives:

$$\mathbf{J}_h^{N,b} = \mathbf{J}_p^N + \sum_p m_p (\mathbf{B}_p^N)^{\mathrm{T}} : \boldsymbol{\epsilon} \tag{9.131}$$

and from Eq. (9.130), one can write:

$$\mathbf{J}_p^{N,b} = \mathbf{J}_h^{(N-1),e} - \sum_p m_p (\mathbf{B}_p^N)^{\mathrm{T}} : \boldsymbol{\epsilon} \tag{9.132}$$

Substituting Eq. (9.132) into Eq. (9.131), one gets:

$$\mathbf{J}_h^{N,b} - \mathbf{J}_h^{(N-1),e} = \mathbf{0} \tag{9.133}$$

So, even though the angular momentum is not conserved on the particles, it is conserved on the grid between time steps. This would result in a growing discrepancy between the two angular momenta over time.

In order to appreciate how $\Delta \mathbf{J}_p - \Delta \mathbf{J}_h$ evolves with time, let's look at the free rotation of a square body with a given initial angular momentum. A 4 m by 4 m square is left free to rotate for a total time of 3 s. The initial velocities of the particles forming the body are set such that at $t = 0$, the body has an angular velocity $\omega = 2\pi$ rad/s around its centre (see Fig. 9.15). The body makes one rotation per second. The material is Neo-Hookean, with a Young's modulus $E = 10^7$ Pa, Poisson's ratio $\nu = 0.3$, and density $\rho_0 = 1000$ kg/m^3. The angular momentum of the square is recorded and normalized with respect to the constant theoretical value $I\omega$ (I being

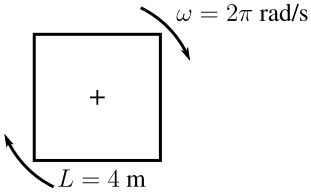


Fig. 9.15 Free body rotation: problem description

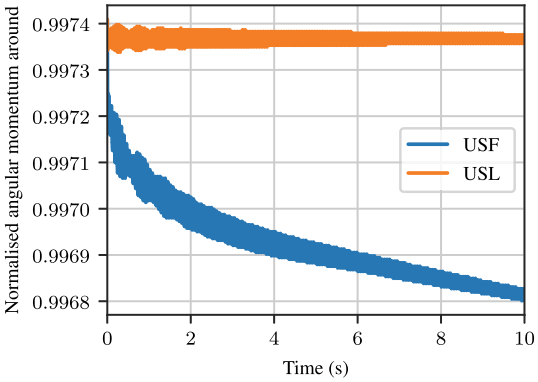


Fig. 9.16 Free body rotation: angular momentum as a function of time obtained using APIC ULMPM with cubic B-splines and 4 particles per cell. The angular momentum is normalized using its theoretical value: $I\omega$

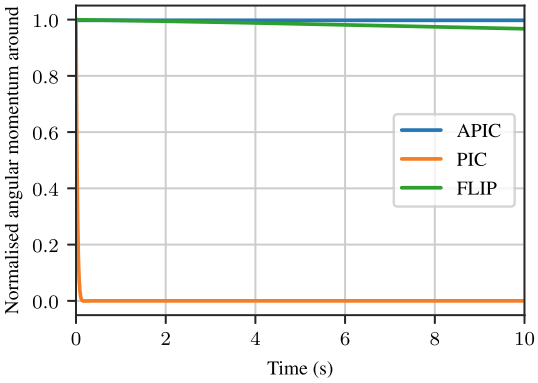


Fig. 9.17 Free body rotation: comparison of the angular momentum as a function of time obtained using APIC, PIC, and FLIP ULMPM with cubic B-splines and 4 particles per cell. The angular momentum is normalized using its theoretical value: $I\omega$

the second moment of inertia of the considered square). The cell size is $h = 1/40$ m or a grid of 40×40 cells is used.

The numerical values obtained using both the USF and USL schemes shown in Fig. 9.16 indicate that the change of angular momentum in that case is less than 0.1% which is negligible. Compared to the other variants PIC and FLIP, APIC is the type of MPM that best conserves the angular momentum as shown in Fig. 9.17.

9.4.5 Energy Conservation

The calculation of the change in total energy made in a single time step in the absence of heat transfer is similar to that performed in Sect. 9.1.3 in the case of PIC. This change, also referred to as error since if the energy were to be conserved remains:

$$\Delta E_{\text{error}} = -\Delta E_{\text{interpolation}} - \Delta E_{\text{algorithm}} \tag{9.134}$$

Compared to PIC, the error due to the algorithm, $\Delta E_{\text{algorithm}}$ is unchanged by the use of the affine particle-to-grid interpolation. However, the interpolation error becomes:

$$\begin{aligned}\Delta E_{\text{interpolation}} = \frac{1}{2}\Bigg[&\sum_{I,J} \mathbf{v}_I^{t+\Delta t} \cdot (\bar{M}_{IJ} - M_{IJ})\mathbf{v}_J^{t+\Delta t} + \sum_{I,J} \mathbf{v}_I^t \cdot (\bar{M}_{IJ} - M_{IJ})\mathbf{v}_J^t \\ &+ \sum_p m_p \left|\left| \mathbf{v}_p^t - \sum_I \phi_I(\mathbf{x}_p^t)\mathbf{v}_I^t \right|\right|^2 \Bigg] \\ &- \sum_p m_p \sum_I \phi_I(\mathbf{x}_p^t)(\mathbf{v}_I^t)^{\mathrm{T}} \mathbf{B}_p^{t+\Delta t}(\mathbf{W}_p^t)^{-1}(\mathbf{x}_I - \mathbf{x}_p^t)\end{aligned} \tag{9.135}$$

The total energy error increment for APIC is therefore:

$$(\Delta E_{\text{error}})_{\text{APIC}} = (\Delta E_{\text{error}})_{\text{PIC}} - \sum_p m_p \sum_I \phi_I(\mathbf{x}_p^t)(\mathbf{v}_I^t)^{\mathrm{T}} \mathbf{B}_p^{t+\Delta t}(\mathbf{W}_p^t)^{-1}(\mathbf{x}_I - \mathbf{x}_p^t) \tag{9.136}$$

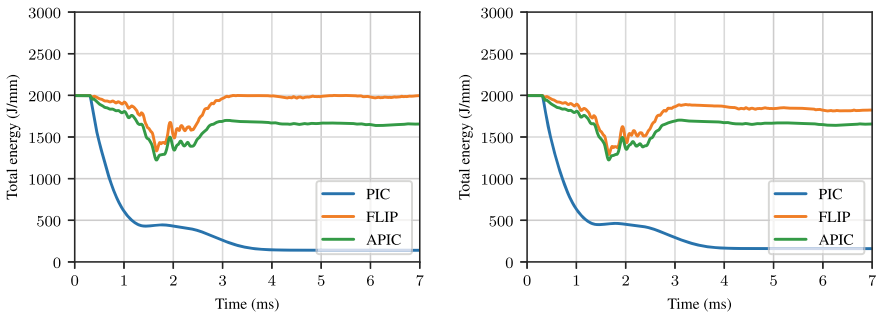
where $(\Delta E_{\text{error}})_{\text{PIC}}$ is the total error increment obtained with PIC given by either Eq. (9.39) or Eq. (9.48), when combined with USF and USL, respectively.

The sign of the second term in Eq. (9.136) is difficult to assess as it depends on the relative position of the particles and also the strain distribution within the solid. To get a better feel for how it affects the change in total energy, let's use the problem of the impact of two elastic bodies first introduced in Sect. 9.1.3 of which description is illustrated in Fig. 9.1.

Change of total energy during the impact of these two rings shown in Fig. 9.18 informs us that the extra term in the total error increment improves significantly the energy conservation of APIC compared to PIC. It brings the energy conservation of APIC close to that of FLIP. But, its contribution is not large enough to obtain better nor similar results as with FLIP. This is true when using both USF and USL.

9.5 Convergence Tests

We present two convergence tests in this section. As it is always easier to code and debug 1D problems, we consider first a 1D test (Sect. 9.5.1). Furthermore, 1D problems allow to use extremely fine meshes which reveal instability of some MPM



(a) USF (b) USL

Fig. 9.18 Change of total energy during the impact of two compressible Neo-Hookean rings obtained with ULMPM using cubic B-splines and 4 particles per cell

variants. Then, we study the generalized vortex problem (Sect. 9.5.2). The manufactured solutions and the corresponding body forces were presented in Sect. 9.2.

9.5.1 One Dimensional Convergence Test

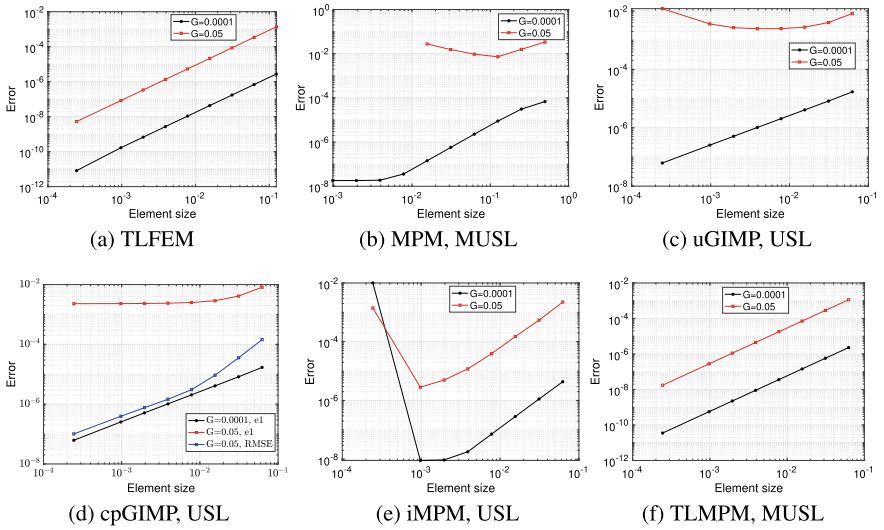
Let's consider a unit segment i.e., the spatial domain is $0 \leq X \leq 1$. The manufactured displacement field is assumed to be

$$u(X,t) = G\sin(\pi X)\sin(c\pi t) \tag{9.137}$$

where G is the maximum amplitude of the displacement; $c = \sqrt{E/\rho_0}$ and E denotes the Young's modulus. The period is thus given by $T = {}^{2\pi}/_{c\pi}$; X denotes the material coordinates i.e., coordinates in the reference configuration, and the time domain is $0 \leq t \leq T$ i.e., one period of oscillation is considered. The corresponding body force and boundary/initial conditions for this manufactured solution are given in Sect. 9.2.1.

This MMS verification test is done using the following material parameters: $E = 10^7$ Pa, $\nu = 0.3$, and $\rho_0 = 1000$ kg/m^3. Both small and large displacements are tested by setting the maximum amplitude displacement $G = 10^{-4}$ and $G = 0.05$, respectively. The same time step of $0.2h/c$, where h is the cell size, was used: this small time step eliminates any error due to time discretization. And note that herein we focus on spatial convergence only. We study the convergence of the FEM (see Chapter D for the formulation), MPM, TLMPM, GIMP and iMPM. Note that the CPDI performs identically to the cpGIMP for this problem and thus not discussed.

In the literature, for convergence tests, rather coarse meshes have been used, by many researchers, with which convergence was always obtained, but Gong (2015)



(a) TLFEM (b) MPM, MUSL (c) uGIMP, USL

(d) cpGIMP, USL (e) iMPM, USL (f) TLMPM, MUSL

Fig. 9.19 Axis-align unit segment test: convergence of FEM and various MPM variants using the un-normalized error measure e_1 given in Eq. (9.79) (de Vaucorbeil et al. 2020)

pointed out that when the mesh is very fine, many MPM variants exhibit divergence. That is why, herein we use very fine meshes to have a fair view of the convergence behavior of MPMs. Actually, grids of size $h = 2^{-k}$, $k = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12\}$, with the smallest cell size is 0.000244141, are considered.

The results are given in Fig. 9.19 where it can be seen that optimal convergence (of order 2) was obtained with the FEM and TLMPM. This is expected as there is no error in updating the stress for this manufactured solution (the constitutive model is a hyperelasticity). In the TLMPM, there is no quadrature error (for this axis-aligned problem) and cell-crossing instability. The TLMPM accuracy is certainly slightly lower than the FEM accuracy. The standard MPM does converge for the small deformation case up to a certain mesh level and the error plateaus. However, it does not converge at all for the large deformation case. The convergence is better with uGIMP even though the error increases for fine meshes (large deformation case). The cpGIMP does perform better when large deformation occurs but the convergence rate is far from optimal. The convergence of the iMPM is optimal for meshes with size smaller than 10^{-3} and all of a sudden, it diverges for more refined meshes. The reason of that is unclear at this stage (Gong 2015).

While performing this convergence test, we have realized that the convergence does depend on the error measure. To demonstrate this, considering the following root mean square error

$$e^{\text{RMSE}} = \sqrt{\frac{\sum_{t^n=0}^{t_f} \sum_{p=1}^{n_p} \left|\left|\mathbf{u}_p^h(t^n) - \mathbf{u}^{\text{exact}}(\mathbf{X}_p, t^n)\right|\right|^2}{n_T \times n_p}} \tag{9.138}$$

It is obvious that when the mesh gets finer, both n_p and n_T (the number of time steps) gets bigger and thus the error measure in Eq. (9.138) will always give convergence, cf. Figure 9.19d. We do not think this RMSE is objective for dynamics simulations because one can get good convergence simply by refining the time steps i.e., making n_T bigger.

9.5.2 Generalized Vortex Problem

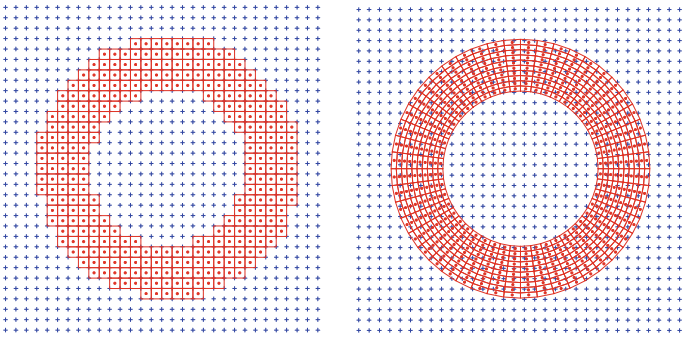
We consider the convergence of the MPM using the generalized vortex problem introduced in Sect. 9.2.3. This problem demonstrates that none of existing MPM variants converge when the grid is fine enough. The reason why we do not know.

Similarly to Kamojjala et al. (2015), the inner and outer radii are taken as 0.75 and 1.25 m, respectively. Material parameters are taken as: $E = 10^3$ Pa, $\rho_0 = 1000$ kg/m^3, $\nu = 0.3$, and the maximum displacement amplitude $G = 1$; $T_0 = 1$ s and the total simulation time is one second i.e., one period is considered. The velocity of all the nodes outside of the ring is set to zero, and the body forces (Eq. (9.76)) are applied directly on the nodes. Alternatively, these forces can be applied to the particles and mapped to the nodes. But we find it more efficient and accurate to directly compute the forces at the nodes.

We solve this problem using three MPM variants: the TLMPM (with linear and quadratic Bernstein functions), the boundary non-conforming CPDI-Q4 and the boundary conforming CPDI-Q4. The `Karamelo` code was used for these intensive convergence analyses. The particles' velocities is updated using a mix of PIC and FLIP with the mixing factor $\alpha = 0.99$. In the boundary non-conforming CPDI, the particles are generated using the same algorithm for the standard MPM and thus the particle domains do not fit the boundary, see Fig. 9.20a. The TLMPM has the same particle positions. On the other hand, in the conforming CPDI, the particle domains are generated using a mesh generator (we used Gmsh (Geuzaine and Remacle 2009)). The particle domains are conforming to the boundary, see Fig. 9.20b.

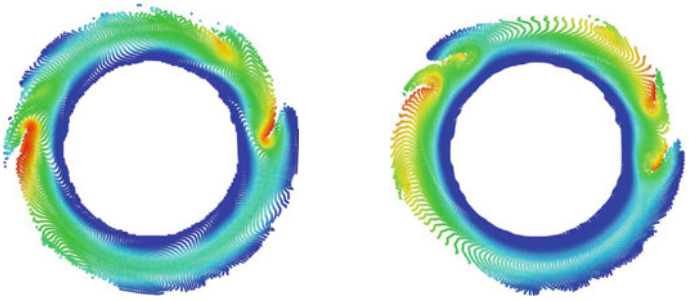
The results obtained with CPDI-Q4 are given in Fig. 9.21. The performance of CPDI-Q4 is much better than the standard MPM and uGIMP (not shown here), but the deformation is not well captured: the boundaries of the domain are not smooth and circular as the exact solution. The particle domains are so much distorted. For this reason, we do not perform mesh convergence for CPDI. The remaining of this section focuses on the TLMPM.

Figure 9.22 shows how the displacement errors e_1 and e_2 change according to the element size. It can be seen that at large element size, the convergence rate is quadratic, but decreases progressively as the cell size decreases. This is the sign of a competition between two errors: cell size related errors and mapping errors. As the cell size decreases, so does the error associated to it, while the mapping error which appeared negligible for large cell sizes becomes dominant. Thus the error plateaus. Note that there is another source of error—the mapping of particle momenta to the



(a) Non-conforming CPDI-Q4 (b) Conforming CPDI-Q4

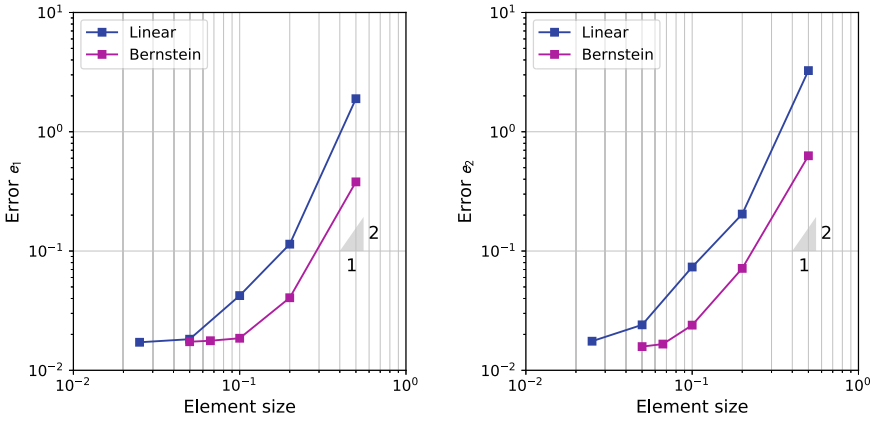
Fig. 9.20 Generalized vortex problem. Grid and initial particles used in CPDI. These images were created using the `PyPlot` graphical package (Johnson 2012)



(a) Non-conforming (b) Conforming

Fig. 9.21 Deformed configuration obtained with CPDI-Q4. The color represents the magnitude of the displacement field. Blue is zero and read is maximum. The grid cell size is 0.05 and there are about 14 000 particles. Images obtained using `Ovito` (Stukowski 2009)

grid as shown in Sulsky and Gong (2016). We did not implement those improvements to mitigate this error and leave it as a future work. However, the order of magnitude of this plateaus is so low that very good qualitative agreement exist between the deformed configurations at peak rotation angle as obtained with the TLMPM using Bernstein shape functions and the analytical solution (see Fig. 9.23).



(a) Non-normalized error plot (b) Normalized error plot

Fig. 9.22 Plot of the displacement error as a function of the background mesh refinement obtained using the TLMPM. Note that as e_2 is normalized (de Vaucorbeil et al. 2020)

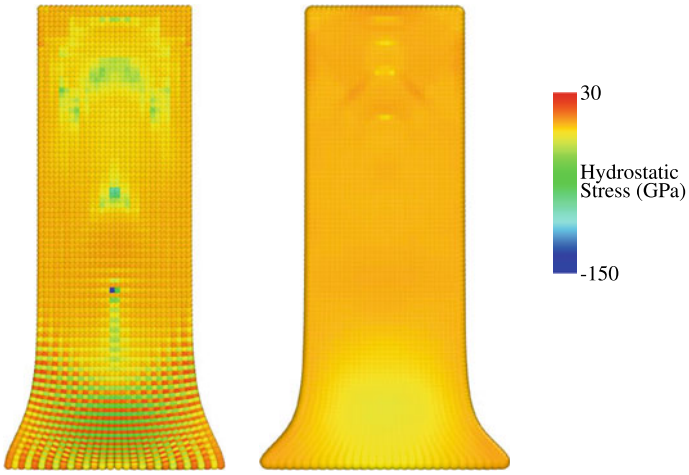
(a) analytical solution (b) TLMPM solution

Fig. 9.23 Deformed configuration at $t = 0.5\ s$ obtained **a** analytically and **b** with the TLMPM using quadratic Bernstein polynomials shape functions and a cell size of 0.033 (de Vaucorbeil et al. 2020)

9.6 Volumetric Locking

When Poisson's ratio approaches 0.5 or when a plastic flow is constrained by the volume conservation condition, well known overly stiff numerical solutions appear. This problem is known as volumetric locking. It is associated with a rather large (in magnitude) and chaotic state of stress as shown in Fig. 9.24a. The severity of volumetric locking increases when low order shape functions are employed in the standard MPM.

Various solutions to this problem have been proposed, all borrowing methods developed for the FEM (Love and Sulsky 2006b, a; Mast et al. 2012; Yang et al. 2018;



(a) Full integration (b) Reduced integration

Fig. 9.24 Taylor bar impact simulated with TLMPM and linear shape functions showing the difference in the hydrostatic stress when using either **a** full integration or **b** reduced integration where this problem is not present. When using full integration, the hydrostatic stress field is chaotic and its magnitude is much larger than with reduced integration. This is a typical illustration of the effect of volumetric locking

Coombs et al. 2018; Iaconeta et al. 2019). In particular, Love and Sulsky (2006b); Mast et al. (2012) adopted the Hu–Washizu multi-field variational principle which introduces independent approximations for the volumetric and the deviatoric components of the strain and stress fields. On the other hand, a $\mathbf{u} - p$ mixed formulation was used in Iaconeta et al. (2019). All these works employ the standard MPM which is known for its poor accuracy due to, among others, cell-crossing issue. Coombs et al. (2018) applied the F-bar method, developed for the FEM (de Souza Neto et al. 1996; Neto et al. 2005) to quasi-static MPM and GIMP.

This section discusses the F-bar method in the context of the MPM. We start with a brief introduction to the F-bar method in Sect. 9.6.1. Then, how the F-bar method can be implemented in the MPM is given. In Sect. 9.6.2, we present the first implementation that we refer to as cell averaging F-bar. And in Sect. 9.6.3 we present the other scheme that we name nodal averaging F-bar method.

9.6.1 Overview of the F-bar Method

In FEM, one of the ways to mitigate this problem is to use reduced integration (with high order elements to prevent hour-glassing). This also works when using TLMPM (see Fig. 9.24b). However, this simple trick does not apply for ULMPM for we cannot control how many particles are present in a given cell. In that case, the simple F-

bar method can be adopted. The F-bar technique uses an effective combination of reduced integration for the computation of the volumetric part of the deformation tensor (equivalent to the particles volume) and full integration for its isochoric part.

The isochoric/volumetric split of the deformation gradient is defined as

$$\mathbf{F} = \mathbf{F}_{\text{iso}}\mathbf{F}_{\text{vol}} \tag{9.139}$$

where the isochoric and volumetric parts are given by

$$\mathbf{F}_{\text{iso}} = \frac{\mathbf{F}}{(\det \mathbf{F})^{1/3}}, \quad \mathbf{F}_{\text{vol}} = (\det \mathbf{F})^{1/3}\mathbf{I} \tag{9.140}$$

Note that $\det \mathbf{F}_{\text{iso}} = [(\det \mathbf{F})^{-1/3}]^3 \det \mathbf{F} = 1$, thus justify the term 'isochoric'.

In the F-bar method, one defines the following modified gradient deformation tensor (Neto et al. 2005)

$$\bar{\mathbf{F}} = \left(\frac{\det \mathbf{F}_0}{\det \mathbf{F}}\right)^{1/3} \mathbf{F} \tag{9.141}$$

where $\mathbf{F}_0$ is the gradient deformation tensor evaluated at the centroid of the finite element. It can be shown that the isochoric/volumetric parts of the modified deformation tensor are given by

$$\begin{aligned} \bar{\mathbf{F}}_{\text{iso}} &= (\det \mathbf{F})^{-1/3}\mathbf{F} = \mathbf{F}_{\text{iso}} \\ \bar{\mathbf{F}}_{\text{vol}} &= (\det \mathbf{F}_0)^{1/3}\mathbf{I} = \mathbf{F}_{0,\text{vol}} \end{aligned} \tag{9.142}$$

i.e. the isochoric component of $\bar{\mathbf{F}}$ coincides with the current (integration point) isochoric deformation gradient while its volumetric part corresponds to the dilatation at the centroid of the element. Now, in order to compute the stresses at a particular integration point one uses the modified gradient deformation tensor rather than the original one. For nearly incompressible materials, the pressure variable is often decoupled from the stress and hence the pressure is a function of $\det \bar{\mathbf{F}}$ and thus constant within a cell using the F-bar method.

9.6.2 F-bar Method in MPM: Cell Averaging

In the MPM, the F-bar method was first implemented in the Uintah MPM code, where it is called *pressure stabilization*, long before the published work of Coombs et al. (2018).

When the F-bar method is directly applied to the MPM, the deformation gradient at the element centroid is replaced by the cell-centered or cell-averaged deformation gradient computed according to

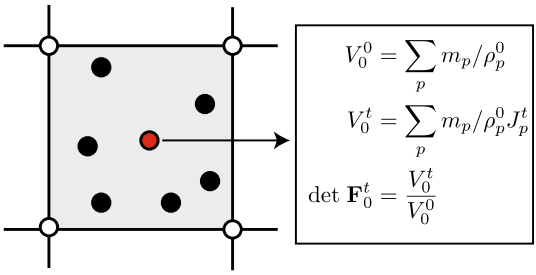


Fig. 9.25 F-bar method applied for the material point method

$$\det \mathbf{F}_0^t = J_0^t = \frac{V_0^t}{V_0^0} = \frac{\sum_p m_p/\rho_p^0 J_p^t}{\sum_p m_p/\rho_p^0} \tag{9.143}$$

i.e., $\det \mathbf{F}_0^t$ is defined as the ratio of the cell-centered current volume and the cell-centered initial volume. In the above equation, the sum is over all particles in the cell at hand, cf. Figure 9.25. For implementation, the procedure is presented in Algorithm 20. We refer to a related work (Moutsanidis et al. 2019) on this topic.

Algorithm 20 Algorithm for F-bar in an explicit MPM code.

1: **for** p=1:np **do**
2: Compute/retrieve the particle deformation gradient and $J_p = \det \mathbf{F}_p$
3: Get index of cell contains p, named c
4: $cellVol0(c) = cellVol0(c) + (m_p/\rho_p^0)$
5: $cellVol(c) = cellVol(c) + (m_p/\rho_p^0)J_p$
6: **end for**
7: Compute the centered gradient deformation determinant $cellJ = cellVol/cellVol0$
8: **for** p=1:np **do**
9: Retrieve the particle deformation gradient and $J_p = \det \mathbf{F}_p$
10: Get index of cell contains p, named c
11: Retrieve the cell-centered $J0 = cellJ(c)$
12: Compute modified gradient deformation $\bar{\mathbf{F}}_p = (J0/J_p)^{1/3}\mathbf{F}_p$
13: **end for**

9.6.3 F-bar Method in MPM: Nodal Averaging

There are several problem associate with the cell averaging F-bar technique. First, when using non-linear shape functions, the concept of cells in the MPM is blur. For instance, cubic B-spline shape functions spane over what would be 4 grid cells when using linear shape functions. Second, a list of particles per cell needs to be kept up-to-date.

The alternative to the F-bar method is to replace the current volume of all particles V_p^t by the interpolation of their nodal averaging $\tilde{V}_p^t$. This is done in three steps:

1. Mapping the particle's volume to the nodes:

$$V_I^t = \sum_p \phi_I(\mathbf{x}_p) V_p^t \tag{9.144}$$

2. Interpolation from the nodes back to the particles:

$$\tilde{V}_p^t = m_p^t \sum_I \phi_I(\mathbf{x}_p) \frac{V_I^t}{m_I^t} \tag{9.145}$$

3. Modification of the deformation gradient:

$$\bar{\mathbf{F}}_p = \left(\frac{\tilde{V}_p^t / V_P^0}{\det \mathbf{F}_p} \right)^{1/3} \mathbf{F}_p \tag{9.146}$$

It is paramount that the total volume is conserved during this averaging. The total volume on the nodes $V_{\text{tot,nodes}}$ is exactly the total volume of the particles $V_{\text{tot,particles}}$:

$$\begin{aligned} V_{\text{tot,nodes}} &= \sum_I V_I^t \\ &= \sum_I \sum_p \phi_I(\mathbf{x}_p) V_p^t \\ &= \sum_p V_p^t \sum_I \phi_I(\mathbf{x}_p) \\ &= \sum_p V_p^t \quad \text{(partition of unity)} \\ &= V_{\text{tot,particles}} \end{aligned} \tag{9.147}$$

This is pretty straight-forward.

Interestingly, both nodal and particle masses are present in the interpolation of the volume back to the particles (Eq. (9.145)). The only motivation for this is the conservation of the total volume. In the MPM, the quantity that is always conserved is the total linear momentum (Sect. 9.1.1). In the particle to grid step of the MPM algorithm, the particles momentum is mapped onto the nodes. In the grid to particle step, it is the nodal velocities that are projected, i.e., the ratio between the nodal momentum and the nodal mass. Now in the last two sentences, replace momentum by volume and you get equations Eqs. (9.144) and (9.145): first mapping of the particles volume onto the nodes, then projection of the ratio between the nodal volume the nodal mass.

(a) Without nodal averaging (b) With nodal averaging

Fig. 9.26 Taylor bar impact simulated with ULMPM and cubic B-spline shape functions showing the difference in the distribution of the positive and negative hydrostatic pressures without and with nodal averaging. In both cases, black and white colors correspond to negative and positive hydrostatic pressure, respectively. One can easily see that nodal averaging resolves the problem of chaotic pressure

The new total volume of the particles is exactly the same as the original volume:

$$\begin{aligned}\tilde{V}_{\text{tot,particles}} &= \sum_p \tilde{V}_p^t = \sum_p m_p \sum_I \phi_I(\mathbf{x}_p)\frac{V_I^t}{m_I^t} \\ &= \sum_I \frac{V_I^t}{m_I^t} \sum_p \phi_I(\mathbf{x}_p) m_p \\ &= \sum_I \frac{V_I^t}{m_I^t} m_I^t \\ &= \sum_I V_I^t = V_{\text{tot,nodes}} \\ &= V_{\text{tot,particles}} \quad (9.147)\end{aligned} \tag{9.148}$$

To demonstrate the performance of this nodal averaging scheme, we solve the well known Taylor bar problem using ULMPM with and without it. The results are shown in Fig. 9.26. One can see that without volume averaging, the sign of the hydrostatic pressure fluctuates heavily between adjacent particles; this is usually referred to as the checkerboard phenomenon. This is not the case when nodal averaging is used. There, the pressure field is way smoother.

Note that nodal averaging can be used for TLMPM. The only difference is that the shape functions are evaluated in the reference configuration, i.e., now $\phi_I(\mathbf{x}_p)$ is replaced by $\phi_I(\mathbf{X}_p)$.

References

Bardenhagen, S.G.: Energy conservation error in the material point method for solid mechanics. J. Comput. Phys. **180**(1), 383–403 (2002)

Bardenhagen, S.G., Kober, E.M.: The generalized interpolation material point method. Comput. Model. Eng. Sci. **5**(6), 477–495 (2004)

Berzins, M.: Nonlinear stability and time step selection for the MPM method. Comput. Part. Mech. **5**(4), 455–466 (2018)

Beuth, L., Wieckowski, Z., Vermeer, P.A.: Solution of quasi-static large-strain problems by the material point method. Int. J. Numer. Anal. Meth. Geomech. **35**(13), 1451–1465 (2011)

Brannon, R.M., Kamojjala, K., Sadeghirad, A.: Establishing credibility of particle methods through verification testing. In: Particle-Based Methods II—Fundamentals and Applications, pp. 685–696 (2011)

Ciarlet, P.G., Lions, J.L.: Handbook of Numerical Analysis. North-Holland, Amsterdam (1991)

Coombs, W.M., Charlton, T.J., Cortis, M., Augarde, C.E.: Overcoming volumetric locking in material point methods. Comput. Methods Appl. Mech. Eng. **333**, 1–21 (2018)

de Souza Neto, E. A., Perić, D., Dutko, M., Owen, D.: Design of simple low order finite elements for large strain analysis of nearly incompressible solids. Int. J. Solids Struct. **33**(20–22), 3277–3296 (1996)

de Vaucorbeil, A., Nguyen, V.P., Hutchinson, C.R.: A total-lagrangian material point method for solid mechanics problems involving large deformations. Comput. Methods Appl. Mech. Eng. **360**, 112783 (2020). https://doi.org/10.1016/j.cma.2019.112783

de Vaucorbeil, A., Nguyen, V.P., Sinaie, S., Wu, J. Y.: Chapter two—material point method after 25 years: theory, implementation, and applications. In: Advances in Applied Mechanics, vol. 53, pp. 185–398. Elsevier (2020)

de Vaucorbeil, A., Nguyen, V.P.: Modeling contacts with a total lagrangian matertial point method. Comput. Methods Appl. Mech. Eng. **360**, 112783 (2021). https://doi.org/10.1016/j.cma.2019.112783. Mar

de Souza Neto, E.A., Andrade Pires, F.M., Owen, D.R.J.: F-bar-based linear triangles and tetrahedra for finite strain analysis of nearly incompressible solids. part i: formulation and benchmarking. Int. J. Numer. Methods Eng. **62**(3), 353–383 (2005)

Geuzaine, C., Remacle, J.F.: Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. Int. J. Numer. Meth. Eng. **79**(11), 1309–1331 (2009)

Gong, M.: Improving the material point method. Ph.D. thesis, The University of New Mexico, Albuquerque (2015)

Gritton, C., Berzins, M.: Improving accuracy in the MPM method using a null space filter. Comput. Part. Mech. **4**(1), 131–142 (2017)

Hammerquist, C.C., Nairn, J.A.: A new method for material point method particle updates that reduces noise and enhances stability. Comput. Methods Appl. Mech. Eng. **318**, 724–738 (2017)

Huang, P., Zhang, X., Ma, S., Huang, X.: Contact algorithms for the material point method in impact and penetration simulation. Int. J. Numer. Meth. Eng. **85**(4), 498–517 (2011)

Iaconeta, I., Larese, A., Rossi, R., Oñate, E.: A stabilized mixed implicit material point method for non-linear incompressible solid mechanics. Comput. Mech. **63**(6), 1243–1260 (2019)

Jiang, C., Schroeder, C., Selle, A., Teran, J., Stomakhin, A.: The affine particle-in-cell method. ACM Trans. Graph. **34**(4), 51:1–51:10 (2015)

Johnson, S.G.: PyPlot module for Julia (2012). https://github.com/stevengj/PyPlot.jl

Kamojjala, K., Brannon, R., Sadeghirad, A., Guilkey, J.: Verification tests in solid mechanics. Eng. Comput. **31**(2), 193–213 (2015)

Knupp, P., Salari, K.: Verification of Computer Codes in Computational Science and Engineering. Chapman and Hall/CRC (2003)

Lancaster, G.M.: Surfaces generated by moving least squares methods. Math. Comput. **3**(37), 141–158 (1981)

Liew, K.M., Cheng, Y., Kitipornchai, S.: Boundary element-free method (BEFM) for two-dimensional elastodynamic analysis using Laplace transform. Int. J. Numer. Methods Eng. **64**(12), 1610–1627 (2005)

Love, E., Sulsky, D.L.: An energy-consistent material-point method for dynamic finite deformation plasticity. Int. J. Numer. Meth. Eng. **65**(10), 1608–1638 (2006a)

Love, E., Sulsky, D.L.: An unconditionally stable, energy-momentum consistent implementation of the material-point method. Comput. Methods Appl. Mech. Eng. **195**(33–36), 3903–3925 (2006b)

Mast, C.M., Mackenzie-Helnwein, P., Arduino, P., Miller, G.R., Shin, W.: Mitigating kinematic locking in the material point method. J. Comput. Phys. **231**(16), 5351–5373 (2012)

Moutsanidis, G., Koester, J.J., Tupek, M.R., Chen, J.S., Bazilevs, Y.: Treatment of near-incompressibility in meshfree and immersed-particle methods. Comput. Part. Mech. 1–19 (2019)

Müller, M., Keiser, R., Nealen, A., Pauly, M., Gross, M., Alexa, M.: Point based animation of elastic, plastic and melting objects. In: Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation—SCA. ACM Press (2004). https://doi.org/10.1145/1028523.1028542

Shepard, D.: A two-dimensional function for irregularly spaced points. In: 23rd ACM National Conference, pp. 517–524 (1968)

Song, Y., Liu, Y., Zhang, X.: A transport point method for complex flow problems with free surface. Comput. Part. Mech. (2019)

Steffen, M., Kirby, R.M., Berzins, M.: Analysis and reduction of quadrature errors in the material point method (MPM). Int. J. Numer. Meth. Eng. **76**(6), 922–948 (2008a)

Steffen, M., Wallstedt, P.C., Guilkey, J.E., Kirby, R.M., Berzins, M.: Examination and analysis of implementation choices within the material point method (MPM). Comput. Model. Eng. Sci. **31**(2), 107–127 (2008b)

Steffen, M., Kirby, R.M., Berzins, M.: Decoupling and balancing of space and time errors in the material point method (MPM). Int. J. Numer. Meth. Eng. **82**(10), 1207–1243 (2010)

Strang, W.G., Fix, G.J.: An Analysis of the Finite Element Method. Prentice-Hall, Englewood Cliffs (1973)

Stukowski, A.: Visualization and analysis of atomistic simulation data with OVITO-the open visualization tool. Modell. Simul. Mater. Sci. Eng. **18**(1), 015012 (2009)

Sulsky, D., Gong, M.: Improving the material-point method. In: Innovative Numerical Approaches for Multi-field and Multi-scale Problems, pp. 217–240. Springer, Berlin (2016)

Tran, Q., Berzins, M., Sołowski, W.T.: An improved moving least squares method for the material point method. In: 2nd International Conference on the Material Point Method for Modelling Soil-Water-Structure Interaction (2019)

Tran, L.T., Kim, J., Berzins, M.: Solving time-dependent pdes using the material point method, a case study from gas dynamics. Int. J. Numer. Meth. Fluids **62**(7), 709–732 (2010)

Wallstedt, P.C., Guilkey, J.E.: Improved velocity projection for the material point method. Comput. Model. Eng. Sci. **19**(3), 223–232 (2007)

Wallstedt, P.C., Guilkey, J.E.: An evaluation of explicit time integration schemes for use with the generalized interpolation material point method. J. Comput. Phys. **227**(22), 9628–9642 (2008)

Wang, L., Coombs, W.M., Augarde, C.E., Cortis, M., Charlton, T.J., Brown, M.J., Knappett, J., Brennan, A., Davidson, C., Richards, D., et al.: On the use of domain-based material point methods for problems involving large distortion. Comput. Methods Appl. Mech. Eng. **355**, 1003–1025 (2019)

Yang, W.C., Arduino, P., Miller, G.R., Mackenzie-Helnwein, P.: Smoothing algorithm for stabilization of the material point method for fluid–solid interaction problems. Comput. Methods Appl. Mech. Eng. **342**, 177–199 (2018)

Chapter 10
Other Topics: Modeling of Fluids, Membranes and Temperature Effects

In this final chapter of the book, we discuss some topics including modeling fluids and gases (Sect. 10.1), modeling membranes (Sect. 10.2), heat conduction (Sect. 10.3), and fluid-structure interaction (Sect. 10.4).

10.1 Fluids and Gases

Even though MPM has been developed for solid mechanics applications, it has also been used to model fluids and gases. This section presents a brief discussion on the so-called weakly compressible MPM for fluids. We refer to Zhang et al. (2017) and references therein for advanced incompressible MPM formulations for free surface flow. This simple weakly compressible formulation allows us to model various interesting FSI problems, see York et al. (1999, 2000); Gan et al. (2011); Mao (2013); Yang et al. (2018); Su et al. (2019).

The governing equations for fluids/gases are the same as for the solid, i.e., Eq. 2.20, except the constitutive model. An artificial equation of state is adopted to describe the pressure. Therefore, an MPM code for solids can be equally used to model fluids/gases.

10.1.1 Fluids

For fluids, the stress field is defined as

$$\boldsymbol{\sigma}^f = 2\mu_N \dot{\boldsymbol{\epsilon}} + \lambda_N \text{tr}(\dot{\boldsymbol{\epsilon}})\mathbf{I} - \hat{p}\mathbf{I} \tag{10.1}$$

V. P. Nguyen et al., *The Material Point Method*, Scientific Computation,
https://doi.org/10.1007/978-3-031-24070-6_10

where $\hat{p}$ is the pressure which is determined from an equation of state (EOS), μ_N and λ_N are the shear viscosity [Pa s = kg/(ms)] and the bulk viscosity, respectively. Recall that the strain rate tensor is defined as $\dot{\boldsymbol{\epsilon}} = 0.5(\mathbf{L} + \mathbf{L}^{\mathrm{T}})$ where $\mathbf{L}$ denotes the gradient velocity tensor. A superscript f was used to label the fluid stress, as in an FSI problem, one has to deal with two stress fields – one for the solid and one for the fluid.

If $\lambda_N = \frac{2\mu_N}{3}$, the so-called Stokes condition, the stress field thus becomes

$$\boldsymbol{\sigma}^f = 2\mu_N \dot{\boldsymbol{\epsilon}} - \frac{2\mu_N}{3}\mathrm{tr}(\dot{\boldsymbol{\epsilon}})\mathbf{I} - \hat{p}\mathbf{I} = 2\mu_N \left[\dot{\boldsymbol{\epsilon}} - \frac{1}{3}\mathrm{tr}(\dot{\boldsymbol{\epsilon}})\mathbf{I}\right] - \hat{p}\mathbf{I} \tag{10.2}$$

where the term in the bracket is the deviatoric part of the strain rate tensor.

The pressure of the fluid particle is updated by an EOS. In the case of water and air, the EOS is given by Monaghan (1994); Cueto-Felgueroso et al. (2004)

$$\hat{p} = \kappa \left[\left(\frac{\rho}{\rho_0}\right)^{\gamma} - 1\right] \tag{10.3}$$

where ρ_0 is the initial density and κ is the bulk modulus [Pa=kg/(ms^2)] chosen such that the fluid is nearly incompressible and $\gamma = 7$ for water and $\gamma = 1.4$ for air. The advantage of this EOS, providing a direct relationship between pressure and density, is that there is no need to solve any additional equation for the pressure. The bulk modulus can be very high for a nearly incompressible fluid, such as water, resulting in a very small time step. To increase the time step, a reduced bulk modulus can be used as long as the change in density is less than 3%.

10.1.2 Gases

And for an ideal gas, the stress field is given by Hu and Chen (2006)

$$\boldsymbol{\sigma}^f = -\hat{p}\mathbf{I}, \quad \hat{p} = (\gamma - 1)\rho e \tag{10.4}$$

where e is the specific internal energy and γ is the ratio of specific heats. The specific internal energy is updated using the balance of energy equation (thermal effect was neglected)

$$e_p^{t+\Delta t} = e_p^t + \Delta t \boldsymbol{\sigma}_p^{t+\Delta t} : \Delta \mathbf{e}_p^{t+\Delta t} / \rho_p^{t+\Delta t} \tag{10.5}$$

where the density is updated using the following equation

$$\rho_p^{t+\Delta t} = \frac{\rho_p^t}{1 + \Delta t \mathrm{tr}(\Delta \mathbf{e}_p^{t+\Delta t})} \tag{10.6}$$

It is well known that most numerical simulations of compressible-fluid shocks provide more accurate results if some type of artificial viscosity is used at the shock front. The following artificial viscosity is added to the particle pressure, see e.g. Zhang et al. (2016) for details

$$q = \rho L_e \left(c_0 L_e \dot{\epsilon}_{kk}^2 - c_1 a \dot{\epsilon}_{kk} \right), \quad \dot{\epsilon}_{kk} < 0 \tag{10.7}$$

where c_1 and c_2 are coefficients of artificial viscosity, $a = \sqrt{\gamma \hat{p}/\rho}$ is the local sound speed and L_e is the minimum length of cell sides of the background grid.

After adding the artificial viscosity term q, the stress field is calculated as:

$$\boldsymbol{\sigma}^f = -(\hat{p} + q)\mathbf{I} \tag{10.8}$$

10.1.3 Some Examples

We herein present two simple examples to verify the MPM for the modeling of gases and fluids. The first example is the Sod's shock tube problem and the second is a dam break simulation.

Sod's shock tube. Sod's problem is a test case commonly used in computational hydrodynamics to see how well a certain computational approach works (Sod 1978). This problem, shown in Fig. 10.1, consists of a shock tube where a diaphragm is located in the middle of the tube. Two sides of the diaphragm have different pressures and densities, which make the fluid flows when the diaphragm is broken. The left side of density is 1 and pressure is 1. The right side of density is 0.125 and pressure is 0.1, and both sides have a zero initial velocity. Any set of consistent units suffice. At time $t = 0$, the diaphragm is removed. The fluid is modeled as an ideal gas with $\gamma = 1.4$ (cf. Sect. 10.1).

Only a small modification was made to the one dimensional MPM code developed for solid mechanics: the internal energy e is stored for every fluid particles. The initial value for e is computed using an EOS and the initial density and pressure. The results, obtained with the standard MPM (MUSL update), at time $t = 0.143$, when

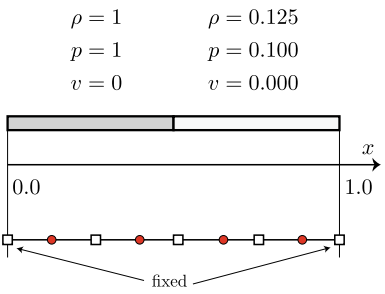


Fig. 10.1 Sod's problem: initial configuration (de Vaucorbeil et al. 2020)

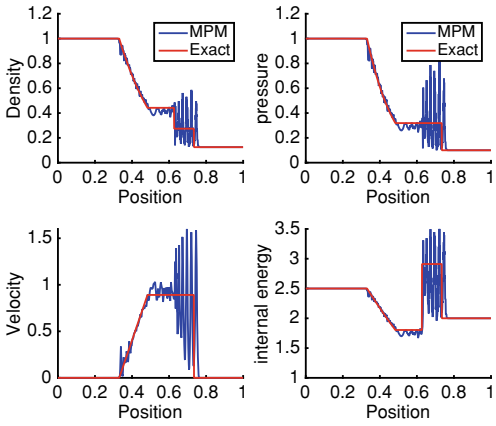


Fig. 10.2 Sod's problem: 200 elements with 3 particle per element. No artificial viscosity (de Vaucorbeil et al. 2020)

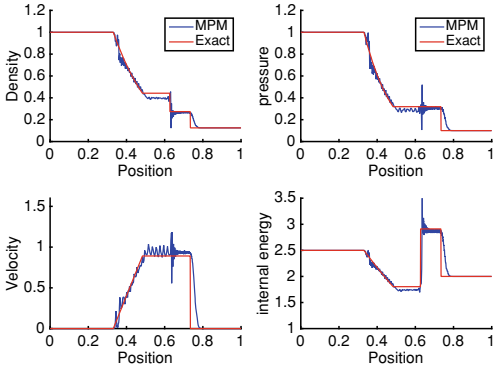


Fig. 10.3 Sod's problem: 200 elements with 3 particle per element. With artificial viscosity (de Vaucorbeil et al. 2020)

the shock traveled a distance of about 0.25, with 300 cells and three particles per cell are given in Figs. 10.2 and 10.3 without and with the use of an artificial viscosity. Note that the data are plotted at the material points. In Fig. 10.4 we also plot the data at the grid nodes. There we observe that a smoother distribution was obtained by this technique as earlier mentioned in the works of e.g. (Andersen and Andersen 2010a). To compute the grid nodal pressure, the particle stress is mapped to the grid nodes in the same manner as the velocity mapping. The M-file for this example is **example1D/mpm1DShockTubeMUSL.m**.

Dam break problems. A simplified dam break simulation with a barrier is depicted in Fig. 10.5. A water column of initial height h_0 and length l_0 is initially constrained by a gate and rests on a smooth, flat surface. At an arbitrary starting time, say $t = 0$, the gate is removed and the water is allowed to flow freely under the force of gravity. This problem has been tackled extensively in the SPH literature and also studied using the MPM by e.g. Mast et al. (2012); Sun et al. (2018).

The domain is discretized by 100×100 cells with 9 PPC (12432 particles). A constant time step of $\Delta = 0.1 h_x/c$ with $c = \sqrt{\kappa/\rho}$ was used. The component normal

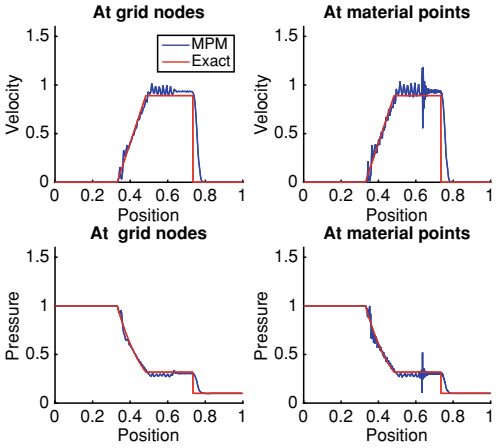


Fig. 10.4 Sod's problem: 200 elements with 3 particle per element with artificial viscosity. Data plotted at material points and at grid nodes (de Vaucorbeil et al. 2020)

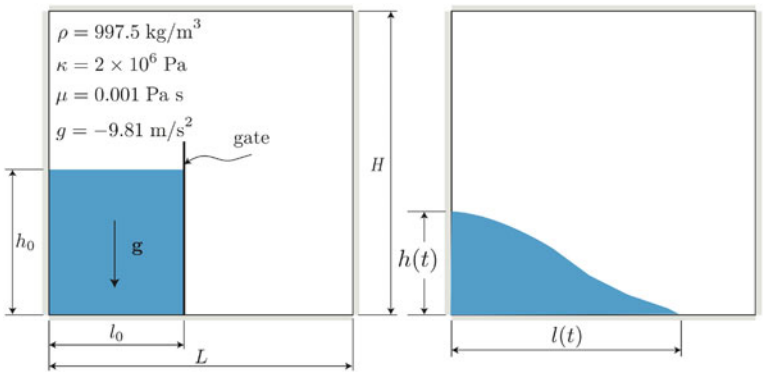


Fig. 10.5 Dam break problem (de Vaucorbeil et al. 2020): $L = 1.61, H = 0.6, l_0 = 0.6, h_0 = 0.6\,\mathrm{m}$ according to Sun et al. (2018)

to the boundaries of the grid velocities are set to zero. The aims of this example are two-fold. First, it is demonstrated that `Karamelo` can do fluid simulations. Second, its solution is validated against the experiment carried out by Lobovskỳ et al. (2014).

For a quantitative assessment of the MPM, the following dimensionless quantities are calculated

$$L(T) := \frac{l(t)}{l_0}, \qquad T := t\sqrt{\frac{h_0 g}{l_0^2}} \tag{10.9}$$

where $l(t)$ is defined as in Fig. 10.5.

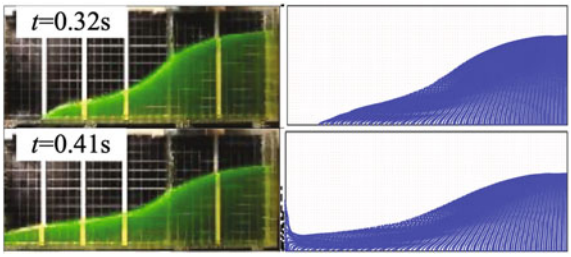


Fig. 10.6 Dam break problem: flow profiles of experiment results (left column) and MPM results (right column). The MPM results were obtained using `Karamelo`

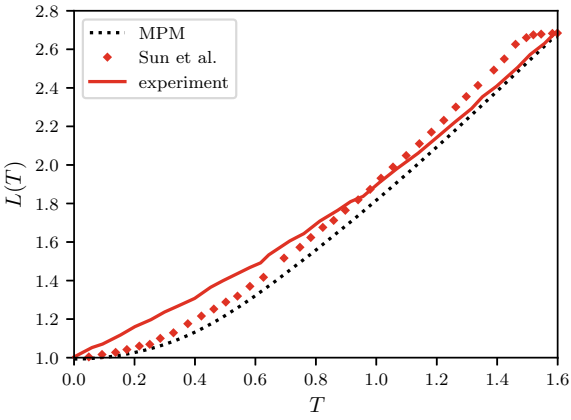


Fig. 10.7 Dam break problem: evolution of the MPM wave front in comparison with experiment result of Lobovskỳ et al. (2014)

The numerical water profile at different time instants are in qualitative good agreement with the experiment (Fig. 10.6). Quantitatively, the simulation result and the experiment are compared in Fig. 10.7.

10.2 Modeling Membranes

Membranes are widely used as structural elements in many diverse engineering applications and also in natur. Examples include parachutes, automobile airbags and human tissues. A membrane is essentially a thin shell with no flexural (bending) stiffness. In other words, membranes have stresses only in the local tangent plane, while other stress components are negligible. In addition, the stress over the thickness is assumed to be constant.

This section presents MPM formulations for modeling membranes. We start with York's technique (York et al. 1999) for one dimensional membranes embedded in a two dimensional solid (Sect. 10.2.1). This simple method, in which a membrane is discretized by a set of particles, is largely sensitive to the grid resolution. Therefore, we discuss another method where a membrane is modeled by a set of bar elements based on the MPM-FEM method of Lian et al. (2011c) (Sect. 10.2.2).

10.2.1 York's MPM Algorithm for Membranes

York et al. (1999) presented a simple MPM algorithm for modeling elastic membranes. First, the membrane is discretized by one layer of particles along the thickness. Doing so enforces automatically the constant through-thickness stress condition. Second, the stresses at the particles are computed in such a way that normal stress (in the local coordinate system attached to the particles) is zero. Figure 10.8 illustrates the ideas.

Let us denote the updated particle strain by $\boldsymbol{\epsilon}_p^{t+\Delta t}$. We transform it to the particle local coordinate system, denoted by $\hat{\boldsymbol{\epsilon}}_p^{t+\Delta t}$. In what follows we skip the superscript $t+\Delta t$ implicitly assuming that a USL formulation is being used so that stress is always updated last. And to bypass the complexity of 3D tensor transformation, we first consider the case of uniaxial tension where we write the local tangent strain

$$\hat{\epsilon}_{p,t} = (\cos^2\theta)\epsilon_{p,xx} + (\sin^2\theta)\epsilon_{p,yy} + (\sin 2\theta)\epsilon_{p,xy} \tag{10.10}$$

where θ is the orientation of the tangent at the particle under consideration. The tangent stress of the particle in the local coordinate system is thus given by

$$\hat{\sigma}_{p,t} = E\hat{\epsilon}_{p,t} \tag{10.11}$$

while other stress components are zeros. The local stresses are transformed back to the global coordinate system where the internal force vector is computed:

$$\begin{bmatrix} \sigma_{p,xx} \\ \sigma_{p,yy} \\ \sigma_{p,xy} \end{bmatrix} = \begin{bmatrix} (\cos^2\theta)\hat{\sigma}_{p,t} \\ (\sin^2\theta)\hat{\sigma}_{p,t} \\ (\sin\theta\cos\theta)\hat{\sigma}_{p,t} \end{bmatrix} \tag{10.12}$$

and this is the particle stress to be stored and used to compute the grid force vectors in the next time step.

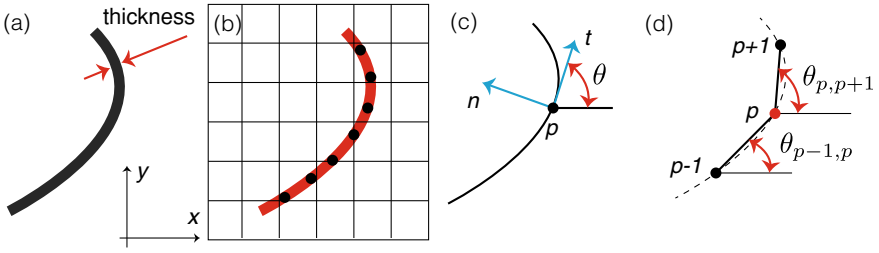


Fig. 10.8 Modeling membrane in the MPM framework: **a** membrane, **b** MPM representation of a membrane, **c** local coordinate system defined for each particle and **d** determination of the local tangent using the connectivity data (Nguyen et al. 2017)

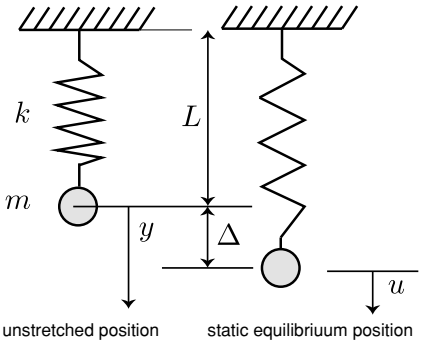


Fig. 10.9 Spring-mass problem (Nguyen et al. 2017)

The initial particle mass is left to determine. Assuming that there is n membrane particles, s is the initial length of the membrane, t its in-plane thickness and ρ is the membrane density, then the membrane particle's mass is given by York et al. (1999)

$$m_p = \frac{st\rho}{n} \tag{10.13}$$

The tangent to particle p is determined using the connectivity data of the membrane curve, cf. Fig. 10.8d. That is

$$\theta_p = \frac{\theta_{p-1,p} + \theta_{p,p+1}}{2} \tag{10.14}$$

which is the most efficient method to determine the local coordinates for the membrane particle at least for 2D problems i.e., 1D membranes.

Examples. To verify the York membrane formulation, we consider the spring-mass problem of which an exact solution is available: a rigid mass is attached to a massless spring of unstretched length L and spring stiffness k which is connected to a stationary wall (Fig. 10.9).

If position y is measured from the undeformed position, the governing differential equation of this system is

$$mg - ky = m\frac{d^2y}{dt^2} \tag{10.15}$$

By using $mg = k\Delta$ where Δ is the spring elongation at static equilibrium, one can change the above equation to

$$m\frac{d^2u}{dt^2} + ku = 0, \quad u = \Delta - y \tag{10.16}$$

of which solution is given by

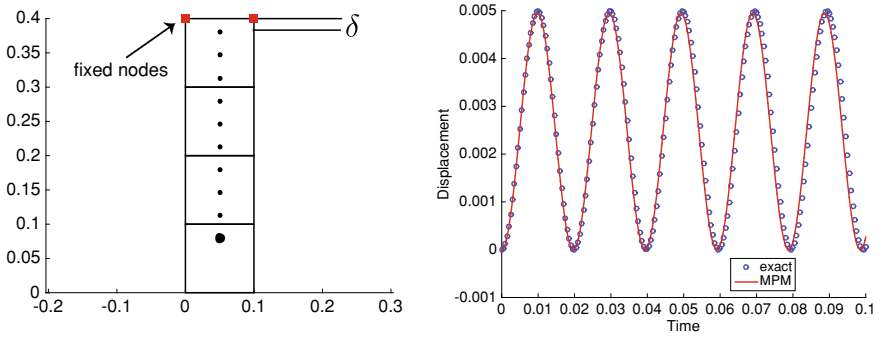


Fig. 10.10 Spring-mass problem: background grid and particles (left) and deflection of the mass in time (right). Time step was $\Delta t = 0.8\Delta_c/c$ where $c = \sqrt{E/\rho}$. It is emphasized that as boundary conditions are imposed on the grid nodes, the top particle must be positioned close to these nodes (Nguyen et al. 2017)

$$u(t) = \Delta \cos(\omega t) + \frac{\dot{u}_0}{\omega} \sin(\omega t) \tag{10.17}$$

where $\omega = \sqrt{k/m}$ is the natural frequency and $\dot{u}_0$ is the initial velocity of the mass, which is simply zero in this example. Therefore the deflection of the mass is given by

$$y(t) = \Delta\,[1 - \cos(\omega t)] \tag{10.18}$$

in which positive value means downward deflection.

For this simulation,[1] $A = 0.1$, $E = 1.0e6$, $\rho = 0.1$, $g = -250$,[2] and $L = 0.3$ which is the distance between the top and bottom particles (as particles do not have extent in MPM). Any set of consistent units suffice. The spring stiffness is $k = EA/L$ where A is the cross-sectional area of the spring; the spring can be thought of as a elastic bar, the static stretch $\Delta = 0.0025$, the period of oscillation T is $T = 2\pi/\omega = 0.02$. As a solid MPM code is used for this simulation, we need a null Poisson's ratio. The heavy particle has a mass of 3.33 which is 10^4 times the mass of the other particles. We first use 4 cells and 10 particles as shown in Fig. 10.10 following York et al. (1999).

The numerical displacement of the mass (the big black circle in Fig. 10.10), calculated as the difference between its current vertical position and its original vertical one, is compared with the exact solution given in Eq. 10.18. The exact kinetic energy is $mv^2/2$ where $v = \omega\Delta \sin(\omega t)$ is the velocity and the potential energy is $ky^2/2$. Plot of these energies is given in Fig. 10.11 with the time history of energy in the MPM simulation. A good match with exact solutions was obtained for this cell/particle arrangement.

[1] The M-file is **mpmSpringMass.m**. Although the problem is one dimensional the grid is 2D as we wanted to test the membrane formulation which is generally a 1D curve embedded in a 2D domain.

[2] Used as a body force in the MPM calculation of grid external forces.

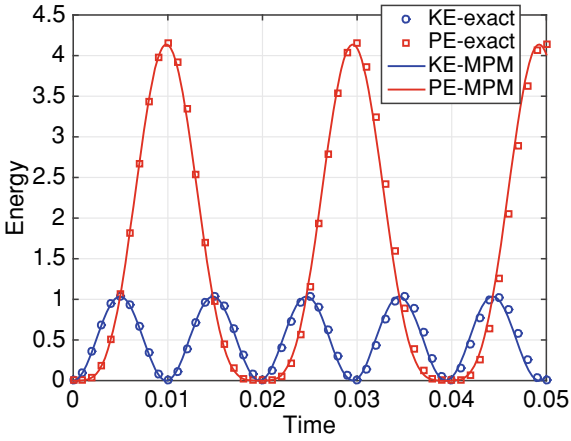


Fig. 10.11 Spring-mass problem: kinetic and potential energies. Time step was $\Delta t = 0.8\Delta_c/c$ where $c = \sqrt{E/\rho}$ (Nguyen et al. 2017)

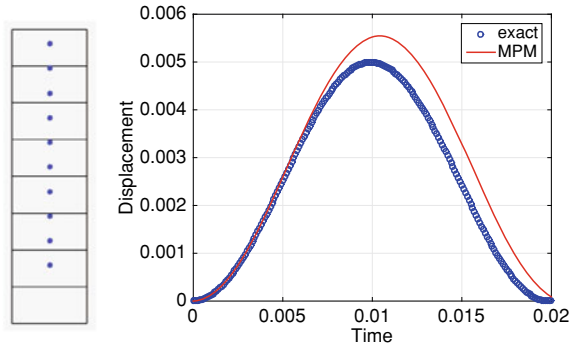


Fig. 10.12 Spring-mass problem: results obtained with 8 cells and 10 particles (Nguyen et al. 2017)

The result is, however, very much sensitive to the discretization. We present in Fig. 10.12 some results obtained with different grids and particles.

10.2.2 A Coupled FEM-MPM for Modeling Membranes

Hamad et al. (2015) developed a new MPM-based approach to simulate the installation process and the behaviour of geosynthetics systems for geomechanical applications. The background grid is a 3D tetrahedral mesh with material points representing the solid whereas the membrane is discretized with another mesh consisting of three-node triangular elements following the coupled FEM-MPM method of Lian et al. (2011c). The membrane is therefore treated differently than the method of York et al. (1999) where only particles are adopted to represent the membrane.

Herein we present the formulation of Lian et al. (2011c); Hamad et al. (2015) in the context of 1D membranes. The idea is illustrated in Fig. 10.13 where a closed

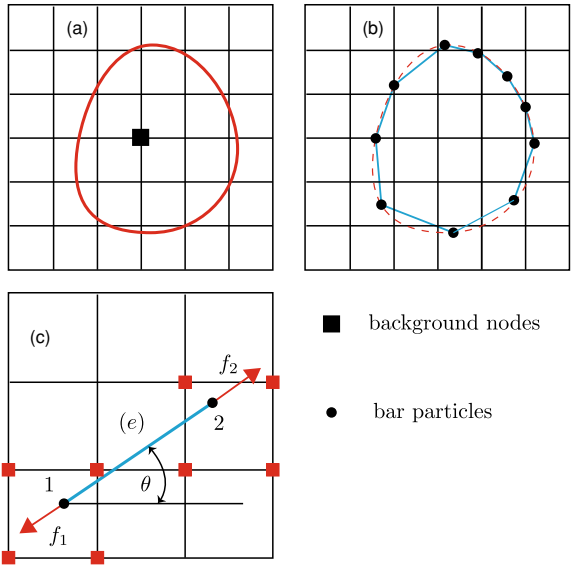


Fig. 10.13 Modeling membranes using a coupled FEM-MPM approach: **a** a membrane overlaid on a background grid, **b** discrete representation of the membrane using bar elements, **c** interaction between a bar element and the background grid (Nguyen et al. 2017)

curved membrane is represented by a number of two-noded bar elements. The nodes of the bar elements are referred to as membrane particles to differentiate them from other particles that might be present in the system. However the membrane particles do not carry stresses. The internal forces are computed in two steps. First, the internal forces at the membrane particles are computed using the bar elements i.e., following the FEM. Second, these forces are projected to the grid to obtain the final internal forces (used in the momentum equation).

For a bar element e the internal forces at its nodes are computed as Lian et al. (2011c)

$$l^{t+\Delta t} = ||\mathbf{x}_2 - \mathbf{x}_1||^{t+\Delta t} \tag{10.19a}$$

$$\epsilon^{t+\Delta t} = \frac{l^{t+\Delta t} - l_0}{l_0} \tag{10.19b}$$

$$\sigma^{t+\Delta t} = E\epsilon^{t+\Delta t} \tag{10.19c}$$

$$f_1^{t+\Delta t} = -\sigma^{t+\Delta t} A \tag{10.19d}$$

$$f_2^{t+\Delta t} = +\sigma^{t+\Delta t} A \tag{10.19e}$$

where A denotes the cross sectional area of the bar element. These element forces are mapped to the grid using the grid shape functions

$$\mathbf{f}_{I,1}^{\text{int}} = -N_I(\mathbf{x}_1)\begin{bmatrix} f_1 \cos\theta \\ f_1 \sin\theta \end{bmatrix}, \quad \mathbf{f}_{I,2}^{\text{int}} = -N_I(\mathbf{x}_2)\begin{bmatrix} f_2 \cos\theta \\ f_2 \sin\theta \end{bmatrix} \tag{10.20}$$

where the minus sign is due to the MPM convention in the internal force vector i.e., $\mathbf{f} = \mathbf{f}^{\text{ext}} + \mathbf{f}^{\text{int}}$ not as $\mathbf{f} = \mathbf{f}^{\text{ext}} - \mathbf{f}^{\text{int}}$ commonly used in FEM. As $N_I(\mathbf{x}_i),\ i = 1, 2$, is only non zero for nodes of the cell containing $\mathbf{x}_i$, the membrane particles contribute only to the internal forces of nodes of the cells in which they reside. These are the red filled squares in Fig. 10.13.

Next we present a more general way to couple FEM and MPM which is applicable not only to membrane elements but also to solid elements. First, the displacement increment at the particles (i.e., the finite element nodes) is computed from the updated grid nodal velocities

$$\Delta\mathbf{u}_J = \Delta t \sum_I N_I(\mathbf{x}_J)\mathbf{v}_I^{t+\Delta t} \tag{10.21}$$

which is then transformed to the local coordinate system defined by the element under consideration

$$\Delta\hat{\mathbf{u}}_J = \mathbf{Q}\Delta\mathbf{u}_J \tag{10.22}$$

where $\mathbf{Q}$ is the vector-to-vector transformation matrix given by for completeness

$$\mathbf{Q} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \tag{10.23}$$

The strain increment evaluated at the element center (as we assumed constant stress elements are considered, therefore there is only one single integration point–the element center) by

$$\Delta\hat{\epsilon} = \frac{1}{l_e}\left(\Delta\hat{\mathbf{u}}_2 - \Delta\hat{\mathbf{u}}_1\right), \quad \Delta\hat{\epsilon} = \mathbf{B}_0\Delta\hat{\mathbf{u}} \tag{10.24}$$

where the first equation applies for membrane/bar elements and the second is general. The strain increment is then used to get the updated stresses at the element center $\hat{\boldsymbol{\sigma}}$ via any constitutive models. The element nodal forces are then determined using

$$\hat{\mathbf{f}} = \mathbf{B}_0^{\mathrm{T}}\hat{\boldsymbol{\sigma}} \tag{10.25}$$

which is transformed to the global coordinate system via

$$\mathbf{f} = \mathbf{Q}^{-1}\hat{\mathbf{f}} \tag{10.26}$$

The FEM-MPM solution process for one time step t to $t + \Delta t$ using the USL is given in Algorithm 21. Note that changes are in lines 3 (computing internal forces not from particle stresses but from particle forces) and lines 14–21 (updating particle forces).

Algorithm 21 Solution procedure of explicit FEM-MPM (USL).

Mapping from particles to nodes
  Compute nodal mass, momentum and external forces as usual
  Compute internal force $\mathbf{f}_I^{\text{int},t} = -\sum_{p=1}^{n_p} N_I(\mathbf{x}_p^t)\mathbf{f}_p^t$
  Compute nodal force $\mathbf{f}_I^t = \mathbf{f}_I^{\text{ext},t} + \mathbf{f}_I^{\text{int},t}$
end
Update the momenta $(m\mathbf{v})_I^{t+\Delta t} = (m\mathbf{v})_I^t + \mathbf{f}_I^t \Delta t$
Fix Dirichlet nodes
Update particles
  Update particle velocities $\mathbf{v}_p^{t+\Delta t} = \mathbf{v}_p^t + \Delta t \sum_I N_I(\mathbf{x}_p^t)\mathbf{f}_I^t/m_I^t$
  Update particle positions $\mathbf{x}_p^{t+\Delta t} = \mathbf{x}_p^t + \Delta t \sum_I N_I(\mathbf{x}_p^t)(m\mathbf{v})_I^{t+\Delta t}/m_I^t$
  Get nodal velocities $\mathbf{v}_I^{t+\Delta t} = (m\mathbf{v})_I^{t+\Delta t}/m_I^t$
  Compute particle displacement increments $\Delta\mathbf{u}_p^{t+\Delta t} = \sum_I \Delta t N_I(\mathbf{x}_p^t)\mathbf{v}_I^{t+\Delta t}$
end
Update particle forces
  for each membrane element **do**
    Get the displacement increments of its nodes $\Delta\mathbf{u}^{t+\Delta t}, \rightarrow \Delta\hat{\mathbf{u}}^{t+\Delta t}$
    Compute the strain increment at the integration point $\Delta\hat{\boldsymbol{\epsilon}} = \mathbf{B}\Delta\hat{\mathbf{u}}^{t+\Delta t}$
    Compute the stress at this integration point $\hat{\boldsymbol{\sigma}}(\hat{\boldsymbol{\sigma}}^t, \Delta\hat{\boldsymbol{\epsilon}})$
    Compute the particle forces $\hat{\mathbf{f}} = \mathbf{B}^{\mathrm{T}}\hat{\boldsymbol{\sigma}}$
    Transform $\hat{\mathbf{f}}$ to the global system $\mathbf{f}_p^{t+\Delta t}$
  end for
end

Spring-mass problem. To verify the implementation and the performance of the FE MPM approach to modeling membranes, we reconsider the spring-mass problem. We considered different discretizations of the membrane while keeping the background grid fixed at 8 cells. Excellent agreement with analytical solutions and insensitivity to the discretization can be observed (Fig. 10.14).

Disk-wire problem. As a solid-membrane interaction problem, we consider the hypothetical problem of two disks impacting a stationary wire[3] originally solved by York et al. (1999), see Fig. 10.15. The constitutive model for the disks is plane stress linear elastic, and that of the wire is uniaxial stress. The material properties used are listed in Table 10.1. The computational domain is 12×10 with unit thickness. The membrane is discretized by a uniform mesh where each element has a length of $L_0/(n-1)$, n is the number of membrane particles. Thus each internal membrane particle has a mass of $\rho_w L_0/(n-1)$ and the mass of the first and last particle is $0.5\rho_w L_0/(n-1)$.

Some simulation snapshots are given in Fig. 10.16. As the disks make contact with the wire they rotate clockwise, and when they are bouncing they rotate counter-clockwise. Figure 10.17 plots the evolution of kinetic, potential and total energies. As can be observed the total energy of the system is fairly well conserved for this relatively coarse discretization. Even though York et al. (1999) did not mention the discretization details we can firmly state that we used a much coarser membrane discretization thanks to the FE membrane elements.

[3] The M-file for this simulation is **mpmFEMDiskWire.m**.

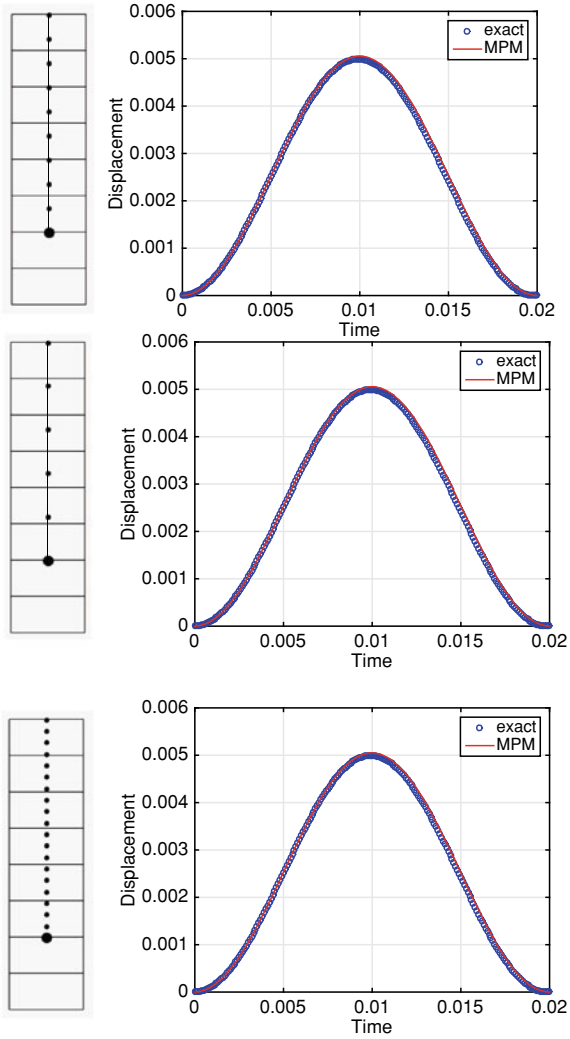


Fig. 10.14 Spring-mass problem: FEM-MPM results obtained with different discretizations. Top figure: 8 cells and 10 particles, 8 cells 6 particles (middle) and bottom figure: 8 cells and 20 particles (Nguyen et al. 2017)

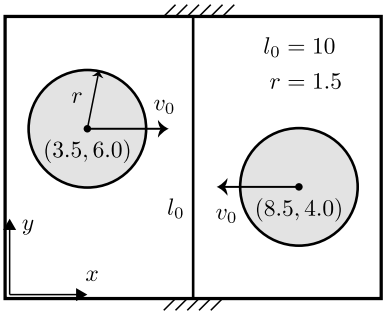


Fig. 10.15 Disk-wire problem. The initial position of the disks are indicated by the coordinates of its centers (Nguyen et al. 2017)

Table 10.1 Parameters for the disk-wire problem

Parameter	Disk value	Wire value
Density	1.0	0.5
Young's modulus	1×10^4	1×10^4
Poisson's ratio	0.3	0.0
Initial velocity	3.0	0.0

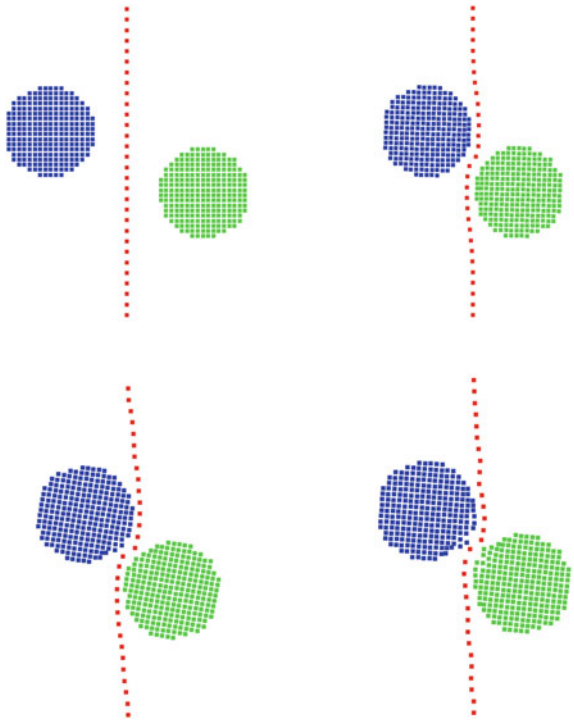


Fig. 10.16 Disk-wire problem. The grid consists of 35×30 cells where each cell initially has 2×2 particles. The wire is discretized by 29 two-noded elements and 30 particles (Nguyen et al. 2017)

10.3 Thermo-Mechanical Problems

Thermo-mechanical problems arise in several engineering applications, particularly manufacturing processes (e.g. welding processes, machining processes and hot/warm metal forming processes). In a thermo-mechanical problem, the unknowns are the deformation field (displacement/velocity) and the temperature field. The latter modifies the former via temperature-dependent constitutive models and the deformation affects the temperature via plastic work dissipated as heat.

This section presents a simple thermo-mechanical MPM formulation. First, the thermal problem is discussed in Sect. 10.3.1, then the coupled thermo-mechanical algorithm is given in Sect. 10.3.2 where the explicit MPM formulation treated in

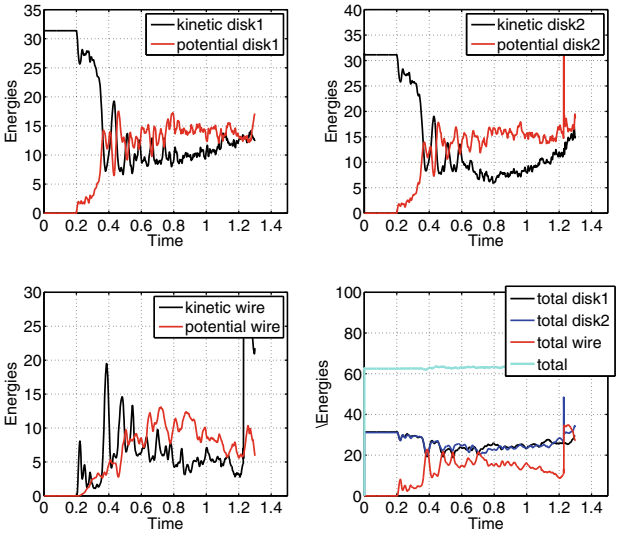


Fig. 10.17 Disk-wire problem. Evolution of kinetic, potential and total energies. The grid consists of 35×30 cells where each cell initially has 2×2 particles. The wire is discretized by 29 two-noded elements and 30 particles (Nguyen et al. 2017)

Sect. 2.5.2 is slightly modified to incorporate the temperature. For more details, we refer to Nairn and Guilkey (2015); Tao et al. (2016).

10.3.1 Thermal Problem

Our derivation of the MPM equation for the thermal problem is to adopt the FEM equation and use the material points as quadrature points. To get the FEM semi-discrete equation, we follow the standard procedure of going from the strong form to the weak form, followed by the introduction of the FE approximations of the trial and test functions. Details can be found in Tao et al. (2016). Note that this thermal MPM algorithm follows the mechanical MPM one. For rectilinear geometries, thanks to the background Eulerian grid, a simple (and robust) finite difference method can be used, see e.g. Chen et al. (2008).

The partial differential equation for the thermal problem is the internal energy balance equation that reads

$$\rho c \dot{T} + \nabla \cdot \mathbf{q} = \gamma^* \qquad \text{in } \Omega \tag{10.27}$$

where $T(\mathbf{x}, t)$ denotes the temperature field; ρ and c being the mass density and specific heat of the material, respectively and the heat source is represented by γ^*. In the above equation, $\mathbf{q}$ is the heat flux which is given by the following Fourier's

law

$$\mathbf{q} = -k\nabla T \tag{10.28}$$

where k is the thermal conductivity. A thermal conductivity tensor can be equally used.

From Eq. 10.27, one can obtain the following semi-discrete equation

$$C_{IJ}\dot{T}_J = Q_I^{\text{int}} + Q_I^{\text{ext}}, \quad I = 1, 2, \ldots, n_n \tag{10.29}$$

where

$$C_{IJ} := \int_\Omega \rho c \phi_I \phi_J \mathrm{d}\Omega \tag{10.30}$$

$$Q_I^{\text{int}} := \int_\Omega \mathbf{q} \cdot \nabla \phi_I \mathrm{d}\Omega \tag{10.31}$$

$$Q_I^{\text{ext}} := \int_\Omega \gamma^* \phi_I \mathrm{d}\Omega - \int_{\Gamma_q} \phi_I q^* \mathrm{d}\Gamma \tag{10.32}$$

where q^* is the prescribed heat flux similar to the tractions in solid mechanics. These quantities are approximated as follows in the spirit of the MPM

$$C_{IJ} = \sum_p m_p c_p \phi_I(\mathbf{x}_p)\phi_J(\mathbf{x}_p) \tag{10.33}$$

$$Q_I^{\text{int}} = \sum_p V_p \mathbf{q}_p \cdot \nabla \phi_I(\mathbf{x}_p) \tag{10.34}$$

$$Q_I^{\text{ext}} = \sum_p V_p \gamma^* \phi_I(\mathbf{x}_p) \tag{10.35}$$

where we have omitted the external force due to q^*, see Sect. 5.2.3 for its treatment.

Similar to the lumped mass matrix, the matrix C_{IJ} is made diagonal using the same row-sum technique and thus Eq. 10.29 is simplified to

$$C_I\dot{T}_I = Q_I^{\text{int}} + Q_I^{\text{ext}}, \quad C_I = \sum_p m_p c_p \phi_I(\mathbf{x}_p) \tag{10.36}$$

Again, in the same manner that the particle velocity is mapped to the grid nodes, one does the same thing for the temperature

$$T_I^t = \left(\frac{1}{C_I^t}\right) \sum_p \phi_I(\mathbf{x}_p^t)(mc)_p^t T_p^t \tag{10.37}$$

The complete algorithm is given in Algorithm 22. Note that a forward Euler was adopted to advance Sect. 10.36 in time. Other time integration schemes can also be used. As can be seen, the algorithm adopts the MUSL scheme in which the updated particle temperature is mapped back to the grid and this updated grid temperature is used to compute the heat flux $\mathbf{q}$. Without doing so would result in bad results. Moreover, we enforce the Dirichlet BCs on the nodes, but it can be enforced on the particles as done in Tao et al. (2016). As this thermal MPM solver is meant to be coupled with a mechanical MPM solver, we use $\mathbf{x}_p^t$ instead of just $\mathbf{X}_p$ as in purely thermal analysis the particles do not move.

Algorithm 22 Solution procedure of explicit thermo MPM.

while $t < t_f$ **do**
  Mapping from particles to nodes (P2G)
    Compute nodal mass $C_I^t = \sum_p \phi_I(\mathbf{x}_p^t) m_p c_p$
    Compute nodal temperature $T_I^t = \left(\frac{1}{C_I^t}\right) \sum_p \phi_I(\mathbf{x}_p^t) m_p c_p T_p^t$
    Compute external force $Q_I^{\text{ext},t} = \sum_p V_p \gamma^* \phi_I(\mathbf{x}_p^t)$
    Compute internal force $Q_I^{\text{int},t} = \sum_p V_p \mathbf{q}_p^t \cdot \nabla \phi_I(\mathbf{x}_p^t)$
    Compute nodal force $Q_I^t = Q_I^{\text{ext},t} + Q_I^{\text{int},t}$
  end
  Update the temperature $\tilde{T}_I^{t+\Delta t} = T_I^t + Q_I^t \Delta t / C_I^t$
  Fix Dirichlet nodes I e.g. $T_I^t = T^*$ and $\tilde{T}_I^{t+\Delta t} = T^*$
  Update particles (G2P)
    Update particle temperature $T_p^{t+\Delta t} = T_p^t + \sum_I \phi_I(\mathbf{x}_p^t) \left[\tilde{T}_I^{t+\Delta t} - T_I^t\right]$
    Update grid temperature $T_I^{t+\Delta t} = \left(\frac{1}{C_I^t}\right) \sum_p \phi_I(\mathbf{x}_p^t)(mc)_p^t T_p^{t+\Delta t}$
    Fix Dirichlet nodes $T_I^{t+\Delta t} = T^*$
    Update flux $\mathbf{q}_p^{t+\Delta t} = -k \sum_I \nabla \phi_I(\mathbf{x}_p^t) T_I^{t+\Delta t}$
  end
  Advance time $t = t + \Delta t$
end while

10.3.2 Coupled Thermo-Mechanical MPM

By combining the thermal algorithm given in Sect. 10.3.1 and the mechanical algorithm in Sect. 2.5.2, one can come up with a simple coupled thermal-mechanical MPM formulation. In its most basic form, the two problems are solved on the same background grid and the same time step are used. The resulting algorithm is shown in Algorithm 23.

To complete the thermal-mechanical MPM formulation, one needs a temperature-dependent constitutive model. For simplicity, we consider a thermo-elastic material model in which the stress is given by

$$\dot{\boldsymbol{\sigma}} = (\lambda \text{tr}\dot{\boldsymbol{\varepsilon}}^e)\mathbf{I} + 2\mu\dot{\boldsymbol{\varepsilon}}^e, \quad \dot{\boldsymbol{\varepsilon}}^e = \dot{\boldsymbol{\varepsilon}} - \dot{\boldsymbol{\varepsilon}}^T, \quad \dot{\boldsymbol{\varepsilon}}^T = \frac{\alpha}{\Delta t}(T^{t+\Delta t} - T^t)\mathbf{I} \tag{10.38}$$

where $\boldsymbol{\varepsilon}^T$ denotes the thermal strain, $\boldsymbol{\varepsilon}^e$ the elastic strain, and α is the coefficient of thermal expansion. Note that Algorithm 23 applies equally to other thermal-elasto-plastic materials.

Algorithm 23 Solution procedure of explicit thermo-mechanical MPM.

Initialization
end
while $t < t_f$ **do**
  Mapping from particles to nodes (P2G)
    Mechanical fields
      Proceed as usual
    end
    Thermal field
      Compute nodal mass $C_I^t = \sum_p \phi_I(\mathbf{x}_p^t) m_p c_p$
      Compute nodal temperature $T_I^t = \left(\frac{1}{C_I^t}\right) \sum_p \phi_I(\mathbf{x}_p^t) m_p c_p T_p^t$
      Compute external force $Q_I^{\text{ext},t} = \sum_p V_p \gamma^* \phi_I(\mathbf{x}_p^t)$
      Compute internal force $Q_I^{\text{int},t} = \sum_p V_p \mathbf{q}_p^t \cdot \nabla \phi_I(\mathbf{x}_p^t)$
      Compute nodal force $Q_I^t = Q_I^{\text{ext},t} + Q_I^{\text{int},t}$
    end
  end
  Update the momenta $(m\tilde{\mathbf{v}})_I^{t+\Delta t} = (m\mathbf{v})_I^t + \mathbf{f}_I^t \Delta t$
  Update the temperature $\tilde{T}_I^{t+\Delta t} = T_I^t + Q_I^t \Delta t / C_I^t$
  Fix mechanical Dirichlet nodes I e.g. $(m\mathbf{v})_I^t = \mathbf{0}$ and $(m\tilde{\mathbf{v}})_I^{t+\Delta t} = \mathbf{0}$
  Fix thermal Dirichlet nodes I e.g. $T_I^t = T^*$ and $\tilde{T}_I^{t+\Delta t} = T^*$
  Update particle velocities and grid velocities (double mapping)
    Get nodal velocities $\tilde{\mathbf{v}}_I^{t+\Delta t} = (m\tilde{\mathbf{v}})_I^{t+\Delta t} / m_I^t$
    Update particle velocities $\mathbf{v}_p^{t+\Delta t} = \mathbf{v}_p^t + \sum_I \phi_I(\mathbf{x}_p^t) \left[\tilde{\mathbf{v}}_I^{t+\Delta t} - \mathbf{v}_I^t\right]$
    Update grid velocities $(m\mathbf{v}_I)^{t+\Delta t} = \sum_p \phi_I(\mathbf{x}_p^t)(m\mathbf{v})_p^{t+\Delta t}$
    Fix Dirichlet nodes $(m\mathbf{v})_I^{t+\Delta t} = \mathbf{0}$
    Update particle temperature $T_p^{t+\Delta t} = T_p^t + \sum_I \phi_I(\mathbf{x}_p^t) \left[\tilde{T}_I^{t+\Delta t} - T_I^t\right]$
    Update grid temperature $T_I^{t+\Delta t} = \left(\frac{1}{C_I^t}\right) \sum_p \phi_I(\mathbf{x}_p^t)(mc)_p^t T_p^{t+\Delta t}$
    Fix Dirichlet nodes $T_I^{t+\Delta t} = T^*$
  end
  Update particles (G2P)
    Get nodal velocities $\mathbf{v}_I^{t+\Delta t} = (m\mathbf{v})_I^{t+\Delta t} / m_I^t$
    Update particle positions $\mathbf{x}_p^{t+\Delta t} = \mathbf{x}_p^t + \Delta t \sum_I \phi_I(\mathbf{x}_p^t) \mathbf{v}_I^{t+\Delta t}$
    Compute velocity gradient $\mathbf{L}_p^{t+\Delta t} = \sum_I \nabla \phi_I(\mathbf{x}_p^t) \mathbf{v}_I^{t+\Delta t}$
    Updated gradient deformation tensor $\mathbf{F}_p^{t+\Delta t} = (\mathbf{I} + \mathbf{L}_p^{t+\Delta t} \Delta t) \mathbf{F}_p^t$
    Update volume $V_p^{t+\Delta t} = \det \mathbf{F}_p^{t+\Delta t} V_p^0$
    Update stresses $\boldsymbol{\sigma}_p^{t+\Delta t} = \boldsymbol{\sigma}_p^t + \Delta \boldsymbol{\sigma}_p(\mathbf{L}_p^{t+\Delta t}, T_p^{t+\Delta t})$
    Update flux $\mathbf{q}_p^{t+\Delta t} = -k \sum_I \nabla \phi_I(\mathbf{x}_p^t) T_I^{t+\Delta t}$
  end
  Advance time $t = t + \Delta t$
end while

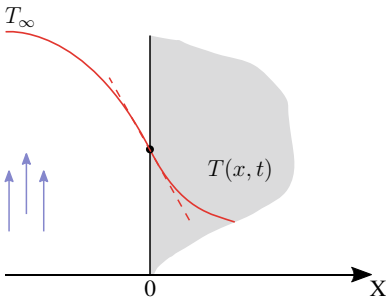


Fig. 10.18 Temperature profile and convection boundary conditions

Convection boundary condition. In heat transfer problems, the convection boundary condition, known also as the Newton boundary condition, corresponds to the existence of convection heating (or cooling) at the surface and is obtained from the surface energy balance. Convection boundary condition is probably the most common boundary condition encountered in practice since most heat transfer surfaces are exposed to a convective environment at specified parameters.

To impose this type of boundary condition, one prescribes a heat flux $\mathbf{q}$ normal to the surface such that:

$$q^* = \mathbf{q} \cdot \mathbf{n} = h\,[T_\infty - T_\Gamma(t)] \tag{10.39}$$

where $\mathbf{n}(t)$ is the normal to the convective surface, h is the heat transfer coefficient, T_∞ the ambiant temperature, and $T_\Gamma(t)$ the surface temperature (see Fig. 10.18).

Applying this boundary condition to a particle is straight forward. However, to apply it to a node, one has to calculate the thermal force at the node:

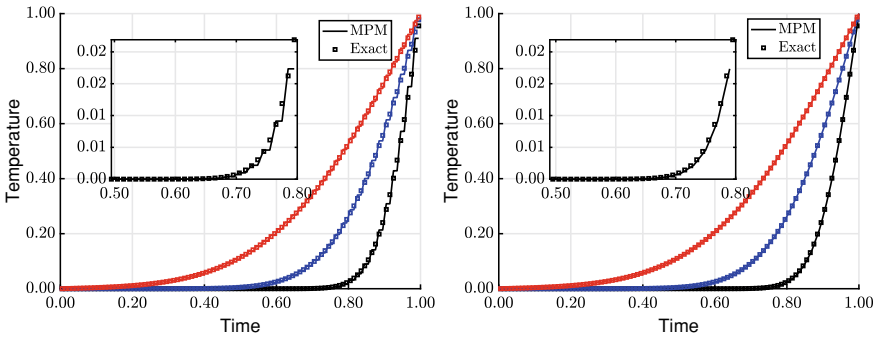
$$Q_I^{\text{convection}} = -\sum_{p=0}^{N_p} A_p h\,[T_\infty - T_\Gamma(t)]\,\phi_I(\mathbf{x}_p^t) \tag{10.40}$$

with A_p the surface area attached to particle p. The minus before the sum is due to Eq. 10.32.

10.3.3 Verification Tests

We present some tests for the verification of the thermal MPM algorithm and the coupled thermal-mechanical MPM. All the tests are solved using the ULMPM only.

One dimensional thermal test. Let us consider a bar of length $L = 1$, with $\rho = c = k = 1$. The bar temperature is initially set at $T_0 = 0$ and the right extremity is heated up to T_1 at time $t = 0$ and held fixed. Any set of consistent units suffice. The exact solution for the temperature in the bar is given by Tao et al. (2016)



(a) particle temperature (b) grid temperature

Fig. 10.19 One dimensional heat conduction: solutions obtained with the ULMPM (hat functions) with 100 grid cells and 1 particle per cell. Black is for $t = 0.0075$, blue is for $t = 0.03$ and red is for $t = 0.05$ (de Vaucorbeil et al. 2020)

$$T^{\text{exact}}(x,t) = T_0 + (T_1 - T_0)x + 2(T_1 - T_0)\sum_{n=1}^{\infty} \frac{(-1)^n}{n\pi} \exp(-[n\pi]^2 t) \sin(n\pi x) \tag{10.41}$$

The MPM solutions, with $T_0 = 0$ and $T^* = T_1 = 1$, are given in Fig. 10.19 in terms of the particle temperature and grid temperature. The time step is chosen to be $\Delta t = 8 \times 10^{-5}$ which is smaller than the theoretical maximum value of $(h_x)^2/(2k)$. However, we find that larger time steps also yield accurate results. The results given in the referred figure confirms the finding in the literature that the particle data are noisy whilst the grid data (i.e., $T_I^{t+\Delta t}$ not $\tilde{T}_I^{t+\Delta t}$) are not. The M-file for this example is **example1D/mpmThermo1D.m**.

Two dimensional thermal test. Let us now consider a rectangular plate of dimensions $L \times H$, with $\rho = c = k = 1$. The bar temperature is initially set at $T_0 = 0$ and the external surface is heated up to $T_1 = 100$ at time $t = 0$ and held fixed. Any set of consistent units suffice. The exact temperature field in the plate is given by Tao et al. (2016)

$$T^{\text{exact}}(\mathbf{x},t) = T_1 + \frac{16(T_0 - T_1)}{\pi^2} \sum_{i=1,3,\ldots} \sum_{j=1,3,\ldots} \frac{\exp(-\pi^2(i^2/L^2 + j^2/H^2)t)}{ij} \sin\left(\frac{i\pi x}{L}\right) \sin\left(\frac{j\pi y}{H}\right) \tag{10.42}$$

In the simulations, a unit square is considered and a grid of 20×20 cells with four particles per cell is used. The temperature boundary condition is applied on the particles residing in the boundary cells. The nodal temperature of both MPM and exact solutions are given in Fig. 10.20 at time $t = 0.05$. A constant time step of $\Delta t = 10^{-4}$ was adopted. Note again that, the particle temperature distribution (not shown) is not as smooth as the nodal temperature field. The M-file for this example is **example2D/mpmThermo2D.m**.

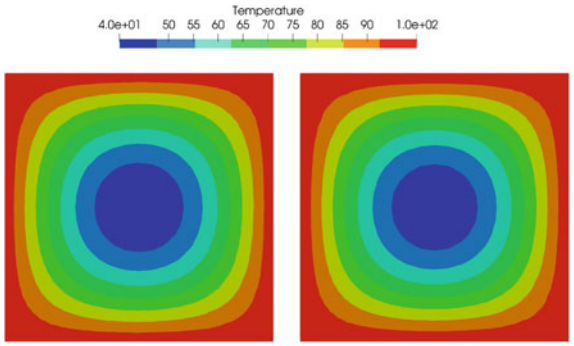


Fig. 10.20 Two dimensional heat conduction: ULMPM (hat functions) solutions (left) and exact solution (right) (de Vaucorbeil et al. 2020)

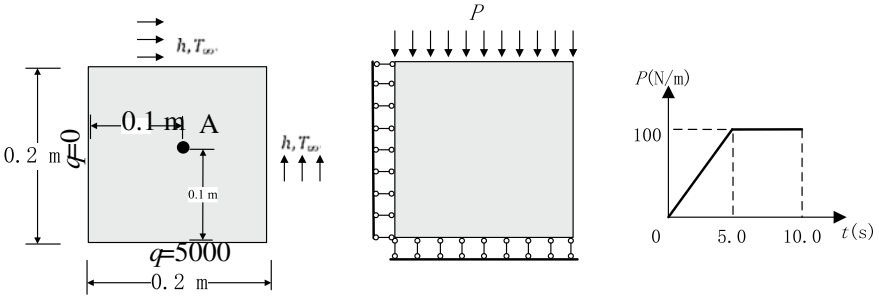


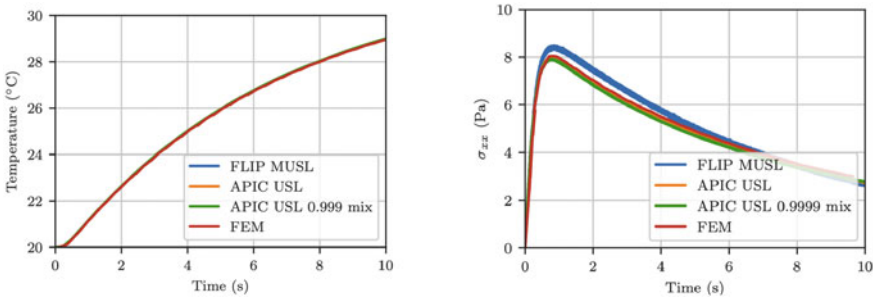
Fig. 10.21 Thermo-mechanical square plate: problem description adapted from Tao et al. (2016)

Table 10.2 Material parameters for the thermo-mechanical square plate problem

Mass density ρ	2100 kg/m^3	Heat conductivity k	500 W/m°C
Poisson's ratio ν	0.33	Heat capacity c	50 J/kg°C
Young's modulus E	70 GPa	Heat expansion α	25×10^{-8} /°C

Square plate with thermal and mechanical loadings. As the first test for the thermal-mechanical formulation, we consider the test proposed in Tao et al. (2016): a square plate with both thermal and mechanical boundary conditions (Fig. 10.21) is simulated. On the top and right surfaces, convection boundary conditions are applied with the convection coefficient $h = 2000$ W/(m^2 °C) and the ambient temperature $T_\infty = 30$ °C. The bottom side is subjected to a constant heat flux $q = 5000$ W/m^2 s and the left side is heat insulated. The initial temperature is 20 °C. In addition, a time-varying pressure is applied on the top surface. The square is made of a thermo-elastic material obeying Eq. 10.38. The material parameters are given in Table 10.2.

We refer to the M-file **example2D/mpmThermoMech2D.m** for a tutorial code for 2D coupled thermal-mechanical MPM simulations. However, we solve all the



(a) (b)

Fig. 10.22 Thermo-mechanical square plate: verification of ULMPM againts FEM solution

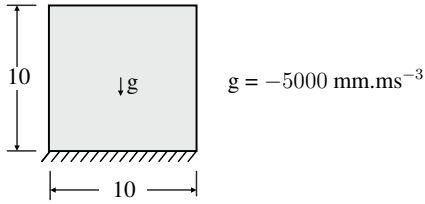


Fig. 10.23 Thermo-mechanical plaste under gravitational compression: problem description. Unit of length is mm

remaining tests in this section using `Karamelo` for the sake of efficiency and convenience.

To verify the ULMPM we monitor the temperature and stress of point A and compare these quantities against that obtained using FEM. The results shown in Fig. 10.22 indicate that the thermal-mechanical ULMPM algorithm works. However, this test only involves small deformation, which is not the application domain of the MPM. In what follows, we present another test for the thermal-mechanical MPM formulation which exhibits large deformation.

Plate under gravitational compression. The purpose of this example is to test the stability of the algorithm against cell crossing, as many of these events occur during the test. Additionally, it also aims to test the generation of heat from plastic strain. As no analytical solution exists, FEM is used as a reference, particularly the FEM result is obtained with the package Abaqus.

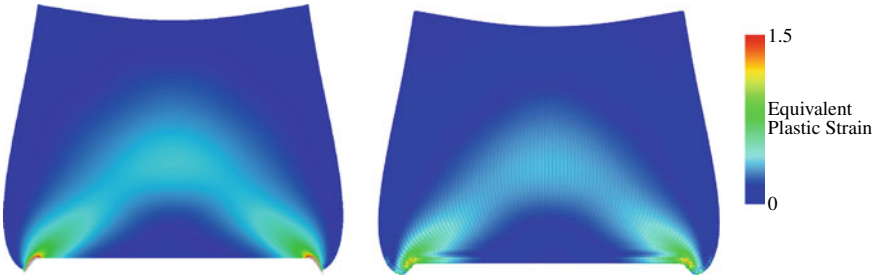
The test consists of a square plate which is fixed at the bottom and subjected to a gravitational force (Fig. 10.23). The sample is made of a ductile material modeled by the Johnson-Cook plastic model described in Chap. 4. The material parameters are given in Table 10.3.

In this test there is no external heat source. The only heat source is internal and it comes from the plastic deformation of the plate and is computed as follows. First, an increase of the plastic strain $\Delta\varepsilon_p$ leads to an increase of the temperature computed as:

$$\Delta T = \frac{\chi}{\rho C_p}\sigma_f \Delta\varepsilon_p \tag{10.43}$$

Table 10.3 Material parameters used for the plate under gravitational compression test

Material parameters		Flow stress params.		Params. for EOS	
Density ρ	8.94×10^{-6} kg/mm^3	A	65 MPa	c_0	3586 m/s
Young's modulus E	115 GPa	B	356 MPa	S_α	1.50
Poisson's ratio ν	0.31	C	0	Γ_0	0
Reference temperature T_0	0 °C	n	0.37		
Heat capacity c	384 J.kg^{-1}.K^{-1}	m	0		
Heat expansion α	1.67×10^{-5} K^{-1}	T_{melt}	1600 °C		
Heat conductivity k	3.86×10^{-4} kW.mm^{-1}.K^{-1}	$\dot{\varepsilon}$	1.0 s^{-1}		
Inelastic heat fraction χ	0.9				



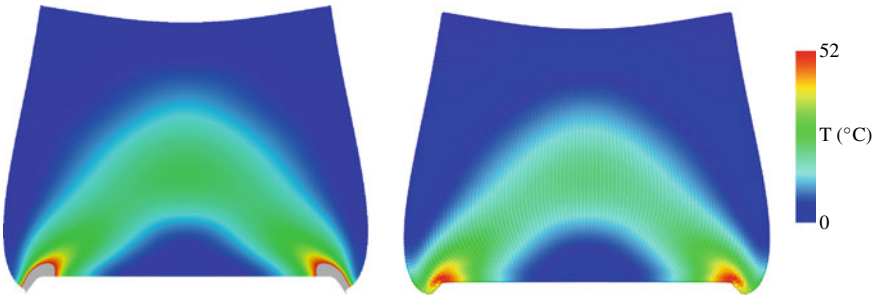
(a) FEM (b) ULMPM - FLIP MUSL

Fig. 10.24 Equivalent plastic strain in the plate under gravitational compression results at $t = 0.2$ ms

where $0 < \chi \leq 1$ is the Taylor-Quinney coefficient that determines how much the plastic work is converted into heat. For metals, $\chi = 0.9$ is often used. Then, ΔT is substituted into Eq. 10.27, the internal energy balance equation, (but without heat flux $\mathbf{q}$) to solve for the corresponding heat source. That heat source is thus given by

$$\gamma^* = \chi \sigma_f \dot{\varepsilon}_p \tag{10.44}$$

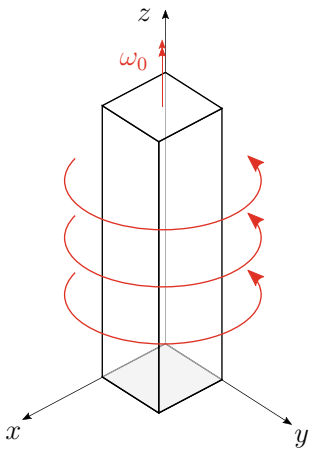
This example is simulated using ULMPM with cubic B-splines (there is no cell-crossing issue with TLMPM) and checked againts FEM results. The obtained equivalent plastic strain and temperature are given in Figs. 10.24 and 10.25, respectively. When the plate is compressed two shear bands are formed. The plastic deformation is localized along these bands, and thus heat is generated in this region. Qualitatively the ULMPM and FEM solutions are in good agreement. At the bottom corners, there are however some discrepancies likely due to boundary descriptions.



(a) FEM (b) ULMPM - FLIP MUSL

Fig. 10.25 Temperature in the plate under gravitational compression results at $t = 0.2$ ms

Fig. 10.26 Highly non-linear twisted column problem description: the bottom surface is fixed and an angular velocity $\omega_0 = 2\pi$ rad/ms is applied to the top surface



Twisted column. The main objective of this test is to assess the robustness of the proposed algorithms in large and highly nonlinear deformations. This problem is inspired by that introduced by Gil et al. (2014). The test consists of a column which is fixed at the bottom and an angular velocity $\omega_0 = 2\pi$ rad/ms is applied to the top surface i.e., the top surface does one rotation per millisecond. The column is 100 mm in length, square in cross-section with a side length of 10 mm. The material parameters for the Johnson-Cook plasticity model used are the same as the previous example (Table 10.3, Fig. 10.26).

The angular velocity ω_0 is applied by constraining the velocity of each and every node on the top surface of the column. Let n be the number of rotations we want to simulate and T is the final time, then we define $\omega = \omega_0 n/T$. The applied velocities as a function of time are therefore given by:

$$v_x(t) = -\omega y(t), \quad v_y(t) = +\omega x(t) \tag{10.45}$$

where $x(t)$ and $y(t)$ are the positions of the node along the x and y axes.

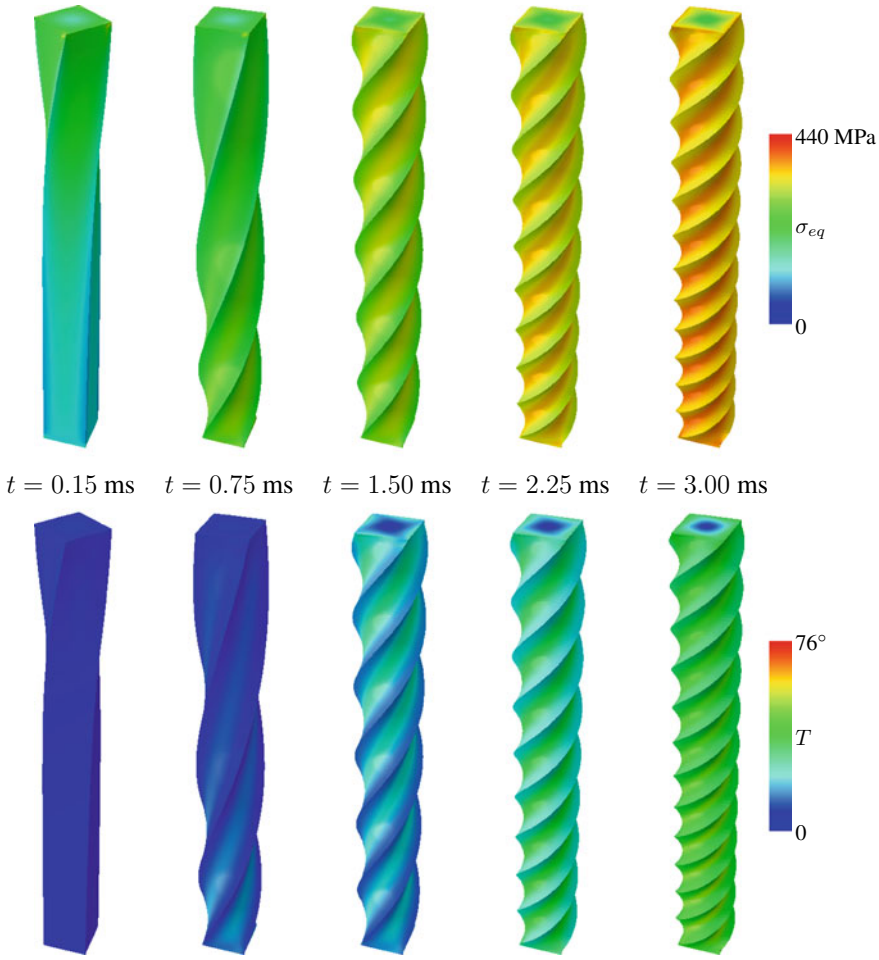


Fig. 10.27 Twisted column: sequence of deformed shape obtained with FEM

In Fig. 10.27 we present a sequence of deformed shape of the twisted column obtained with FEM. On the top is the contour plot of the von Mises equivalent stress and on the bottom is the contour plot of the temperature. And in Fig. 10.28 are the ULMPM results. The two solutions are qualitatively similar and the presented ULMPM algorithm for coupled thermal-mechanical problems exhibiting large deformation seems to be robust.

To conclude this section on thermal mechanical problems, we have presented a simple MPM algorithm for coupled thermal-mechanical simulations. We also have presented some simple tests to verify the formulation and the code. In the literature some researchers have applied similar MPM formulations to practical thermal

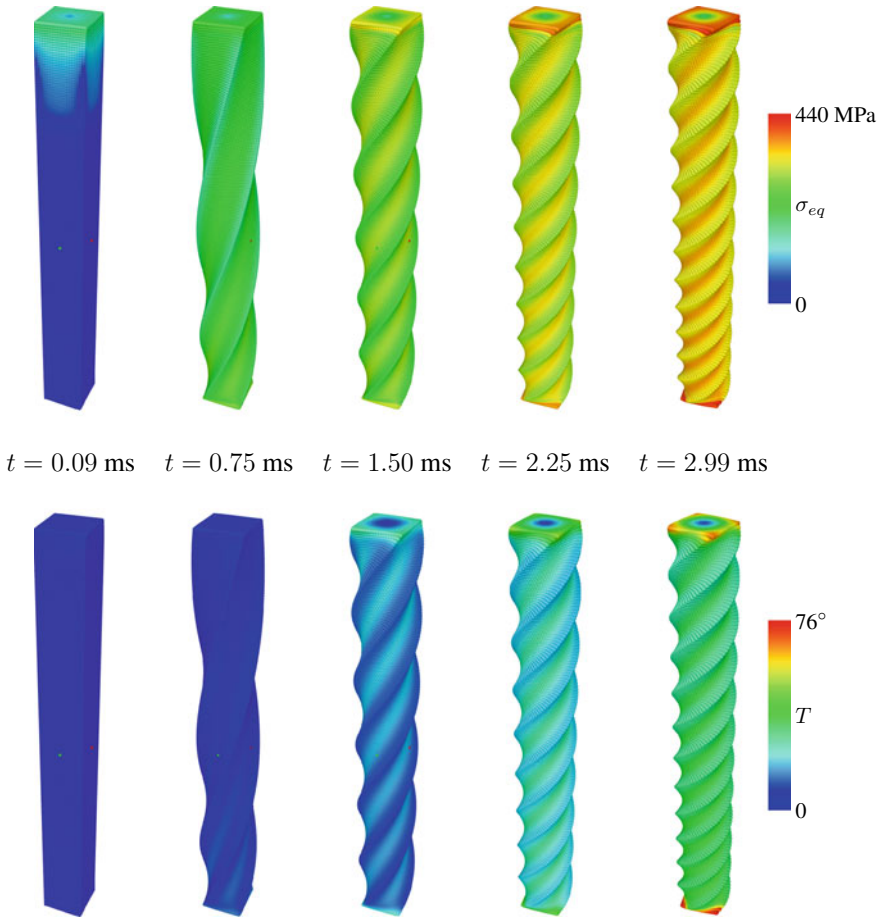


Fig. 10.28 Twisted column: sequence of deformed shape obtained with the ULMPM (cubic B-splines, MUSL)

mechanical problems. For example, among these works, Fagan et al. (2016) demonstrated that the MPM is able to simulate friction stir welding (FSW). FSW is a recent and complicated thermo-mechanical process used to join different materials which is still not well understood. And Lemiale et al. (2010) adopted the MPM to model a metal forming process called the equal channel angular pressing technique.

10.4 Fluid-Structure Interaction

Fluid-structure interaction (FSI) is a broad class of problems where the combined states of a fluid and a solid need to be determined simultaneously. Fluid structure interaction is of great relevance in many fields of engineering as well as in the applied sciences. Some examples are:

- 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, heart valve etc.) (Fig. 10.29)

For most FSI problems, analytical solutions to the model equations are impossible to obtain, whereas laboratory experiments are limited in scope. Therefore, in order to investigate the fundamental physics involved in the complex interaction between fluids and solids, numerical simulations may be employed. Since there is a strong need for effective fluid structure interaction analysis procedures, various approaches have been proposed.

Methods using only a Lagrangian formulation for both fluid and structure have become popular within the framework of meshfree particle methods such as SPH (Smoothed Particle Hydrodynamics), see e.g. Antoci et al. (2007). This popularity comes mainly from the simplicity of the method as the treatment of fluid and solid is nearly identical. However, it might not be the most efficient and accurate method for FSI problems.

Another method that is popular for FSI is the immersed boundary method (IB) developed by Peskin (1972). In this immersed boundary method, the fluid problem is solved on a fixed Eulerian grid and the solid is discretized by a Lagrangian grid. What is unique in this method is that the presence of the solid is accounted for by a force term in the fluid equation. Thus, there is no need to track the solid-fluid boundary Γ.

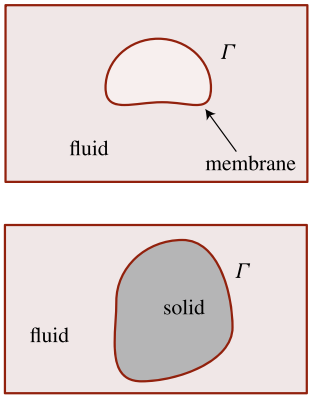


Fig. 10.29 Some examples of FSI: fluid-membrane interaction (e.g. blood flow in a heart valve) and fluid-solid interaction (e.g. suspended flow). These examples cover only situations where the membrane/solid is fully immersed in the fluid. Note that the structure (membrane/solid) is moving not stationary

We refer to the excellent review of different coupling methods and major codes/-softwares in the doctoral thesis of York (1997).

Modeling fluid dynamics and fluid-structure interactions is surprisingly scarse in the MPM literature given the fact that MPM can automatically handles interactions between the fluid and the structures. One can count York et al. (2000); Hu and Chen (2006); Gan et al. (2011) who studied fluid-membrane interactions based on MPM. In their implementations, the fluid and the membrane are solved together using a single background grid i.e., a monolithic approach. The interaction between the fluid and the structure is indirect through the background grid. Therefore, there is no need to track the evolving fluid-membrane interface. As shall be shown this approach is straightforward to implement since it is identical to the interaction of multiple solid bodies. Both solid and fluid are discretized by material points which follow their own constitutive behaviours. However only non-slip boundary conditions at the fluid-membrane interface can be handled. Guilkey et al. (2007) also used MPM to model the fluid-structure interactions, but they used a finite volume scheme for fluid. A coupled FEM-MPM formulation for FSI was presented in Chen et al. (2015) where FEM is used to model the elastic structures and MPM is for the fluid.

It is clear that the literature on the MPM for FSI problems is scarce. As we do not have much experience on this field yet, we only presented an overview of FSI modeling herein.

References

Andersen, S., Andersen, L.: Analysis of spatial interpolation in the material-point method. Comput. Struct. **88**(7–8), 506–518 (2010)

Antoci, C., Gallati, M., Sibilla, S.: Numerical simulation of fluid-structure interaction by SPH. Comput. Struct. **85**(11–14), 879–890 (2007)

Chen, Z., Gan, Y., Chen, J.K.: A coupled thermo-mechanical model for simulating the material failure evolution due to localized heating. Comput. Model. Eng. Sci. **26**(2), 123 (2008)

Chen, Z.P., Qiu, X.M., Zhang, X., Lian, Y.P.: Improved coupling of finite element method with material point method based on a particle-to-surface contact algorithm. Comput. Methods Appl. Mech. Eng. **293**, 1–19 (2015)

Cueto-Felgueroso, L., Colominas, I., Mosqueira, G., Navarrina, F., Casteleiro, M.: On the Galerkin formulation of the smoothed particle hydrodynamics method. Int. J. Numer. Meth. Eng. **60**(9), 1475–1512 (2004)

de Vaucorbeil, A., Nguyen, V. P., Sinaie, S., Wu, J. Y.: Chapter two—material point method after 25 years: theory, implementation, and applications. In: Advances in Applied Mechanics, vol. 53, pp. 185–398. Elsevier (2020)

Fagan, Timothy, Lemiale, Vincent, Nairn, John, Ahuja, Yogita, Ibrahim, Raafat, Estrin, Yuri: Detailed thermal and material flow analyses of friction stir forming using a three-dimensional particle based model. J. Mater. Process. Technol. **231**, 422–430 (2016)

Gan, Y., Chen, Z., Montgomery-Smith, S.: Improved material point method for simulating the zona failure response in piezo-assisted intracytoplasmic sperm injection. Comput. Model. Eng. Sci. 1–24 (2011)

Gil, A.J., Lee, C.H., Bonet, J., Aguirre, M.: A stabilised Petrov-Galerkin formulation for linear tetrahedral elements in compressible, nearly incompressible and truly incompressible fast dynamics.

Comput. Methods Appl. Mech. Eng. **276**, 659–690 (2014). https://doi.org/10.1016/j.cma.2014.04.006

Guilkey, J.E., Harman, T.B., Banerjee, B.: An Eulerian-Lagrangian approach for simulating explosions of energetic devices. Comput. Struct. **85**(11–14), 660–674 (2007)

Hamad, F., Stolle, D., Vermeer, P.: Modelling of membranes in the material point method with applications. Int. J. Numer. Anal. Meth. Geomech. **39**(8), 833–853 (2015)

Hu, W., Chen, Z.: Model-based simulation of the synergistic effects of blast and fragmentation on a concrete wall using the MPM. Int. J. Impact Eng. **32**(12), 2066–2096 (2006)

Lemiale, V., Nairn, J., Hurmane, A.: Material point method simulation of equal channel angular pressing involving large plastic strain and contact through sharp corners. Comput. Model. Eng. Sci. **70**(1), 41–66 (2010)

Lian, Y.P., Zhang, X., Zhou, X., Ma, Z.T.: A FEMP method and its application in modeling dynamic response of reinforced concrete subjected to impact loading. Comput. Methods Appl. Mech. Eng. **200**(17–20), 1659–1670 (2011)

Lobovskỳ, Libor, Botia-Vera, Elkin, Castellana, Filippo, Mas-Soler, Jordi, Souto-Iglesias, Antonio: Experimental investigation of dynamic pressure loads during dam break. J. Fluids Struct. **48**, 407–434 (2014)

Mao, S.: Material point method and adaptive meshing applied to fluid-structure interaction (FSI) problems. In: ASME 2013 Fluids Engineering Division Summer Meeting, pp. V01BT13A004–V01BT13A004. American Society of Mechanical Engineers (2013)

Mast, C.M., Mackenzie-Helnwein, P., Arduino, P., Miller, G.R., Shin, W.: Mitigating kinematic locking in the material point method. J. Comput. Phys. **231**(16), 5351–5373 (2012)

Monaghan, J.J.: Simulating free surface flows with SPH. J. Comput. Phys. **110**(2), 399–406 (1994)

Nairn, J.A., Guilkey, J.E.: Axisymmetric form of the generalized interpolation material point method. Int. J. Numer. Meth. Eng. **101**(2), 127–147 (2015)

Nguyen, V. P., Nguyen, C. T., Rabczuk, T., Natarajan, S.: On a family of convected particle domain interpolations in the material point method. Finite Elem. Anal. Des. **126**, 50–64 (2017)

Peskin, C.S.: Flow patterns around heart valves: a numerical method. J. Comput. Phys. **10**(2), 252–271 (1972)

Sod, G.A.: A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. J. Comput. Phys. **27**(1), 1–31 (1978)

Su, Y. C., Tao, J., Jiang, S., Chen, Z., Lu, J.M.: Study on the fully coupled thermodynamic fluid–structure interaction with the material point method. Comput. Part. Mech. 1–16 (2019)

Sun, Z., Li, H., Gan, Y., Liu, H., Huang, Z., He, L.: Material point method and smoothed particle hydrodynamics simulations of fluid flow problems: a comparative study. Prog. Comput. Fluid Dyn., Int. J. (PCFD) **18**(1), 1–18 (2018)

Tao, Jun, Zheng, Yonggang, Chen, Zhen, Zhang, Hongwu: Generalized interpolation material point method for coupled thermo-mechanical processes. Int. J. Mech. Mater. Des. **12**(4), 577–595 (2016)

Yang, W. C., Arduino, P., Miller, G.R., Mackenzie-Helnwein, P.: Smoothing algorithm for stabilization of the material point method for fluid–solid interaction problems. Comput. Methods Appl. Mech. Eng. **342**, 177–199 (2018)

York, A.R., Sulsky, D., Schreyer, H.L.: Fluid-membrane interaction based on the material point method. Int. J. Numer. Methods Eng. 901–924 (2000)

York, A.R.: Development of modifications to the material point method for the simulation of thin membranes, compressible fluids, and their interactions. Ph.D. thesis, The University of New Mexico, Albuquerque (1997)

York, A.R., Sulsky, D., Schreyer, H.L.: The material point method for simulation of thin membranes. Int. J. Numer. Meth. Eng. **44**(10), 1429–1456 (1999)

Zhang, X., Chen, Z., Liu, Y.: The Material Point Method: A Continuum-Based Particle Method for Extreme Loading Cases. Academic Press (2016)

Zhang, F., Zhang, X., Sze, K. Y., Lian, Y., Liu, Y.: Incompressible material point method for free surface flow. J. Comput. Phys. **330**, 92–110 (2017)

Appendix A
Strong Form, Weak Form and Completeness

This appendix presents, for completeness, the equivalence of the strong form and weak form (Sect. A.1). Section A.2 briefly recalls the completeness requirement of FE shape functions.

A.1 Weak Formulation

Herein we present the derivation of the weak form given in Sect. 2.3 (Sect. A.1.1) and demonstrate the equivalence of the weak form and strong form by getting the strong form from the weak form (Sect. A.1.2).

A.1.1 Strong Form to Weak Form

The construction of the weak form starts by multiplying the momentum equation with the test function δv_i and integrating over the current configuration. That is

$$\int_{\Omega} \delta v_i \left(\rho \frac{\partial \sigma_{ij}^s}{\partial x_j} + \rho b_i - \rho \dot{v}_i \right) \mathrm{d}\Omega = 0 \tag{A.1}$$

With the notation $f_{,j} = {}^{\partial f}/_{\partial x_j}$ and the identity $(\delta v_i \sigma_{ij}^s)_{,j} = \delta v_{i,j} \sigma_{ij}^s + \delta v_i \sigma_{ij,j}^s$, we can write the first term in the above equation as

$$\int_{\Omega} \rho \delta v_i \frac{\partial \sigma_{ij}^s}{\partial x_j} d\Omega = \int_{\Omega} \rho \left(\frac{\partial}{\partial x_j} (\delta v_i \sigma_{ij}^s) - \frac{\partial \delta v_i}{\partial x_j} \sigma_{ij} \right) \mathrm{d}\Omega \tag{A.2}$$

Using the Gauss theorem, we can write

N. V. Phu et al., *The Material Point Method*, Scientific Computation,
https://doi.org/10.1007/978-3-031-24070-6

$$\int_{\Omega} \frac{\partial}{\partial x_j}(\delta v_i \sigma_{ij}^s)\mathrm{d}\Omega = \int_{\Gamma_{\text{int}}} \delta v_i [[n_j \sigma_{ij}^s]] d\Gamma + \int_{\Gamma} \delta v_i n_j \sigma_{ij}^s \mathrm{d}\Gamma \tag{A.3}$$

From the traction continuity on internal boundaries Γ_{int} e.g. cracks, the first term on the RHS vanishes. For the second integrand, using the traction boundary condition $n_j \sigma_{ij}^s = \bar{t}_i^s$ and the fact that δv_i vanishes on the complement of Γ_t, the above equation becomes

$$\int_{\Omega} \frac{\partial}{\partial x_j}(\delta v_i \sigma_{ij}^s)\mathrm{d}\Omega = \int_{\Gamma_t} \delta v_i \bar{t}_i^s \mathrm{d}\Gamma \tag{A.4}$$

Substituting Eq. (A.4) into Eq. (A.2) gives

$$\int_{\Omega} \rho \delta v_i \frac{\partial \sigma_{ij}^s}{\partial x_j}\mathrm{d}\Omega = \int_{\Gamma_t} \delta v_i \bar{t}_i^s \mathrm{d}\Gamma - \int_{\Omega} \frac{\partial \delta v_i}{\partial x_j} \sigma_{ij}^s \mathrm{d}\Omega \tag{A.5}$$

Introducing Eq. (A.5) into Eq. (A.1) yields

$$\int_{\Omega} \rho \frac{\partial \delta v_i}{\partial x_j} \sigma_{ij}^s d\Omega - \int_{\Gamma_t} \rho \delta v_i \bar{t}_i^s d\Gamma - \int_{\Omega} \delta v_i \rho b_i d\Omega + \int_{\Omega} \delta v_i \rho \dot{v}_i \mathrm{d}\Omega = 0 \tag{A.6}$$

Instead of thinking the test function δv_i as a mathematical quantity, if we consider it as a virtual velocity, then each term in the above equation represents a virtual power. Therefore, this equation is named the *virtual power equation*, which is written as: $\delta P^{\text{int}} - \delta P^{\text{ext}} + \delta P^{\text{kin}} = 0$, where

$$\begin{aligned}
\delta P^{\text{int}} &= \int_{\Omega} \nabla \delta \mathbf{v} : \boldsymbol{\sigma} \mathrm{d}\Omega \\
\delta P^{\text{ext}} &= \int_{\Omega} \rho \mathbf{b} \cdot \delta \mathbf{v} \mathrm{d}\Omega + \int_{\Gamma_t} \bar{\mathbf{t}} \cdot \delta \mathbf{v} \mathrm{d}\Gamma \\
\delta P^{\text{kin}} &= \int_{\Omega} \rho \delta \mathbf{v} \cdot \dot{\mathbf{v}} \mathrm{d}\Omega
\end{aligned}$$

where δP^{ext} represents the virtual external power—the power done by external forces and δP^{kin} is the virtual kinetic power.

To demonstrate that δP^{int} is a virtual internal power, let us rewrite $\frac{\partial \delta v_i}{\partial x_j}\sigma_{ij}$ as follows

$$\begin{aligned}
\frac{\partial \delta v_i}{\partial x_j}\sigma_{ij} &= \delta v_{i,j}\sigma_{ij} = \delta L_{ij}\sigma_{ij} = (\delta D_{ij} + \delta W_{ij})\sigma_{ij} \\
&= \delta D_{ij}\sigma_{ij} = \delta \mathbf{D} : \boldsymbol{\sigma}
\end{aligned} \tag{A.7}$$

And from the conservation of energy Eq. (2.19), $\delta D_{ij}\sigma_{ij}$ is the virtual internal power per unit volume.

A.1.2 Weak Form to Strong Form

After spending efforts to derive this weak form, a question arises naturally. Is this weak form equivalent to the strong form from which it was derived? The answer is yes and to this end, consider the first term in Eq. (A.6)

$$\begin{aligned}\int_\Omega \rho \frac{\partial \delta v_i}{\partial x_j}\sigma_{ij}^s \mathrm{d}\Omega &= \int_\Omega \rho \frac{\partial \delta v_i \sigma_{ij}^s}{\partial x_j}\mathrm{d}\Omega - \int_\Omega \rho \delta v_i \frac{\partial \sigma_{ij}^s}{\partial x_j}\mathrm{d}\Omega \\ &= \int_\Gamma \rho \delta v_i n_j \sigma_{ij}^s \mathrm{d}\Gamma - \int_\Omega \rho \delta v_i \frac{\partial \sigma_{ij}^s}{\partial x_j}\mathrm{d}\Omega\end{aligned} \tag{A.8}$$

which after substituted into Eq. (A.6) yields

$$\int_\Gamma \delta v_i (\rho \bar{t}_i^s - \rho n_j \sigma_{ij}^s)\mathrm{d}\Gamma - \int_\Omega \delta v_i \left(\rho \frac{\partial \sigma_{ij}^s}{\partial x_j} + \rho b_i - \rho \dot{v}_i\right)\mathrm{d}\Omega = 0 \tag{A.9}$$

As the above has to be true for any δv_i, we then have, using the fundamental lemma of variational calculus

$$\bar{t}_i = n_j \sigma_{ij} \tag{A.10a}$$

$$\frac{\partial \sigma_{ij}}{\partial x_j} + \rho b_i - \rho \dot{v}_i = 0 \tag{A.10b}$$

which are apparently the traction and the momentum equations.

A.2 Completeness

A variational index is the highest spatial derivative order of the displacement or velocity in the weak form. As the variational index of the weak form in Eq. (A.6) is one, the basis functions must be 1-complete. That is, they must represent exactly all polynomial terms of order ≤ 1 in the Cartesian coordinates.

Let us consider a 2D linear displacement field (a polynomial of order 1)

$$u_x = \alpha_0 + \alpha_1 x + \alpha_2 y, \quad u_y = \beta_0 + \beta_1 x + \beta_2 y, \tag{A.11}$$

where α_i and β_i are non-zero constants.

To show that the basis functions can represent exactly all polynomial terms of order ≤ 1 in the Cartesian coordinates, we first assign the nodal displacements values according to Eq. (A.11). For example, the x-component of the nodal displacements is given by

$$u_{xI} = \alpha_0 + \alpha_1 x_I + \alpha_2 y_I \tag{A.12}$$

Then, we check if the FE displacement field can reproduce a linear field. We check this for the x component. The FE displacement $u_x^h(x, y)$ is given by

$$\begin{aligned} u_x^h &= \sum_I N_I u_{xI} = \sum_I N_I(\alpha_0 + \alpha_1 x_I + \alpha_2 y_I) \quad \text{(use Eq. (A.12))} \\ &= \alpha_0 \sum_I N_I + \alpha_1 \sum_I N_I x_I + \alpha_2 \sum_I N_I y_I \end{aligned} \tag{A.13}$$

Now, we have

$$\sum_I N_I(\mathbf{x}) = 1, \quad \sum_I N_I x_I = x, \quad \sum_I N_I y_I = y \tag{A.14}$$

Thus,

$$u_x^h = \alpha_0 + \alpha_1 x + \alpha_2 y \tag{A.15}$$

Indeed the FE displacement field can reproduce a linear field. As a special case, it can also reproduce a constant field.

Appendix B
Derivation of CPDI Basis Functions

This appendix provides the derivation of some results presented in Sect. 3.6 on the CPDI functions for two-node line elements (Sect. B.1), three-node line elements (Sect. B.2), four-node quadrilateral elements (Sect. B.3), three-node triangle elements (Sect. B.4), and four-node tetrahedron elements (Sect. B.5). Recall that the CPDI basis functions are given by

$$\phi_{Ip} = \frac{1}{V_p} \sum_{c=1}^{n} \left[\int_{\Omega_p} M_c(\mathbf{x}) \mathrm{d}\Omega \right] N_I(\mathbf{x}_c) \tag{B.1}$$

where $M_c(\mathbf{x})$ are the shape functions of the finite element used to represent the particle domain and n denotes the number of nodes of this element. In what follows, the exact integration of the term in brackets is presented for various CPDIs. In subsequent development, we drop the subscript p and without confusing we use N_c in place of M_c.

B.1 CPDI-L2 Basis

As the shape functions of two-node line elements (L2 elements) are defined in a parent domain $\square = [-1, 1]$ in terms of the so-called natural coordinates (ξ), we can thus write the following

$$\int_{\Omega} N_I(\mathbf{x}) \mathrm{d}\Omega = \int_{\square} N_I(\xi) |J| \mathrm{d}\square = \int_{-1}^{+1} N_I(\xi) |J| \mathrm{d}\xi \tag{B.2}$$

where $|J|$ denotes the determinant of the Jacobian of the transformation between the global coordinate system and the parent domain which is given by

N. V. Phu et al., *The Material Point Method*, Scientific Computation,
https://doi.org/10.1007/978-3-031-24070-6

$$|J| = x_{,\xi} = N_{1,\xi}x_1 + N_{2,\xi}x_2 = 0.5(x_2 - x_1) = \frac{l_p}{2} \tag{B.3}$$

where l_p denotes the particle length. Therefore, the integral of N_I over Ω–the particle domain is written as

$$\int_\Omega N_I(\mathbf{x})\mathrm{d}\Omega = \frac{l_p}{2}\int_{-1}^{+1} N_I(\xi)\mathrm{d}\xi \tag{B.4}$$

And specially we have

$$\int_\Omega N_1(\mathbf{x})\mathrm{d}\Omega = \frac{l_p}{2}\int_{-1}^{+1} N_1(\xi)\mathrm{d}\xi = \frac{l_p}{2}\int_{-1}^{+1}\frac{1-\xi}{2}\mathrm{d}\xi = \frac{l_p}{2} \tag{B.5}$$

$$\int_\Omega N_2(\mathbf{x})\mathrm{d}\Omega = \frac{l_p}{2}\int_{-1}^{+1} N_2(\xi)\mathrm{d}\xi = \frac{l_p}{2}\int_{-1}^{+1}\frac{1+\xi}{2}\mathrm{d}\xi = \frac{l_p}{2} \tag{B.6}$$

Finally, the CPDI-L2 shape functions are given by

$$\phi_{Ip} = \frac{1}{l_p}\left(\frac{l_p}{2}N_I(x_1) + \frac{l_p}{2}N_I(x_2)\right) = \frac{1}{2}N_I(x_1) + \frac{1}{2}N_I(x_2) \tag{B.7}$$

And for completeness, the CPDI-L2 derivatives are given by

$$d\phi_{Ip} = -\frac{1}{l_p}N_I(x_1) + \frac{1}{l_p}N_I(x_2) \tag{B.8}$$

B.2 CPDI-L3 Basis

The shape functions and first derivatives of a quadratic line element are given by

$$\begin{array}{ll} N_1 = -0.5(1-\xi)\xi, & dN_1 = \xi - 0.5 \\ N_2 = +0.5(1+\xi)\xi, & dN_2 = \xi + 0.5 \\ N_3 = 1 - \xi^2, & dN_3 = -2\xi \end{array} \tag{B.9}$$

where node 3 is the midside node. The Jacobian is defined by

$$|J| = x_{,\xi} = N_{I,\xi}x_I = c\xi + 0.5x_{21} \tag{B.10}$$

where $x_{21} = x_2 - x_1$ and $c = (x_1 + x_2 - 2x_3)$. Therefore

$$\begin{aligned}
\int_{\Omega} N_1(\mathbf{x})\mathrm{d}\Omega &= \int_{-1}^{1} -0.5(1-\xi)\xi[c\xi + 0.5x_{21}]\mathrm{d}\xi = \frac{1}{6}(x_{21} - 2c) \\
\int_{\Omega} N_2(\mathbf{x})\mathrm{d}\Omega &= \int_{-1}^{1} 0.5(1+\xi)\xi[c\xi + 0.5x_{21}]\mathrm{d}\xi = \frac{1}{6}(x_{21} + 2c) \\
\int_{\Omega} N_3(\mathbf{x})\mathrm{d}\Omega &= \int_{-1}^{1} (1-\xi^2)\,[c\xi + 0.5x_{21}]\mathrm{d}\xi = \frac{4}{6}x_{21}
\end{aligned} \tag{B.11}$$

One can thus collectively gather the function weights into a vector

$$wf = \left(\frac{1}{6l_p}(x_{21} - 2c)\ \frac{1}{6l_p}(x_{21} + 2c)\ \frac{4}{6l_p}(x_{21})\right) \tag{B.12}$$

In the same manner, we can gather the gradient weights into a vector as

$$wg = \begin{pmatrix} -1 & 1 & 0 \end{pmatrix} \tag{B.13}$$

If the midside node is chosen to be identical to the particle position, then one have $x_3 = 0.5(x_1 + x_2)$ or $c = 0$, and noting that $x_{21} = l_p$,

$$wf = \left(\frac{1}{6}\ \frac{1}{6}\ \frac{4}{6}\right) \tag{B.14}$$

B.3 CPDI-Q4 Basis

The four shape functions of a Q4 element are given by

$$\begin{aligned}
N_1(\xi, \eta) &= \frac{1}{4}(1-\xi)(1-\eta) \\
N_2(\xi, \eta) &= \frac{1}{4}(1+\xi)(1-\eta) \\
N_3(\xi, \eta) &= \frac{1}{4}(1+\xi)(1+\eta) \\
N_4(\xi, \eta) &= \frac{1}{4}(1-\xi)(1+\eta)
\end{aligned} \tag{B.15}$$

The shape functions of Q4 elements are defined in a parent domain $\square = [-1, 1] \times [-1, 1]$ in terms of the so-called natural coordinates (ξ, η), see Fig. B.1. One can thus write the following

$$\int_{\Omega} N_I(\mathbf{x})\mathrm{d}\Omega = \int_{\square} N_I(\boldsymbol{\xi})|J|\mathrm{d}\square = \int_{-1}^{+1}\int_{-1}^{+1} N_I(\xi, \eta)|J|\mathrm{d}\xi\mathrm{d}\eta \tag{B.16}$$

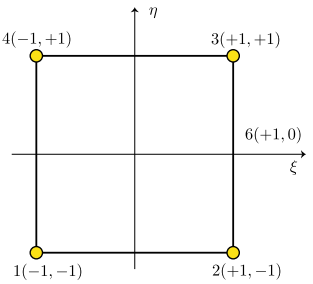


Fig. B.1 The four-node quadrilateral element in the natural coordinate system

where $|J|$ is given by

$$|J| = x_{,\xi}\, y_{,\eta} - y_{,\xi}\, x_{,\eta} \tag{B.17}$$

With $x = N_I(\xi, \eta)x_I$ and $y = N_I(\xi, \eta)y_I$, we obtain

$$\begin{aligned} x_{,\xi} &= \frac{1}{4}\left[(1-\eta)x_{21} + (1+\eta)x_{34}\right] \\ x_{,\eta} &= \frac{1}{4}\left[(1-\xi)x_{41} + (1+\xi)x_{32}\right] \end{aligned} \tag{B.18}$$

where $x_{ij} = x_i - x_j$ and $y_{ij} = y_i - y_j$ are the corner coordinate differences. The expressions for $y_{,\xi}$ and $y_{,\eta}$ follow Eq. (B.18) with y replacing x. Now, we can write

$$\begin{aligned} |J| = \frac{1}{16}\big[&(1-\eta)(1-\xi)(x_{21}y_{41} - y_{21}x_{41}) \\ +\ &(1-\eta)(1+\xi)(x_{21}y_{32} - y_{21}x_{32}) \\ +\ &(1+\eta)(1-\xi)(x_{34}y_{41} - y_{34}x_{41}) \\ +\ &(1+\eta)(1+\xi)(x_{34}y_{32} - y_{34}x_{32})\big] \end{aligned} \tag{B.19}$$

Let take N_1 as an example and by using Eqs. (B.16), (B.19), we write

$$\begin{aligned} \int_\Omega N_1(\mathbf{x})\mathrm{d}\Omega = \frac{1}{64}\int_{-1}^{+1}\int_{-1}^{+1}(1-\xi)(1-\eta)\,\big[&(1-\eta)(1-\xi)c_1 + (1-\eta)(1+\xi)c_2 \\ &+(1+\eta)(1-\xi)c_3 + (1+\eta)(1+\xi)c_4\big]\,\mathrm{d}\xi\mathrm{d}\eta \end{aligned} \tag{B.20}$$

where $c_1 = (x_{21}y_{41} - y_{21}x_{41})$, $c_2 = (x_{21}y_{32} - y_{21}x_{32})$, $c_3 = (x_{34}y_{41} - y_{34}x_{41})$ and $c_4 = (x_{34}y_{32} - y_{34}x_{32})$. The above can be more elaborated as

$$\int_{\Omega} N_1(\mathbf{x})\mathrm{d}\Omega = \frac{1}{64}\Big[c_1\int_{-1}^{+1}\int_{-1}^{+1}(1-\xi)^2(1-\eta)^2\mathrm{d}\xi\mathrm{d}\eta + c_2\int_{-1}^{+1}\int_{-1}^{+1}(1-\xi^2)(1-\eta)^2\mathrm{d}\xi\mathrm{d}\eta$$
$$c_3\int_{-1}^{+1}\int_{-1}^{+1}(1-\xi)^2(1-\eta^2)\mathrm{d}\xi\mathrm{d}\eta + c_4\int_{-1}^{+1}\int_{-1}^{+1}(1-\xi^2)(1-\eta^2)\mathrm{d}\xi\mathrm{d}\eta\Big] \tag{B.21}$$

in which all the integrals can be exactly computed and the final result is

$$\int_{\Omega} N_1(\mathbf{x})\mathrm{d}\Omega = \frac{1}{36}(4c_1 + 2c_2 + 2c_3 + c_4) \tag{B.22}$$

$$\int_{\Omega} N_2(\mathbf{x})\mathrm{d}\Omega = \frac{1}{36}(2c_1 + 4c_2 + c_3 + 2c_4) \tag{B.23}$$

$$\int_{\Omega} N_3(\mathbf{x})\mathrm{d}\Omega = \frac{1}{36}(c_1 + 2c_2 + 2c_3 + 4c_4) \tag{B.24}$$

$$\int_{\Omega} N_4(\mathbf{x})\mathrm{d}\Omega = \frac{1}{36}(2c_1 + c_2 + 4c_3 + 2c_4) \tag{B.25}$$

Using the chain rule, we can compute the derivatives of the shape function with respect to the global coordinates as follows

$$\begin{aligned}\begin{bmatrix}N_{I,x} & N_{I,y}\end{bmatrix} &= \begin{bmatrix}N_{I,\xi} & N_{I,\eta}\end{bmatrix}\begin{bmatrix}x_{,\xi} & x_{,\eta}\\ y_{,\xi} & y_{,\eta}\end{bmatrix}^{-1}\\ &= \frac{1}{|J|}\begin{bmatrix}N_{I,\xi} & N_{I,\eta}\end{bmatrix}\begin{bmatrix}y_{,\eta} & -x_{,\eta}\\ -y_{,\xi} & x_{,\xi}\end{bmatrix}\end{aligned} \tag{B.26}$$

Or, explicitly we can write

$$\begin{aligned}N_{I,x} &= \frac{1}{|J|}\left(N_{I,\xi}y_{,\eta} - N_{I,\eta}y_{,\xi}\right)\\ N_{I,y} &= \frac{1}{|J|}\left(-N_{I,\xi}x_{,\eta} + N_{I,\eta}x_{,\xi}\right)\end{aligned} \tag{B.27}$$

which allows us to write

$$\int_{\Omega} N_{I,x}\mathrm{d}\Omega = \int_{\square} N_{I,x}|J|\mathrm{d}\square = \int_{\square}\left(N_{I,\xi}y_{,\eta} - N_{I,\eta}y_{,\xi}\right)\mathrm{d}\square \tag{B.28}$$

Using the above equation for N_1 one gets

$$\int_{\Omega} N_{1,x} \mathrm{d}\Omega = \frac{1}{16} \int_{\square} \Big[-(1-\eta)\left[(1-\xi)y_{41} + (1+\xi)y_{32}\right] + (1-\xi)\Big[(1-\eta)y_{21} + (1+\eta)y_{34}\Big]\Big] \mathrm{d}\square = \frac{1}{2} y_{24} \tag{B.29}$$

$$\int_{\Omega} N_{1,y} \mathrm{d}\Omega = \frac{1}{16} \int_{\square} \Big[(1-\eta)\left[(1-\xi)x_{41} + (1+\xi)x_{32}\right] - (1-\xi)\Big[(1-\eta)x_{21} + (1+\eta)x_{34}\Big]\Big] \mathrm{d}\square = \frac{1}{2} x_{42} \tag{B.30}$$

B.4 Derivation of CPDI-T3 Basis

The three shape functions of a linear triangular element are written as

$$\begin{aligned} M_1(\xi,\eta) &= 1-\xi-\eta \\ M_2(\xi,\eta) &= \xi \\ M_3(\xi,\eta) &= \eta \end{aligned} \tag{B.31}$$

The Jacobian of the transformation is given by

$$|J| = x_{,\xi} y_{,\eta} - y_{,\xi} x_{,\eta} = x_{21} y_{31} - y_{21} x_{31} \tag{B.32}$$

Note that $|J|$ is constant and $|J| = 2A$ where A is the area of the element.

We therefore can compute the integrals of the shape functions as follows

$$\begin{aligned} \int_{\Omega} N_1(\mathbf{x}) \mathrm{d}\Omega &= |J| \int_0^1 \int_0^{1-\eta} (1-\xi-\eta) \mathrm{d}\xi \mathrm{d}\eta = \frac{|J|}{6} \\ \int_{\Omega} N_2(\mathbf{x}) \mathrm{d}\Omega &= |J| \int_0^1 \int_0^{1-\eta} (\xi) \mathrm{d}\xi \mathrm{d}\eta = \frac{|J|}{6} \\ \int_{\Omega} N_3(\mathbf{x}) \mathrm{d}\Omega &= |J| \int_0^1 \int_0^{1-\eta} (\eta) \mathrm{d}\xi \mathrm{d}\eta = \frac{|J|}{6} \end{aligned} \tag{B.33}$$

The derivatives of the shape functions are written as

$$
\begin{aligned}
N_{1,x} &= \frac{1}{|J|}(y_{23}), \quad N_{1,y} = \frac{1}{|J|}(x_{32}) \\
N_{2,x} &= \frac{1}{|J|}(-y_{13}), \quad N_{2,y} = \frac{1}{|J|}(x_{13}) \\
N_{3,x} &= \frac{1}{|J|}(y_{12}), \quad N_{3,y} = \frac{1}{|J|}(-x_{12})
\end{aligned}
\tag{B.34}
$$

which are constants so that the integrals of the shape function derivatives are trivial. For example, we have

$$
\int_{\Omega} N_{1,x} \mathrm{d}\Omega = \int y_{23} \mathrm{d}\xi \mathrm{d}\eta = \frac{1}{2} y_{23} \tag{B.35}
$$

Putting all together the CPDI-T3 basis functions and first derivatives are given by

$$
\begin{aligned}
\phi_{Ip} &= \frac{1}{3}\Big[N_I(x_1) + N_I(x_2) + N_I(x_3)\Big] \\
\phi_{Ip,x} &= \frac{1}{2A}\left(y_{23}N_I(x_1) - y_{13}N_I(x_2) + y_{12}N_I(x_3)\right) \\
\phi_{Ip,y} &= \frac{1}{2A}\left(x_{32}N_I(x_1) + x_{13}N_I(x_2) - x_{12}N_I(x_3)\right)
\end{aligned}
\tag{B.36}
$$

from which the function weights and gradient weights can be extracted.

B.5 Derivation of CPDI-Tet4 Basis

The four shape functions of linear tetrahedron elements are $\xi_i, i = 1, 2, 3, 4$ with $\xi_1 + \xi_2 + \xi_3 + \xi_4 = 1$. For analytical integration over the linear tetraheron can be done by using the following general formula (Felippa 2022)

$$
\int_{\Omega} \xi_1^i \xi_2^j \xi_3^k \xi_4^l \mathrm{d}\Omega = \frac{i!j!k!l!}{(i+j+k+l+3)!} 6V \tag{B.37}
$$

where V is the volume of the tetraheron of which expression is given in the text. By using Eq. (B.37) one can write

$$
\int_{\Omega} M_c \mathrm{d}\Omega = \int_{\Omega} \xi_c \mathrm{d}\Omega = \frac{V}{4} \quad c = 1, 2, 3, 4 \tag{B.38}
$$

Therefore, the CPDI-Tet4 weighting function ϕ_{Ip} is given by

$$
\phi_{Ip} = \frac{1}{4}\Big[N_I(\mathbf{x}_1) + N_I(\mathbf{x}_2) + N_I(\mathbf{x}_3) + N_I(\mathbf{x}_4)\Big] \tag{B.39}
$$

To derive the expression for the first derivatives, we use the following identities

$$\frac{\partial F}{\partial x} = \frac{a_i}{6V}\frac{\partial F}{\partial \xi_i}, \quad \frac{\partial F}{\partial y} = \frac{b_i}{6V}\frac{\partial F}{\partial \xi_i}, \quad \frac{\partial F}{\partial z} = \frac{c_i}{6V}\frac{\partial F}{\partial \xi_i} \tag{B.40}$$

where a_i, b_i and c_i are defined in Eq. (3.44). Here the Einstein summation convention over $i = 1, 2, 3, 4$ applies to the repeated indexes. For example, Eq. (B.40) applies to ξ_1 i.e., the first shape function gives

$$\frac{\partial \xi_1}{\partial x} = \frac{a_1}{6V}, \quad \frac{\partial \xi_1}{\partial y} = \frac{b_1}{6V}, \frac{\partial \xi_1}{\partial z} = \frac{c_1}{6V}, \tag{B.41}$$

Therefore, we have

$$\int_\Omega \frac{\partial \xi_1}{\partial x} \mathrm{d}\Omega = \frac{a_1}{6}, \quad \int_\Omega \frac{\partial \xi_1}{\partial y} \mathrm{d}\Omega = \frac{b_1}{6}, \quad \int_\Omega \frac{\partial \xi_1}{\partial z} \mathrm{d}\Omega = \frac{c_1}{6} \tag{B.42}$$

By repeating this for other shape functions, we obtain the final expression for the CPDI-Tet4 gradient weighting functions as

$$\begin{aligned}
\phi_{Ip,x} &= \frac{a_1}{6V} N_I(\mathbf{x}_1) + \frac{a_2}{6V} N_I(\mathbf{x}_2) + \frac{a_3}{6V} N_I(\mathbf{x}_3) + \frac{a_4}{6V} N_I(\mathbf{x}_4) \\
\phi_{Ip,y} &= \frac{b_1}{6V} N_I(\mathbf{x}_1) + \frac{b_2}{6V} N_I(\mathbf{x}_2) + \frac{b_3}{6V} N_I(\mathbf{x}_3) + \frac{b_4}{6V} N_I(\mathbf{x}_4) \\
\phi_{Ip,z} &= \frac{c_1}{6V} N_I(\mathbf{x}_1) + \frac{c_2}{6V} N_I(\mathbf{x}_2) + \frac{c_3}{6V} N_I(\mathbf{x}_3) + \frac{c_4}{6V} N_I(\mathbf{x}_4)
\end{aligned} \tag{B.43}$$

Appendix C
Utilities

This chapter is devoted to utilities that have been proved to be useful for computational mechanicians. First, we present codes for the visualization of one/two dimensional functions (such as the shape functions employed in FEM and the MPM) in Sect. C.1. Then, we illustrate the use of a computer algebra system open source package to carry out length derivations (such as derivation of CPDI functions) in Sect. C.2. Next, we present a derivation of the modified cubic and quadratic B-splines basis functions (Sect. C.3). Our derivation used a Python symbolic module. Section C.4 briefly explains how remote machines are used to compile and run codes. Finally Sect. C.5 discusses dimensions and units.

C.1 Scripts to Plot Basis Functions

This section presents `Matlab` scripts to plot various MPM basis functions. Listing C.1 is used to plot one dimensional GIMP functions on a grid of four cells and five nodes. The resulting plot was given in Fig. 3.7. Listing C.2 presents commands to create plots of two dimensional GIMP functions given in Fig. 3.8. Since the codes are quite self explanatory, we do not provide any explanations.

N. V. Phu et al., *The Material Point Method*, Scientific Computation,
https://doi.org/10.1007/978-3-031-24070-6

Listing C.1 `Matlab` script to plot 1D GIMP basis functions

```
L           = 4;            % physical domain
elemCount = 4;              % no of elements
nodes       = linspace(0,L,elemCount+1); % nodes
dx          = L/elemCount;% grid spacing
n           = 2;
ptsCount    = 250;
x           = 0:L/ptsCount:L;% sampling points where phi_I evaluated;
shapeFunc   = zeros(elemCount+1,length(x));
dshapeFunc = zeros(elemCount+1,length(x));
% compute phi_I and dphi_I, I=1:elemCount+1 at x(j): j=1:ptsCount
for I=1:elemCount+1
    for i=1:length(x)
        pts = x(i) - nodes(I);
        [shapeFunc(I,i), dshapeFunc(I,i)] = getGIMP(pts,dx,1.5);
    end
end
%%
figure(1)
hold on
plot(x,shapeFunc(1,:),'black--','LineWidth',1.4);
plot(x,shapeFunc(2,:),'red--'  ,'LineWidth',1.4);
%plot(x,sum(dshapeFunc,1),'black---' ,'LineWidth',1.4);
axis equal, axis([0 4 0 1])
```

Listing C.2 `Matlab` script to plot 2D GIMP basis functions

```
[X,Y] = meshgrid (linspace(0,4,200));  % X,Y: grid of 200x200 points on a square 4x4
R     = zeros(size(X,1),size(X,1));
xI    = [2 2]; % node I with phi_I
lp    = 1;     % particle size
h     = 1;     % grid space
% compute phi_I at 200x200 points
for i=1:size(X,1)
    for j=1:size(X,1)
        x           = X(1,i);
        y           = Y(j,1);
        pts         = x - xI(1);
        [Nx, dshape] = getGIMP(pts,h,lp);
        pts         = y - xI(2);
        [Ny, dshape] = getGIMP(pts,h,lp);
        R(i,j)      = Nx*Ny;
    end
end
figure % 3D plot
surf (X,Y,R, 'EdgeColor','none','LineStyle','none','FaceLighting','phong')
hold off
figure % contour plot
surf (X,Y,R, 'EdgeColor','none','LineStyle','none')
hold off
axis equal, view([0 90]), colorbar
```

C.2 Symbolic Calculus

A computer algebra system (CAS) is a software program that allows computation over mathematical expressions in a way that is similar to the traditional hand computations carried out by mathematicians, scientists and engineers. CAS have been shown extremely useful, among other things, for deriving complex expressions such as the material tangent matrices of complex constitutive models in the FEM community. There exists excellent commercial CAS such as Maple, Mathematica and Matlab. However, herein we present the derivation of CPDI basis functions using SageMath, an open source CAS. The first version of SageMath was released in 2005 with the initial goals of creating an 'open source alternative to Magma, Maple, Mathematica, and Matlab'. Furthermore, SageMath supports technical writing using LaTeX.

Listing C.3 SageMath script to derive CPDI-L3 basis functions

```
1  var('xi x1 x2 x3')              # define variables
2  N1=-0.5*(1-xi)*xi
3  N2= 0.5*(1+xi)*xi
4  N3= 1-xi^2
5  x = N1*x1 + N2*x2 + N3*x3
6  J = x.diff(xi)                  # Jacobian
7  N1J = N1*J
8  N2J = N2*J
9  N3J = N3*J
10 wf1 = N1J.integral(xi,-1,1)     # 1st func. weight without 1/Vp
11 wf2 = N2J.integral(xi,-1,1)
12 wf3 = N3J.integral(xi,-1,1)
13 view(wf1)
14 latex(wf1)
```

Listing C.3 presents the SageMath scripts used to derive the CPDI-L3 basis functions. Note that the final command `latex(wf1)` is to export the SageMath object `wf1` to LaTeX which was copied to our LaTeX document to generate the following equation

$$-\frac{1}{2}x_1 - \frac{1}{6}x_2 + \frac{2}{3}x_3 \tag{C.1}$$

Listing C.4 presents the SageMath scripts used to derive the CPDI-Q4 basis functions. Line 13 is used to carry out double integrations.

Listing C.4 SageMath script to derive CPDI-Q4 basis functions

```
var('xi eta x1 y1 x2 y2 x3 y3 x4 y4')
N1=0.25*(1-xi)*(1-eta)
N2=0.25*(1+xi)*(1-eta)
N3=0.25*(1+xi)*(1+eta)
N4=0.25*(1-xi)*(1+eta)
x = N1*x1 + N2*x2 + N3*x3 + N4 * x4
y = N1*y1 + N2*y2 + N3*y3 + N4 * y4
J = x.diff(xi)*y.diff(eta) - x.diff(eta)*y.diff(xi)
N1J = N1*J
N2J = N2*J
N3J = N3*J
N4J = N4*J
wf1 = integral(integral(N1J,eta,-1,1),xi,-1,1)
wf2 = integral(integral(N2J,eta,-1,1),xi,-1,1)
wf3 = integral(integral(N3J,eta,-1,1),xi,-1,1)
wf4 = integral(integral(N4J,eta,-1,1),xi,-1,1)
show(wf1.simplify_full())
show(wf1+wf2+wf3+wf4).simplify_full()
```

C.3 Derivation of B-Spline Basis Functions

We present herein the derivation of the boundary modified cubic B-splines given in Sect. 3.4, see Sect. C.3.1 and quadratic B-splines (Sect. C.3.2). In the process, we introduce the package `SymPy` which is suitable for this kind of task. `SymPy` is a `Python` module for symbolic mathematics. It's similar to commercial CAS like Maple or Mathematica. Note that we also used Mathematica to do this; but we prefer open source tools. We did not use SageMath to illustrate the point that it is beneficial to know many programming languages, and use the best one for a given task. Recall that we have used `C++`, `Matlab`, `Python`, `Julia` and `SageMath` in this book.

C.3.1 Cubic B-Splines

We start with the knot vector $\Xi = \{0, 0, 0, 0, 1, 2, 3, 4, 5, 5, 5, 5\}$. Using Eqs. (3.16) and (3.17) implemented in `SymPy`, one can get all the functions. They are given by (we used x instead of ξ as all the following equations are copied from `SymPy` and in the following equations, $\wedge$ means or)

$$N_{0,3}(x) = \begin{cases} -x^3 + 3x^2 - 3x + 1 & \text{for } x \geq 0 \wedge x \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{C.2}$$

$$N_{1,3}(x) = \begin{cases} \frac{7x^3}{4} - \frac{9x^2}{2} + 3x & \text{for } x \geq 0 \wedge x \leq 1 \\ -\frac{x^3}{4} + \frac{3x^2}{2} - 3x + 2 & \text{for } x \geq 1 \wedge x \leq 2 \\ 0 & \text{otherwise} \end{cases} \tag{C.3}$$

$$N_{2,3}(x) = \begin{cases} -\frac{11x^3}{12} + \frac{3x^2}{2} & \text{for } x \geq 0 \wedge x \leq 1 \\ \frac{7x^3}{12} - 3x^2 + \frac{9x}{2} - \frac{3}{2} & \text{for } x \geq 1 \wedge x \leq 2 \\ -\frac{x^3}{6} + \frac{3x^2}{2} - \frac{9x}{2} + \frac{9}{2} & \text{for } x \geq 2 \wedge x \leq 3 \\ 0 & \text{otherwise} \end{cases} \tag{C.4}$$

$$N_{3,3}(x) = \begin{cases} \frac{x^3}{6} & \text{for } x \geq 0 \wedge x \leq 1 \\ -\frac{x^3}{2} + 2x^2 - 2x + \frac{2}{3} & \text{for } x \geq 1 \wedge x \leq 2 \\ \frac{x^3}{2} - 4x^2 + 10x - \frac{22}{3} & \text{for } x \geq 2 \wedge x \leq 3 \\ -\frac{x^3}{6} + 2x^2 - 8x + \frac{32}{3} & \text{for } x \geq 3 \wedge x \leq 4 \\ 0 & \text{otherwise} \end{cases} \tag{C.5}$$

$$N_{4,3}(x) = \begin{cases} \frac{x^3}{6} - \frac{x^2}{2} + \frac{x}{2} - \frac{1}{6} & \text{for } x \geq 1 \wedge x \leq 2 \\ -\frac{x^3}{2} + \frac{7x^2}{2} - \frac{15x}{2} + \frac{31}{6} & \text{for } x \geq 2 \wedge x \leq 3 \\ \frac{x^3}{2} - \frac{11x^2}{2} + \frac{39x}{2} - \frac{131}{6} & \text{for } x \geq 3 \wedge x \leq 4 \\ -\frac{x^3}{6} + \frac{5x^2}{2} - \frac{25x}{2} + \frac{125}{6} & \text{for } x \geq 4 \wedge x \leq 5 \\ 0 & \text{otherwise} \end{cases} \tag{C.6}$$

$$N_{5,3}(x) = \begin{cases} \frac{x^3}{6} - x^2 + 2x - \frac{4}{3} & \text{for } x \geq 2 \wedge x \leq 3 \\ -\frac{7x^3}{12} + \frac{23x^2}{4} - \frac{73x}{4} + \frac{227}{12} & \text{for } x \geq 3 \wedge x \leq 4 \\ \frac{11x^3}{12} - \frac{49x^2}{4} + \frac{215x}{4} - \frac{925}{12} & \text{for } x \geq 4 \wedge x \leq 5 \\ 0 & \text{otherwise} \end{cases} \tag{C.7}$$

and so on (we skipped the remaining functions for brevity).

Then, we replace the first basis function i.e., $N_{0,3}(x)$ by $N_{0,3}(x) = N_{0,3}(x) + {}^2/_3 N_{1,3}(x)$, and we get this

$$N_{1,3}(x) := \begin{cases} \frac{1}{6}x^3 - x + 1 & \text{for } x \geq 0 \wedge x \leq 1 \\ -\frac{1}{6}x^3 + x^2 - 2x + 4/3 & \text{for } x \geq 1 \wedge x \leq 2 \\ 0 & \text{otherwise} \end{cases} \tag{C.8}$$

And we replace the third function $N_{3,3}(x)$ by $N_{3,3}(x) = N_{3,3}(x) + {}^1/_3 N_{1,3}(x)$, and obtain

$$N_{2,3}(x) := \begin{cases} x\left(1 - \frac{x^2}{3}\right) & \text{for } x \geq 0 \wedge x \leq 1 \\ 0.5x^3 - 2.5x^2 + 3.5x - \frac{5}{6} & \text{for } x \geq 1 \wedge x \leq 2 \\ -\frac{x^3}{6} + \frac{3x^2}{2} - \frac{9x}{2} + \frac{9}{2} & \text{for } x \geq 2 \wedge x \leq 3 \\ 0 & \text{otherwise} \end{cases} \tag{C.9}$$

By distributing $N_{1,3}(x)$ to the first and second basis functions, we maintain the partition of unity of the basis functions. The coefficients $2/3$ and $1/3$ were obtained to make sure that that the peak of $N_{2,3}(x)$ equals that of $N_{3,3}(x)$.

We do not change $N_{3,3}(x)$ as it is node centred, so we have

$$N_{3,3}(x) = \begin{cases} \frac{x^3}{6} & \text{for } x \geq 0 \wedge x \leq 1 \\ -\frac{x^3}{2} + 2x^2 - 2x + \frac{2}{3} & \text{for } x \geq 1 \wedge x \leq 2 \\ \frac{x^3}{2} - 4x^2 + 10x - \frac{22}{3} & \text{for } x \geq 2 \wedge x \leq 3 \\ -\frac{x^3}{6} + 2x^2 - 8x + \frac{32}{3} & \text{for } x \geq 3 \wedge x \leq 4 \\ 0 & \text{otherwise} \end{cases} \tag{C.10}$$

And one proceeds in the same manner for the remaining functions.

In the MPM, we need to evaluate the grid basis at a particle i.e., $N_I(x_p)$ using the coordinate $r := (x_p - x_I)/h$. For the second basis in Eq. (C.9), we have $-1 \leq r \leq 2$, so we use the transformation $x = r + 1$ in Eq. (C.9) to obtain the final MPM cubic spline function

$$N_{2,3}(r) := \begin{cases} -\frac{1}{3}r^3 - r^2 + \frac{2}{3} & \text{for } -1 \leq r \leq 0 \\ \frac{1}{2}r^3 - r^2 + \frac{2}{3} & \text{for } 0 \leq r \leq 1 \\ -\frac{1}{6}r^3 + r^2 - 2r + \frac{4}{3} & \text{for } 1 \leq r \leq 2 \end{cases} \tag{C.11}$$

The Python script for all this derivation is given in Listing C.5; and the plot of the modified cubic splines is shown in Fig. C.1. The sum of all basis functions is added

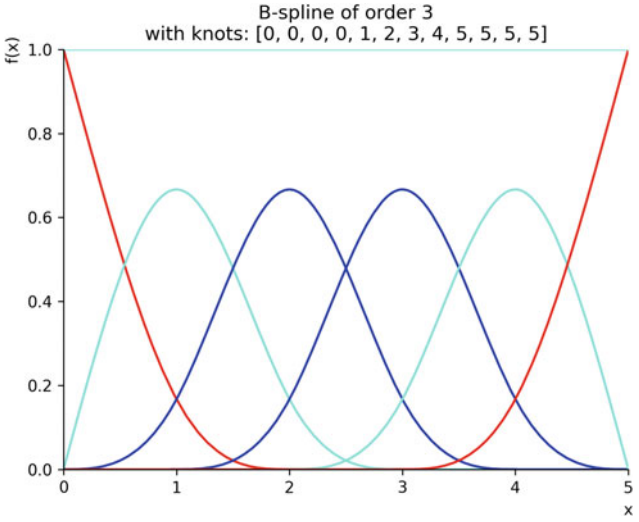


Fig. C.1 Modified cubic splines obtained and visualized using `SymPy`

to this plot to prove the partition of unity of the modified functions. Note that the plotting functionality of SymPy is quite limited.

Listing C.5 Python script to derive modified cubic basis functions

```
from sympy import *
from sympy.abc import x
from matplotlib import style
import matplotlib.pyplot as plt
d           = 3
knots       = [0, 0, 0, 0, 1, 2, 3, 4, 5, 5, 5, 5]
bsplines    = bspline_basis_set(d, knots, x)
line_colors = ['red','blue','blue','cyan','green','blue','blue','red']
# modified bsplines
bsplines1 = bsplines[0]+(2./3.)*bsplines[1]
bsplines2 = bsplines[2]+(1./3.)*bsplines[1]
bsplines3 = bsplines[3]
bsplines4 = bsplines[4]
bsplines5 = bsplines[5]+(1./3)*bsplines[6]
bsplines6 = bsplines[7]+(2./3)*bsplines[6]
# to check PUM
tspline   = bsplines1+bsplines2+bsplines3+bsplines4+bsplines5+bsplines6
# plotting the modified bsplines
p = None
p2=plot(bsplines1,(x, 0, knots[-1]),line_color=line_colors[0])
p = p2
p2=plot(bsplines2, (x, 0, knots[-1]),line_color=line_colors[3])
p.extend(p2)
...
p.show()
print(latex(simplify(bsplines2))) # to get latex for the equation
```

C.3.2 Quadratic B-Splines

Our goal is to obtain 5 boundary modified B-splines centered at the nodes (Fig. C.2). We start from the normal quadratic B-splines as shown in Fig. C.3. The idea is to use these normal functions for $0.5 \leq x \leq 3.5$, outside of this domain, we use linear functions.

The internal basis function is already centered at nodes, so we just use it. Thus, we get type 3 basis (cyan basis in Fig. C.2)

$$N_{3,2}(r) = \begin{cases} \frac{1}{2}r^2 + \frac{3}{2}r + \frac{9}{8} & \text{for } -3/2 \leq r \leq -1/2 \\ -r^2 + \frac{3}{4} & \text{for } -1/2 \leq r \leq 1/2 \\ \frac{1}{2}r^2 - \frac{3}{2}r + \frac{9}{8} & \text{for } 1/2 \leq r \leq 3/2 \\ 0 & \text{otherwise} \end{cases} \tag{C.12}$$

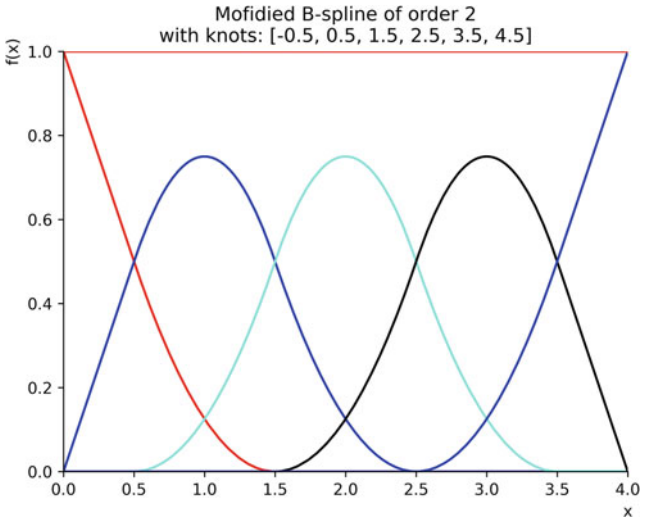


Fig. C.2 Modified quadratic splines obtained and visualized using `SymPy`. At any point there are always three non-zero basis functions. For internal cells, one need to check if point p is on the left half or right half of the cell to know which basis functions are non-zero. In 2D, there are 3×3 non-zero functions

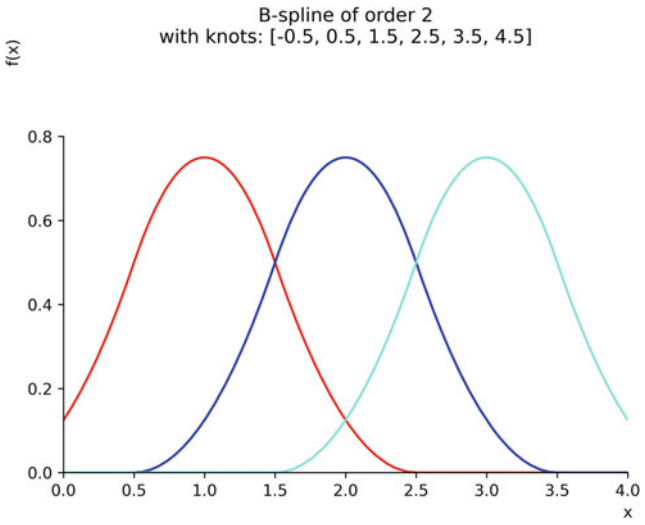


Fig. C.3 Modified quadratic splines obtained and visualized using `SymPy`

This is the basis for internal nodes i.e., nodes do not belong to boundary cells.

To get the type 1 basis (red basis in Fig. C.2), we reuse the first quadratic basis (red in Fig. C.3) for $0.5 \leq x \leq 1.5$. And add a linear function $y = x$ for $0 \leq x \leq 0.5$ to it. The resulting function is

$$N_{1,2}(x) = \begin{cases} x & \text{for } x \geq 0 \wedge x \leq 0.5 \\ -1.0x^2 + 2.0x - 0.25 & \text{for } x \geq 0.5 \wedge x \leq 1.5 \\ 0.5x^2 - 2.5x + 3.125 & \text{for } x \geq 1.5 \wedge x \leq 2.5 \\ 0 & \text{otherwise} \end{cases} \tag{C.13}$$

Using $x = r + 1$, we obtain the final expression for type 1 basis function (this variable change can be done using `SymPy` directly)

$$N_{1,2}(r) = \begin{cases} r + 1 & \text{for } r \geq -1 \wedge r \leq -0.5 \\ 0.75 - 1.0r^2 & \text{for } r \geq -0.5 \wedge r \leq 0.5 \\ 0.5r^2 - 1.5r + 1.125 & \text{for } r \geq 0.5 \wedge r \leq 1.5 \\ 0 & \text{otherwise} \end{cases} \tag{C.14}$$

Similarly, to get the type 4 basis (black basis in Fig. C.2), we reuse the third basis (cyan in Fig. C.3), and add the linear segment $4 - x$ for $3.5 \leq x \leq 4.5$ to it. The result is the following function

$$N_{4,2}(x) = \begin{cases} 0.5x^2 - 1.5x + 1.125 & \text{for } x \geq 1.5 \wedge x \leq 2.5 \\ -1.0x^2 + 6.0x - 8.25 & \text{for } x \geq 2.5 \wedge x \leq 3.5 \\ 4.0 - x & \text{for } x \geq 3.5 \wedge x \leq 4 \\ 0 & \text{otherwise} \end{cases} \tag{C.15}$$

And using $x = r + 3$, we get

$$N_{4,2}(r) = \begin{cases} 0.5r^2 + 1.5r + 1.125 & \text{for } r \geq -1.5 \wedge r \leq -0.5 \\ 0.75 - 1.0r^2 & \text{for } r \geq -0.5 \wedge r \leq 0.5 \\ 1.0 - r & \text{for } r \geq 0.5 \wedge r \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{C.16}$$

Actually this function can also be obtained from Eq. (C.14) using vertical reflection.

Finally, to get the type 1 basis (red basis in Fig. C.2) for $0.5 \leq x \leq 1.5$, we use the third normal bspline with $x = 3 - y$ and add $1 - x$ for $0 \leq x \leq 0.5$. The final expression for type basis function is

$$N_{1,2}(r) = \begin{cases} 0.5r^2 + 1.5r + 1.125 & \text{for } r \geq -1.5 \wedge r \leq -0.5 \\ r + 1 & \text{for } r \geq -0.5 \wedge r \leq 0 \\ 1 - r & \text{for } r \geq 0 \wedge r \leq 0.5 \\ 0.5r^2 - 1.5r + 1.125 & \text{for } r \geq 0.5 \wedge r \leq 1.5 \\ 0 & \text{otherwise} \end{cases} \tag{C.17}$$

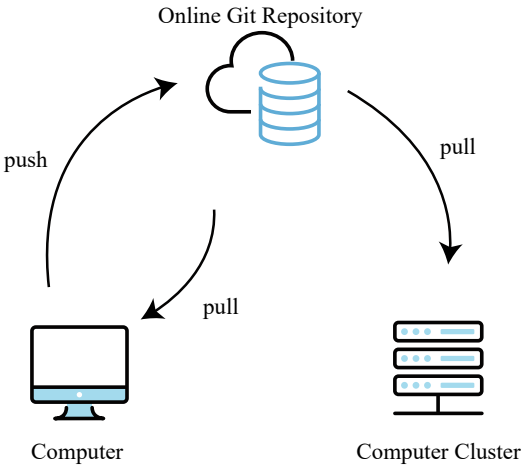


Fig. C.4 A workflow of nowadays computational scientist: a PC or a laptop is used to edit the source code, an online git repository to store the code and (3) a remote machine such as a cluster for running the simulations

C.4 Running Simulations Using a Remote Machine

As engineering problems are getting more complex, using an office computer is not sufficient. We often find ourselves using remote clusters. This appendix briefly documents this working process. We need three things: (1) a PC or a laptop that is used to edit the source code, (2) an online git repository to store our code and (3) a remote machine which is used for running the simulations. Figure C.4 illustrates this workflow in which a pull request is made to get the latest code from the online repository and a push is done to upload the changed code. The compilation of the code (if written in `C++` or `Fortran`) is done in the cluster using SSH (Secure Shell).

When it comes to remote machines, `tmux` (Fig. C.5) is particularly useful. Let's imagine that you need to run a very long script on your remote server. You could:

- Connect to your remote server via SSH and launch `tmux` on the remote server.
- Run a script which takes hours.
- Close the SSH connection. The script will still run on the remote server, thanks to `tmux`. You can switch off your own computer.

C.5 Units

We want to discuss units in stress analyses. On one hand, we can choose what we like, as long as it is consistent. This freedom comes from the fact that many commercial simulation packages such as Abaqus have no built-in system of units. This is also the case for the many codes presented in this book. And by consistent, we mean that derived units of the chosen system can be expressed in terms of the fundamental units without conversion factors. Fundamental units are Length (L), Mass (M), Force (F)

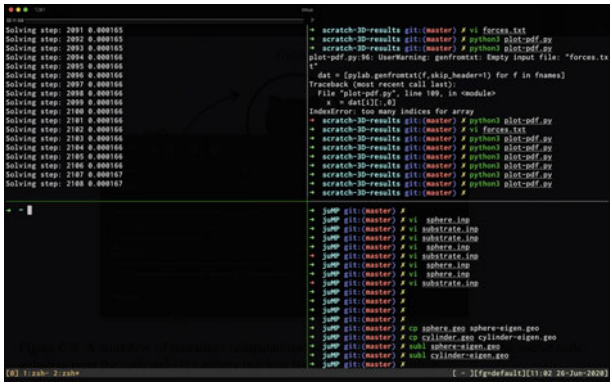


Fig. C.5 On a PC, `tmux` is kind of a powerful window manager. Shown here is one `tmux` session with four panes on one window. You can have multiple sessions and each session can have its own windows. But it is extremely useful when it comes to SSH to remote machines

Table C.1 Common consistent units for stress analyses. One example of unit conversion is the density: $7850\,\text{kg/m}^3 = 7.85 \times 10^{-9}$ tonne/mm^3 $= 7850 \times 10^{-9}$ kg/mm^3

Variable	Meaning	SI (m, s, N)	SI (mm, s, N)	SI (mm, ms, kN)
l	Length	m	mm	mm
t	Time	s	s	ms
F	Force	N	N	kN
m	Mass	kg	tonne [10^3 kg]	kg
T	Temperature	K or C	K	K
ρ	Density	kg/m^3	tonne/mm^3 [10^{-6} kg/m^3]	kg/mm^3 [10^{-9} kg/m^3]
E	Young's modulus	Pa	MPa [10^6 Pa]	GPa [10^9 Pa]
σ	Stress	Pa [N/m^2]	MPa [10^6 Pa]	GPa [10^9 Pa]
e	Energy	J [Nm]	mJ [10^{-3} J]	J
G_c	Fracture energy	J/m^2	N/mm [10^3 J/m^2]	[10^6 J/m^2]
v	Velocity	m/s	mm/s [10^{-3} m/s]	mm/ms [m/s]

and Time (T). Other units are derived from them e.g. volume = length powered to the three. On the other hand, this is a common source of mistakes. Therefore we want to go into a bit more detail.

Some common consistent SI units are given in Table C.1.

The relation between F, M, L and T is given by Newton's second law $F = ma$

$$F = ma \Rightarrow \text{N} = \text{kg} \times \frac{\text{m}}{\text{s}^2} \tag{C.18}$$

Therefore, if you use mm for length and still N for force and s for time, then the mass must be in tonne:

$$\mathrm{N} = 10^3\mathrm{kg} \times \frac{\mathrm{mm}}{\mathrm{s}^2} \tag{C.19}$$

Appendix D
Explicit Lagrangian Finite Elements

It is a common practice to verify a new method with the finite element method, which is reliable. Therefore, it is useful to have a FEM code in hand. Due to this reason, in this appendix we present the implementation of explicit dynamic Lagrangian finite elements for large deformation problems. Readers are referred to textbooks e.g. Belytschko et al. (2000), Wu and Wu (2012) for derivations and a comprehensive treatment of the topic. Herein, we focus on computer implementation aspects and verification tests using the method of manufactured solutions. Topics such as contact and impact are not covered as they are more difficult to deal with by FEM than by the MPM. Furthermore, we confine our discussion to nonlinear elastic materials in the framework of hyperelasticity. This is because this type of materials is sufficient to demonstrate finite element formulations for large deformation problems without delving too much into the stress updates of more complex materials such as hyperelastic-plastic ones.

Finite elements using Lagrangian meshes are commonly classified as updated Lagrangian formulation (Sect. D.1) and total Lagrangian formulation (Sect. D.2). In both formulations, the independent variables are the material coordinates $\mathbf{X}$ and time t. In the total Lagrangian formulation, the stress and strain are Lagrangian, i.e., they are defined with respect to the reference configuration (for example, the nominal or second Piola-Kirchhoff stress are employed), the derivatives are computed with respect to the material coordinates. The corresponding weak form therefore involves integrals over the reference configuration. On the other hand, the updated Lagrangian formulation uses the Eulerian strain and stress measures e.g., the Cauchy stress, the derivatives are computed with respect to the spatial coordinates $\mathbf{x}$. The corresponding weak form therefore involves integrals over the current (deformed) configuration.

D.1 Updated Lagrangian Finite Elements

The computational domain is discretized by continuum elements and the solution is advanced in time using the central difference scheme (or the leapfrog scheme).

N. V. Phu et al., *The Material Point Method*, Scientific Computation,
https://doi.org/10.1007/978-3-031-24070-6

We pay attention to the overall flowchart (Sect. D.1.1) and the computation of the internal force vector (Sect. D.1.2).

D.1.1 General Flowchart

The flowchart of an updated Lagrangian FE code is given in Algorithm 24. As the mass matrix is constant, it is computed once.

Algorithm 24 Updated Lagrangian finite elements.

Initialization
  Initialize nodal velocities/displacements and stresses at Gauss points;
  Compute the consistent mass matrix, diagonalize it;
end
while $t < T$ **do**
  Compute nodal forces (refer to Algorithm 25)
    Compute external/internal forces $\mathbf{f}_I^{\text{ext},t}$, $\mathbf{f}_I^{\text{int},t}$
    Compute nodal force $\mathbf{f}_I^t = \mathbf{f}_I^{\text{ext},t} + \mathbf{f}_I^{\text{int},t}$
  end
  Update the velocity $\mathbf{v}_I^{t+\Delta t} = \mathbf{v}_I^t + (\mathbf{f}_I^t/m_I)\Delta t$
  Fix Dirichlet nodes I e.g. $\mathbf{v}_I^{t+\Delta t} = \bar{\mathbf{v}}$
  Update displacements $\mathbf{u}_I^{t+\Delta t} = \mathbf{u}_I^t + \Delta t \mathbf{v}_I^{t+\Delta t}$
  Update mesh $\mathbf{x}_I^{t+\Delta t} = \mathbf{x}_I^t + \Delta t \mathbf{v}_I^{t+\Delta t}$
end while

D.1.2 Computation of Internal Force

For simplicity we confine our discussion to two dimensions. The internal force vector at a generic node I is given by

$$\begin{bmatrix} f_{xI}^{\text{int}} \\ f_{yI}^{\text{int}} \end{bmatrix} = \int_\Omega \begin{bmatrix} N_{I,x} & 0 & N_{I,y} \\ 0 & N_{I,y} & N_{I,x} \end{bmatrix} \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{bmatrix} \mathrm{d}\Omega \tag{D.1}$$

where derivatives are with respect to the current configuration. For an element with n nodes, we can collectively gather the internal force vectors of all n nodes to get the so-called element internal force vector[1]

$$\mathbf{f}_e^{\text{int}} = \int_{\Omega_e} \mathbf{B}^{\mathrm{T}} \boldsymbol{\sigma} \mathrm{d}\Omega \tag{D.2}$$

[1] For finite elements it is more common to work on an element basis rather than on a node basis. Therefore we do not compute the internal force node by node but rather we compute it element by element. This is not a requirement but we decided to do so to be consistent with the FEM practices.

where Ω_e denotes the element domain and the well known $\mathbf{B}$ matrix in finite element codes for small deformation inelastic materials appear. Explicitly written, $\mathbf{B}$ is given by

$$\mathbf{B} = \begin{bmatrix} N_{1,x} & 0 & N_{2,x} & 0 & \dots & N_{n,x} & 0 \\ 0 & N_{1,x} & 0 & N_{2,x} & \dots & 0 & N_{n,y} \\ N_{1,y} & N_{I,x} & N_{2,y} & N_{2,x} & \dots & N_{n,y} & N_{n,x} \end{bmatrix} \tag{D.3}$$

which is a $3 \times 2n$ matrix. The procedure is identical to FEM for small deformation inelastic materials case, except that the mesh is updated (Algorithm 25).

Algorithm 25 Updated Lagrangian: computation of internal forces at time t.

for each element e **do**
  get element connectivity of e, $esctr$
  get element coordinates of e, $\mathbf{x}_e$
  initialize element internal force $\mathbf{f}_e^{\text{int}} = \mathbf{0}$
  for each Gauss point g **do**
    compute the shape functions/derivatives $\mathbf{N}(\boldsymbol{\xi}_g)$ and $\mathbf{dNdxi}(\boldsymbol{\xi}_g)$
    compute the Jacobian $\mathbf{J} = (\mathbf{x}_e)^{\mathrm{T}}\mathbf{dNdxi}$
    compute the shape function derivatives $\mathbf{dNdx} = \mathbf{dNdxiJ}^{-1}$
    compute $\mathbf{B}$ matrix
    compute stresses
    compute internal force $\mathbf{f}_e^{\text{int}} = \mathbf{f}_e^{\text{int}} + \mathbf{B}^{\mathrm{T}}\boldsymbol{\sigma}_g w_g \det \mathbf{J}$
  end for
  Assemble $\mathbf{f}_e^{\text{int}}$ to the global internal force $\mathbf{f}^{\text{int}}$ using $esctr$
end for

If the Cauchy stresses are stored in a full matrix rather than in a vector (using the Voigt notation), then the element internal force is given by

$$\begin{bmatrix} f_{x1}^{\text{int}} & f_{y1}^{\text{int}} \\ f_{x2}^{\text{int}} & f_{y2}^{\text{int}} \\ \vdots & \vdots \\ f_{xn}^{\text{int}} & f_{yn}^{\text{int}} \end{bmatrix} = \int_{\Omega^e} \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_1}{\partial y} \\ \frac{\partial N_2}{\partial x} & \frac{\partial N_2}{\partial y} \\ \vdots & \vdots \\ \frac{\partial N_n}{\partial x} & \frac{\partial N_n}{\partial y} \end{bmatrix} \begin{bmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy} \end{bmatrix} \mathrm{d}\Omega = \int_{\Omega^e} \boldsymbol{\mathcal{B}}^{\mathrm{T}}[\boldsymbol{\sigma}]\mathrm{d}\Omega \tag{D.4}$$

which is implemented in our Matlab code. We find it convenient in Matlab to work this way when we assemble the element force into the global force.

One can compute the gradient deformation from the velocity gradient $\mathbf{L}$ as done in the MPM. Alternatively one can compute $\mathbf{F}$ directly as follows

$$\mathbf{F} = \frac{\partial \mathbf{x}}{\partial \mathbf{X}} = \left(\frac{\partial \mathbf{X}}{\partial \mathbf{x}}\right)^{-1} = \left(\mathbf{I} - \frac{\partial \mathbf{u}}{\partial \mathbf{x}}\right)^{-1} = \left(\mathbf{I} - \mathbf{u}^e \boldsymbol{\mathcal{B}}^{\mathrm{T}}\right)^{-1} \tag{D.5}$$

where $\mathbf{u}^e$ denotes the element displacement matrix of which explicit expression will be given shortly in Eq. (D.11).

D.2 Total Lagrangian Finite Elements

The general flowchart of a total Lagrangian FE code is similar to the UL finite element code except that one does not update the mesh (i.e., the node coordinates are not updated) as we are always working on the initial reference configuration. For ease of implementation the flowchart is given in Algorithm 26.

Algorithm 26 Total Lagrangian finite elements.

Initialization
  Initialize nodal velocities/displacements and stresses at Gauss points;
  Compute the consistent mass matrix, diagonalize it;
end
while $t < T$ **do**
  Compute nodal forces (refer to Algorithm 27)
    Compute external/internal forces $\mathbf{f}_I^{\text{ext},t}$, $\mathbf{f}_I^{\text{int},t}$
    Compute nodal force $\mathbf{f}_I^t = \mathbf{f}_I^{\text{ext},t} + \mathbf{f}_I^{\text{int},t}$
  end
  Update the velocity $\mathbf{v}_I^{t+\Delta t} = \mathbf{v}_I^t + (\mathbf{f}_I^t/m_I)\Delta t$
  Fix Dirichlet nodes I e.g. $\mathbf{v}_I^{t+\Delta t} = \bar{\mathbf{v}}$
  Update displacements $\mathbf{u}_I^{t+\Delta t} = \mathbf{u}_I^t + \Delta t \mathbf{v}_I^{t+\Delta t}$
end while

The TL form of the internal force vector is given by (Belytschko et al. 2000)

$$f_{iI}^{\text{int}} = \int_{\Omega_0^e} \frac{\partial N_I}{\partial X_k} P_{ki} \mathrm{d}\Omega \tag{D.6}$$

which can be rewritten explicitly as in 2D

$$\begin{bmatrix} f_{xI}^{\text{int}} & f_{yI}^{\text{int}} \end{bmatrix} = \int_{\Omega_0} \begin{bmatrix} \frac{\partial N_I}{\partial X} & \frac{\partial N_I}{\partial Y} \end{bmatrix} \begin{bmatrix} P_{xx} & P_{xy} \\ P_{yx} & P_{yy} \end{bmatrix} \mathrm{d}\Omega \tag{D.7}$$

And for an element with n nodes, Eq. (D.7) yields the following expression for the elemental internal force matrix

$$\begin{bmatrix} f_{x1}^{\text{int}} & f_{y1}^{\text{int}} \\ f_{x2}^{\text{int}} & f_{y2}^{\text{int}} \\ \vdots & \vdots \\ f_{xn}^{\text{int}} & f_{yn}^{\text{int}} \end{bmatrix} = \int_{\Omega_0^e} \begin{bmatrix} \frac{\partial N_1}{\partial X} & \frac{\partial N_1}{\partial Y} \\ \frac{\partial N_2}{\partial X} & \frac{\partial N_2}{\partial Y} \\ \vdots & \vdots \\ \frac{\partial N_n}{\partial X} & \frac{\partial N_n}{\partial Y} \end{bmatrix} \begin{bmatrix} P_{xx} & P_{xy} \\ P_{yx} & P_{yy} \end{bmatrix} \mathrm{d}\Omega \tag{D.8}$$

Or in a more compact form as

$$\left[\mathbf{f}_e^{\text{int}}\right] = \int_{\Omega_0^e} \mathcal{B}_0^{\mathrm{T}} \left[\mathbf{P}\right] \mathrm{d}\Omega, \quad \mathcal{B}_0 = \left[\mathcal{B}_1 \ \mathcal{B}_2 \ \ldots \ \mathcal{B}_n\right], \quad \mathcal{B}_I = \begin{bmatrix} N_{I,X} \\ N_{I,Y} \end{bmatrix} \tag{D.9}$$

which is very similar to the corresponding UL form of the internal force. However it should be noted that when using the nominal stress the stress is now stored as a matrix (because it is a non-symmetric tensor) not as a vector, and thus the internal force is now a matrix. We use square brackets to differentiate an internal force matrix from an internal force vector.

Next we present how to compute the deformation gradient (which is needed to determine the stresses). There are different ways to achieve this goal. For example, one can compute $\mathbf{F}$ using the displacements via the following equation

$$F_{ij} = \delta_{ij} + \frac{\partial u_i}{\partial X_j} = \delta_{ij} + \frac{\partial N_I}{\partial X_j} u_{iI} \tag{D.10}$$

which can be explicitly written for an element with n nodes

$$\mathbf{F} = \mathbf{I} + \begin{bmatrix} \frac{\partial N_I}{\partial X} u_{xI} & \frac{\partial N_I}{\partial Y} u_{xI} \\ \frac{\partial N_I}{\partial X} u_{yI} & \frac{\partial N_I}{\partial Y} u_{yI} \end{bmatrix} = \mathbf{I} + \begin{bmatrix} u_{x1} & u_{x2} & \dots & u_{xn} \\ u_{y1} & u_{y2} & \dots & u_{yn} \end{bmatrix} \begin{bmatrix} \frac{\partial N_1}{\partial X} & \frac{\partial N_1}{\partial Y} \\ \frac{\partial N_2}{\partial X} & \frac{\partial N_2}{\partial Y} \\ \vdots & \vdots \\ \frac{\partial N_n}{\partial X} & \frac{\partial N_n}{\partial Y} \end{bmatrix} \tag{D.11}$$

or compactly as $\mathbf{F} = \mathbf{I} + \mathbf{u}_e \boldsymbol{\mathcal{B}}_0^{\mathrm{T}}$ with $\mathbf{u}_e$ is often referred to as the element displacement matrix. We are now ready to compute the TL internal force. The algorithm is given in Algorithm 27. Note that there are different implementations that employ the second Piola-Kirchhoff stress tensor $\mathbf{S}$. We refer to the textbook of Belytschko et al. (2000) for details.

Algorithm 27 Total Lagrangian: computation of internal forces at time t.

for each element e **do**
  get element connectivity of e, $esctr$
  get element coordinates of e, $\mathbf{X}_e$
  get element displacements of e, $\mathbf{u}_e$
  initialize element internal force $[\mathbf{f}_e^{\mathrm{int}}] = \mathbf{0}$
  for each Gauss point g **do**
    compute the shape functions/derivatives $\mathbf{N}(\boldsymbol{\xi}_g)$ and $\mathbf{dNdxi}(\boldsymbol{\xi}_g)$
    compute the Jacobian $\mathbf{J} = (\mathbf{X}_e)^{\mathrm{T}}\mathbf{dNdxi}$
    compute the shape function derivatives $\mathbf{dNdx} = \mathbf{dNdxi}\mathbf{J}^{-1}$
    compute $\boldsymbol{\mathcal{B}}_0$ matrix
    compute $\mathbf{F} = \mathbf{I} + \mathbf{u}_e \boldsymbol{\mathcal{B}}_0^{\mathrm{T}}$
    compute stresses $[\mathbf{P}]$ using $\mathbf{F}$
    compute internal force $[\mathbf{f}_e^{\mathrm{int}}] = [\mathbf{f}_e^{\mathrm{int}}] + \boldsymbol{\mathcal{B}}_0^{\mathrm{T}}[\mathbf{P}]_g w_g \det \mathbf{J}$
  end for
  Assemble $[\mathbf{f}_e^{\mathrm{int}}]$ to the global internal force $\mathbf{f}^{\mathrm{int}}$ using $esctr$
end for

D.3 Implementation

We now present computer implementation aspects of the two aforementioned TL and UL FE formulations. First, we discuss how to get the coordinates and weights of Gauss integration points and compute the FE shape functions and the first derivatives (with respect to natural coordinates). In our implementation, the shape functions and derivatives are stored as (discussion is limited to 2D problems for the sake of simplicity)

$$\mathbf{N} = \begin{bmatrix} N_1 \\ N_2 \\ \vdots \\ N_n \end{bmatrix}, \quad \frac{\partial \mathbf{N}}{\partial \boldsymbol{\xi}} = \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & \frac{\partial N_1}{\partial \eta} \\ \frac{\partial N_2}{\partial \xi} & \frac{\partial N_2}{\partial \eta} \\ \vdots & \vdots \\ \frac{\partial N_n}{\partial \xi} & \frac{\partial N_n}{\partial \eta} \end{bmatrix}, \quad \frac{\partial \mathbf{N}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_1}{\partial y} \\ \frac{\partial N_2}{\partial x} & \frac{\partial N_2}{\partial y} \\ \vdots & \vdots \\ \frac{\partial N_n}{\partial x} & \frac{\partial N_n}{\partial y} \end{bmatrix} \tag{D.12}$$

where n is the number of nodes per element.

At integration point level one store at least the coordinates and weights of the integration points. In our code, they are stored in $\mathbf{Q}$ and $\mathbf{W}$ as

$$\mathbf{Q} = \begin{bmatrix} \xi_1 & \eta_1 \\ \xi_2 & \eta_2 \\ \xi_3 & \eta_3 \\ \xi_4 & \eta_4 \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} \tag{D.13}$$

The code is given in Listing D.1.

Listing D.1 Functions to get Gauss rule and compute shape functions.

```
% get the Gauss quadrture points and weights
% noGP = # Gauss points in each direction,
% nsd  = # of spatial dimensions (2 or 3)
[W,Q]      = quadrature( noGP, 'GAUSS', nsd );
% for triangle elements, use the following instead
[W,Q]      = quadrature( noGP, 'TRIANGLE', 2 );
% shape functions and derivatives at one GP, for example pt=Q(1,:)
[N,dNdxi] = lagrange_basis(elemType,pt);
```

Listing D.2 Nodal quantities .

```
elements = zeros(elemCount,4);          % element connectivity
nodes    = zeros(nodeCount,2);          % node coordinates
nmass    = zeros(nodeCount,nodeCount);  % lumped mass matrix
nacce    = zeros(nodeCount,2);          % nodal  acceleration
nvelo    = zeros(nodeCount,2);          % nodal  velocity vector
ndisp    = zeros(nodeCount,2);          % nodal displacement vector
niforce  = zeros(nodeCount,2);          % nodal internal force vector
neforce  = zeros(nodeCount,2);          % nodal external force vector
```

Next, we discuss data structures to store nodal quantities that include node coordinates `nodes`, element connectivity `elements` (for simplicity we deal with meshes consisting of one element type only), accelerations, velocities, displacements, internal and external forces. The corresponding code is shown in Listing D.2.

Computation of the lumped mass matrix using the row-sum method is given in Listing D.3.

Listing D.3 Computing the lumped mass matrix.

```
1  for e=1:elemCount
2    esctr = elements(e,:);
3    enode = nodes(esctr,:);
4    for p=1:length(W) % loop over Gauss point
5      pt        = Q(p,:);
6      [N,dNdxi]= lagrange_basis(elemType,pt);   % element shape functions
7      J0        = enode'*dNdxi;                 % element Jacobian matrix
8      detJ      = det(J0);
9      rm        = N * N' * rho * detJ * W(p);
10     nmass(esctr,esctr) = nmass(esctr,esctr) + rm;
11   end
12 end
13 nmassd = 1./sum(nmass,1)'; % already inversed
```

Listing D.4 presents the code to compute the internal force using a TL formulation. To be concrete we use a Neo-Hookean material. Line 15 illustrates the assembly of element quantities into global quantities. This assembly is also known as a *scatter* operation. Line 3 is a typical *gather* operation–the element quantities are gathered from the global quantities. Scatter and gather operations are exactly the same with linear finite elements and the readers are referred to e.g. Hughes (1987), Belytschko et al. (2000) for details. By replacing the nominal stress by the Cauchy stress one obtains a UL formulation. Note that in a UL formulation we constantly update the node coordinates (`nodes`). Listing D.5 presents the code to update velocities, displacements and mesh (for UL only). Note that step 4 is to implement the leapfrog method and step 10 is not carried out if a total Lagrangian formulation is adopted.

Listing D.4 Computing the internal force (specialized for Neo-Hookean materials).

```
1  for e=1:elemCount % loop over elements
2    esctr = elements(e,:);
3    enode = nodes(esctr,:);
4    ue    = ndisp(esctr,:)';
5    for p=1:length(W) % loop over integration points
6      pt        = Q(p,:);
7      [N,dNdxi]= lagrange_basis(elemType,pt);   % element shape functions
8      J0        = enode'*dNdxi;                 % element Jacobian matrix
9      dNdx      = dNdxi/J0;                     % equals B0^T
10     wt        = W(p)*det(J0);
11     F         = identity + ue*dNdx;           % gradient deformation F
12     invF      = inv(F); detF = det(F);
13     P         = mu*invF*(F*F'-identity) + lambda*log(detF)*invF;
14     % internal force
15     niforce(esctr,:) = niforce(esctr,:) + wt*dNdx*P;
16     % external force ...
17   end
18 end
```

Listing D.5 Updating velocities, displacements and mesh.

```
nforce      = niforce + neforce;
nacce(:,1) = nforce(:,1).*nmassd;
nacce(:,2) = nforce(:,2).*nmassd;
if (istep==0), nacce = 0.5*nacce; end
nvelo       = nvelo + nacce*dtime;
% boundary conditions
nvelo(bGrid.INodes,1)  = 0.;
deltaU      = dtime*nvelo;          % displacement increment
ndisp       = ndisp + deltaU;       % displacement at the end of time step
nodes       = nodes + deltaU;       % update mesh for UL
% advance to the next time step
t           = t + dtime;
```

D.4 Examples

Herein we present some examples so that readers can verify their own implementations. We use the method of manufactured solutions presented in Sects. 9.2 and 9.2.2 to test the implementation and convergence of the UL/TL finite elements in one (Sect. D.4.1) and two dimensions (Sect. D.4.2). Finally, we solve a problem that involves large tensile stress (Sect. D.4.3). Some MPM variants cannot solve this problem due to numerical fracture but the FEM works well with even coarse meshes.

D.4.1 One Dimensional Convergence Test

As a test to verify the implementation of the total Lagrangian FE method, we consider the one dimensional problem presented in Sect. 9.2.1 with the data: $E = 10^7$ Pa, $\nu = 0$, $\rho = 1000$ kg/m^3. The maximum displacement amplitude G takes two values: $G = 0.0001$ m and $G = 0.01$ m that corresponds to small and large deformation cases, respectively. Recall that we used a manufactured displacement field to obtain a corresponding body force which is used in our FEM code to obtain the numerical displacements $u^h(x,t)$. The space domain $[0, 1]$ is discretized by two-noded linear elements and a leapfrog time integration is used to advance solutions in time.

As the problem is time dependent there are different choices for the error norms. Herein we use the maximum norm defined as

$$L_\infty = \max_{0\le t\le T} ||e(t)||_{L_2} = \max_t \left(\int_0^1 \left[u^{ex}(x,t) - u^h(x,t) \right]^2 \mathrm{d}x \right)^{1/2} \tag{D.14}$$

which means that, for each time step a L_2 error is computed and one takes the maximum error; T is the simulation time which is 0.02 s. The L_2 norm involves an

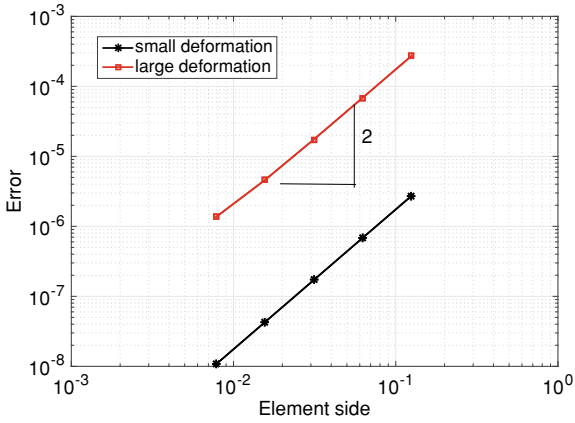


Fig. D.1 Plot of L_∞ displacement error norms with respect to mesh refinement for small and large deformation with the total Lagrangian FEM. The same time step of $0.2h/c$ was used

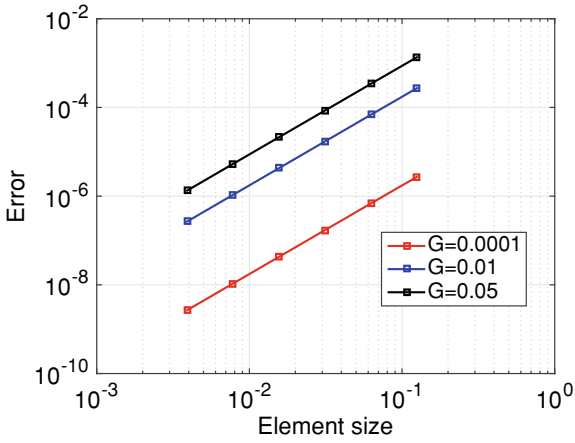


Fig. D.2 Plot of L_∞ displacement error norms with respect to mesh refinement for small and large deformation with the updated Lagrangian finite elements. The same time step of $0.2h/c$ was used

integral which is computed using the Gauss quadrature adopted in the computation of the internal force.

The results are given in Fig. D.1 (the M-file for this test is **fem/femTLMMS.m**). A time step $\Delta t = 0.2h/c$ where h denotes the element size which is $h = 1/2^m$ with $m = 3, 4, 5, 6, 7$ was used for both G. Optimal convergence rate of 2 for linear elements was obtained. The results obtained with the updated Lagrangian FE code are given in Fig. D.2 (the corresponding M-file is **femULMMS.m**).

D.4.2 Two Dimensional Convergence Test

In this section, we consider the unit square problem of which the manufactured solution was presented in Sect. 9.2.2. Recall that the manufactured displacement field is assumed to be

$$\begin{aligned} u_1(\mathbf{X}, t) &= G \sin(\pi X) \sin(c\pi t) \\ u_2(\mathbf{X}, t) &= G \sin(\pi Y) \sin(c\pi t + \pi) \end{aligned} \tag{D.15}$$

where G is the maximum amplitude of the displacement; $c = \sqrt{E/\rho_0}$ and E denotes the Young's modulus. The period is thus given by $T = {}^{2\pi}/_{c\pi}$; $\mathbf{X} = (X, Y)$ denotes the material coordinates i.e., coordinates in the reference configuration.

From Sect. 9.2.2, the body force is given by

$$\begin{aligned} b_1(\mathbf{X}, t) &= \frac{\pi^2 u_1(X, t)}{\rho_0} \left[\frac{\lambda}{F_{11}^2} (1 - \ln(F_{11}F_{22})) + \mu \left(1 + \frac{1}{F_{11}^2} \right) - E \right] \\ b_2(\mathbf{X}, t) &= \frac{\pi^2 u_2(Y, t)}{\rho_0} \left[\frac{\lambda}{F_{22}^2} (1 - \ln(F_{11}F_{22})) + \mu \left(1 + \frac{1}{F_{22}^2} \right) - E \right] \end{aligned} \tag{D.16}$$

And it is this body force to be used in a 2D TL FE code to determine the corresponding numerical displacements. And the boundary conditions are

$$\begin{aligned} v_1(0, y, t) &= v_1(1, y, t) = 0 \\ v_2(x, 0, t) &= v_2(x, 1, t) = 0 \end{aligned} \tag{D.17}$$

or in words, the normal components of the velocity are set to zeroes at the boundaries of the square. Besides, one needs to enforce the initial velocity conditions.

Material data are $E = 10^7$ Pa, $\nu = 0.3$, $\rho = 1000$ kg/m^3. The results, obtained with the TL formulation, are depicted in Fig. D.3 for two cases: $G = 0.0001$ m and $G = 0.05$ m. Four meshes of 8×8, 16×16, 32×32 and 64×64 four-noded quadrilateral elements are considered. Optimal convergence rate of 2 was obtained. The M-file of this example is **femTLMMS2D.m**.

Similar results were obtained with updated Lagrangian finite element of which the M-file is **femULMMS2D.m**. There is one thing that warrants a further discussion when the MMS is used with an UL formulation. Recall that the manufactured body force is a function of the material coordinates i.e., $\mathbf{b}(\mathbf{X}, t)$. Therefore the external force due to the MMS body force is computed in the initial configuration i.e., a TL form for the body force is used in an UL code. For example the x-component of the external force is computed as

$$\mathbf{f}_1^{\text{ext}} = \int_{\Omega_0} \rho_0 \mathbf{N} b_1(\mathbf{X}, t) \mathrm{d}\Omega \tag{D.18}$$

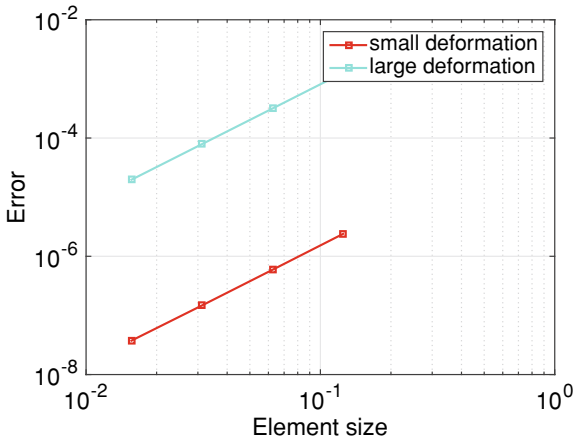


Fig. D.3 Plot of L_∞ displacement error norms with respect to mesh refinement for small and large deformation with the total Lagrangian finite elements. The same time step of $0.2h/c$ was used

To this end we store the initial node coordinates in `nodes0` and use it to compute the external force. And `nodes0` is also used to calculate the material coordinates $\mathbf{X}$ of a Gauss point in the determination of the exact displacement solution $\mathbf{u}^{\text{ex}}(\mathbf{X}, t)$.

D.4.3 Large Deformation Vibration of a Cantilever Beam

We consider yet another problem that involves large tensile stress which can reveals shortcomings of a numerical method in handling numerical fracture. This problem is a cantilever beam which is soft and subjected to a large gravity force; the tensile stress at the top surface near the left end support can cause numerical fracture (Fig. D.4). Sadeghirad et al. (2011) presented this example for the first time in the MPM literature but similar problems appeared earlier in the SPH literature.

Material (Neo-Hookean) data are $E = 10^6$ Pa, $\nu = 0.3$, $\rho = 1050$ kg/m^3. The beam is discretized by 12×3 Q4 elements with 2×2 Gauss point quadrature rule. Large deformation vibration of this highly compliant cantilever beam under its weight is induced by suddenly applying gravity ($g = 10$ m/s^2) at $t = 0$ s. This example is analyzed using constant time steps of 0.002 s, which is about $0.2h/c$ where h denotes

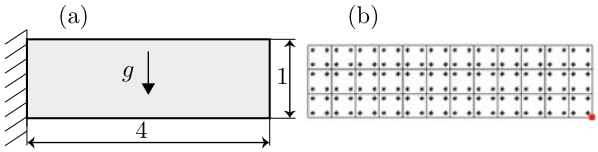


Fig. D.4 Cantilever beam problem: **a** geometry and **b** numerical model. Length unit is mm and the red point denotes the node of which the vertical displacement is recorded

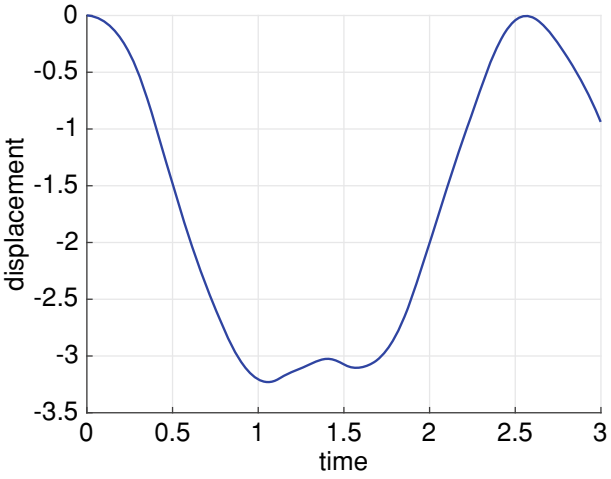


Fig. D.5 Cantilever beam problem: vertical displacement of the marked node in time

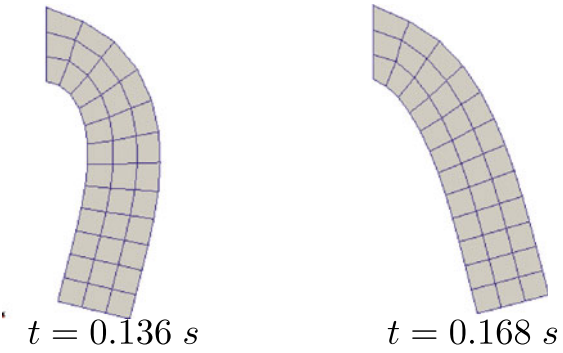


Fig. D.6 Cantilever beam problem: two simulation snapshots

the element size, and real-time simulation is $T = 3$ s. The M-files are **femULVibratingCantilever.m** and **femTLVibratingCantilever.m**. Evolution of the displacement of the marked point in time is given in Fig. D.5 whereas some simulation snapshots are shown in Fig. D.6.

Appendix E
Implicit Lagrangian Finite Elements

E.1 Implicit Dynamics FEM

Since the MPM can be considered as a FEM with moving integration points, the implicit dynamics formulation for the MPM can be obtained from the FEM formulation with appropriate modifications. A general formulation of an implcit dynamics ULFEM is first presented in Sect. E.1.1 and simplification to the case of linear elastic materials is given in Sect. E.1.2. This treatment of linear materials will facilitate the implementation of an implicit dynamics MPM formulation and makes the transition to the general nonlinear case is more straightforward.

E.1.1 General Case

We first recall the implicit dynamics finite element formulation following Belytschko et al. (2000). Let's denote $\mathbf{a}$ and $\mathbf{d}$ the nodal accelerations and displacements, respectively; and $\mathbf{f}^{\text{int}}$ the nodal internal force vector and $\mathbf{f}^{\text{ext}}$ the external force. The semi-discrete equations that we need to solve is the second Newton's law that reads

$$\mathbf{M}\mathbf{a}^{t+\Delta t} = \mathbf{f}^{\text{ext}}(\mathbf{d}^{t+\Delta t}) - \mathbf{f}^{\text{int}}(\mathbf{d}^{t+\Delta t}) \tag{E.1}$$

where $\mathbf{M}$ is the consistent mass matrix which is written as

$$\mathbf{M}_{IJ} = \mathbf{I}\int_{\Omega} \rho N_I N_J \mathrm{d}\Omega \tag{E.2}$$

and $\mathbf{I}$ is the identity matrix.

To solve Eq. (E.1) we first define the following residual

$$\mathbf{r}(\mathbf{d}^{t+\Delta t}) \equiv \mathbf{M}\mathbf{a}^{t+\Delta t} + \mathbf{f}^{\text{int}}(\mathbf{d}^{t+\Delta t}) - \mathbf{f}^{\text{ext}}(\mathbf{d}^{t+\Delta t}) = \mathbf{0} \tag{E.3}$$

N. V. Phu et al., *The Material Point Method*, Scientific Computation,
https://doi.org/10.1007/978-3-031-24070-6

Defining the predictors as

$$\begin{aligned}\tilde{\mathbf{d}}^{t+\Delta t} &= \mathbf{d}^t + \Delta t\mathbf{v}^t + \frac{(\Delta t)^2}{2}(1-2\beta)\mathbf{a}^t \\ \tilde{\mathbf{v}}^{t+\Delta t} &= \mathbf{v}^t + (1-\gamma)\Delta t\mathbf{a}^t\end{aligned} \tag{E.4}$$

Then, the displacement and velocity at the end of the time step $t + \Delta t$ are computed by

$$\begin{aligned}\mathbf{d}^{t+\Delta t} &= \tilde{\mathbf{d}}^{t+\Delta t} + \beta(\Delta t)^2\mathbf{a}^{t+\Delta t} \\ \mathbf{v}^{t+\Delta t} &= \tilde{\mathbf{v}}^{t+\Delta t} + \gamma\Delta t\mathbf{a}^{t+\Delta t}\end{aligned} \tag{E.5}$$

Note that substituting Eq. (E.4) into the above results in the standard form of the Newmark integration scheme. The updated acceleration can be computed using Eq. (E.5) as

$$\mathbf{a}^{t+\Delta t} = \frac{1}{\beta\Delta t^2}(\mathbf{d}^{t+\Delta t} - \tilde{\mathbf{d}}^{t+\Delta t}) \tag{E.6}$$

Substituting Eq. (E.6) into Eq. (E.3) gives the following equation which is a nonlinear algebraic system of equations in the nodal displacement $\mathbf{d}^{t+\Delta t}$

$$\mathbf{r} = \frac{1}{\beta\Delta t^2}\mathbf{M}(\mathbf{d}^{t+\Delta t} - \tilde{\mathbf{d}}^{t+\Delta t}) + \mathbf{f}^{\text{int}}(\mathbf{d}^{t+\Delta t}) - \mathbf{f}^{\text{ext}}(\mathbf{d}^{t+\Delta t}) = \mathbf{0} \tag{E.7}$$

Solving this equation using the Newton-Raphson method yields $\mathbf{d}^{t+\Delta t}$. After that, the acceleration $\mathbf{a}^{t+\Delta t}$ is determined by Eq. (E.6). Finally, the velocity $\mathbf{v}^{t+\Delta t}$ is determined through Eq. (E.5). The undamped, unconditionally stable, and second-order in time trapezoidal integration scheme is obtained with $\beta = 0.25$ and $\gamma = 0.5$.

According to the iterative Newton-Raphson method, at time step t and iteration $k+1$, the residual $\mathbf{r}$ can be approximated as

$$\mathbf{r}(\mathbf{d}_{k+1}^{t+\Delta t}) \simeq \mathbf{r}(\mathbf{d}_k^{t+\Delta t}) + \left.\frac{\partial\mathbf{r}}{\partial\mathbf{d}}\right|_{\mathbf{d}_k^{t+\Delta t}} \Delta\mathbf{d}_{k+1}^{t+\Delta t} = 0 \tag{E.8}$$

Hence we obtain the linearized model which allows to compute the displacement increments $\Delta\mathbf{d}_{k+1}^{t+\Delta t}$

$$\mathbf{K}^{\text{T}}(\mathbf{d}_k^{t+\Delta t})\Delta\mathbf{d}_{k+1}^{t+\Delta t} = -\mathbf{r}(\mathbf{d}_k^{t+\Delta t}) \tag{E.9}$$

where

$$\mathbf{K}^{\text{T}} = \frac{1}{\beta\Delta t^2}\mathbf{M} + \frac{\partial\mathbf{f}^{\text{int}}}{\partial\mathbf{d}} - \frac{\partial\mathbf{f}^{\text{ext}}}{\partial\mathbf{d}} \equiv \frac{1}{\beta\Delta t^2}\mathbf{M} + \mathbf{K}^{\text{mat}} + \mathbf{K}^{\text{geo}} \tag{E.10}$$

This matrix is called the finite element Jacobian of the system of equation or most often the *tangent stiffness matrix*. Expression of the material tangent $\mathbf{K}^{\text{mat}}$ and geo-

metric tangent $\mathbf{K}^{\text{geo}}$ will be given shortly. Note that, for simplicity, we have assumed that the external force is independent of the displacements. If this is not the case, there would be another term in the tangent stiffness matrix.

Having obtained the displacement increment, the displacements are updated as follows

$$\mathbf{d}_{k+1}^{t+\Delta t} = \mathbf{d}_{k}^{t+\Delta t} + \Delta \mathbf{d}_{k+1}^{t+\Delta t} \tag{E.11}$$

and this process is repeated until a predefined convergence criterion is satisfied.

Algorithm 28 presents a flowchart for implicit transient nonlinear dynamics using the updated Lagrangian formulation. Except line 1, the flowchart corresponds to the iterative solution phase to advance the solution in time from t to $t + \Delta t$. Line 1 was introduced to emphasize that the FE mass matrix is constant and only needs to be computed once in the beginning of the simulation. This is different from the MPM. When updating the nodal displacements (line 12) the mesh is updated as well. Note that we used, for illustrative purpose only, a simple convergence criterion based on the magnitude of the displacement increments (line 13). Readers are referred to the textbook Belytschko et al. (2000) for a comprehensive discussion on this. Finally, note that this flowchart is by no means unique. Different implementations exist.

Algorithm 28 Implicit dynamics FEM (Updated Lagrangian) with Newmark integration scheme

1: Form the (constant) mass matrix $\mathbf{M}$
2: Initial value for displacements: $\mathbf{d}^{t+\Delta t} = \mathbf{d}^{t}$
3: Compute $\tilde{\mathbf{d}}^{t+\Delta t} = \mathbf{d}^{t} + \Delta t \mathbf{v}^{t} + 0.5\Delta t^2(1-2\beta)\mathbf{a}^{t}$
4: **while** $err > \epsilon$ **do**
5: Compute $\mathbf{a}^{t+\Delta t} = 1/(\beta \Delta t^2)(\mathbf{d}^{t+\Delta t} - \tilde{\mathbf{d}}^{t+\Delta t})$
6: Compute $\mathbf{K}(\mathbf{d}^{t+\Delta t}) = 1/(\beta \Delta t^2)\mathbf{M} + \mathbf{K}^{\text{mat}} + \mathbf{K}^{\text{geo}}$
7: Compute stress $\boldsymbol{\sigma}(\mathbf{d}^{t+\Delta t})$ and material tangent $\mathbf{D}$ at Gauss points
8: Compute $\mathbf{f}^{\text{int}}(\mathbf{d}^{t+\Delta t})$ using $\boldsymbol{\sigma}(\mathbf{d}^{t+\Delta t})$
9: Compute $\mathbf{r} = \mathbf{M}\mathbf{a}^{t+\Delta t} + \mathbf{f}^{\text{int}}(\mathbf{d}^{t+\Delta t}) - \mathbf{f}^{\text{ext}}(\mathbf{d}^{t+\Delta t})$
10: Modify $\mathbf{K}$ for Dirichlet boundary conditions
11: Solve for displacement increment $\Delta \mathbf{d} = -\mathbf{K}^{-1}\mathbf{r}$
12: Update displacements $\mathbf{d}^{t+\Delta t} = \mathbf{d}^{t+\Delta t} + \Delta \mathbf{d}$
13: Compute error $err = ||\Delta \mathbf{d}||$
14: **end while**
15: Compute $\mathbf{v}^{t+\Delta t} = \mathbf{v}^{t} + (1-\gamma)\Delta t \mathbf{a}^{t} + \gamma \Delta t \mathbf{a}^{t+\Delta t}$
16: Compute $\mathbf{a}^{t+\Delta t} = 1/(\beta \Delta t^2)(\mathbf{d}^{t+\Delta t} - \tilde{\mathbf{d}}^{t+\Delta t})$
17: Advance to next time step $t = t + \Delta t$

The internal force vector, the material tangent and geometric tangent stiffness matrices are given by Belytschko et al. (2000)

$$\begin{aligned}
(\mathbf{f}_I^{\text{int}})_k^{t+\Delta t} &= \int_{\Omega_k^{t+\Delta t}} \mathbf{B}_I^{\mathrm{T}} \boldsymbol{\sigma} \, \mathrm{d}\Omega \\
(\mathbf{K}_{IJ}^{\text{mat}})_k^{t+\Delta t} &= \int_{\Omega_k^{t+\Delta t}} \mathbf{B}_I^{\mathrm{T}} \mathbf{D} \mathbf{B}_J \, \mathrm{d}\Omega \\
(\mathbf{K}_{IJ}^{\text{geo}})_k^{t+\Delta t} &= \mathbf{I} \left(\int_{\Omega_k^{t+\Delta t}} \boldsymbol{\mathcal{B}}_I^{\mathrm{T}} [\boldsymbol{\sigma}] \boldsymbol{\mathcal{B}}_J \, \mathrm{d}\Omega \right)
\end{aligned} \tag{E.12}$$

where $\mathbf{I}$ is the identity matrix of dimension 3×3 (in 3D) and the strain-displacement matrices are given by

$$\mathbf{B}_I = \begin{bmatrix} N_{I,x} & 0 & 0 \\ 0 & N_{I,y} & 0 \\ 0 & 0 & N_{I,z} \\ 0 & N_{I,y} & N_{I,z} \\ N_{I,x} & 0 & N_{I,x} \\ N_{I,y} & N_{I,x} & 0 \end{bmatrix}, \quad \boldsymbol{\mathcal{B}}_I = \begin{bmatrix} N_{I,x} & N_{I,y} & N_{I,z} \end{bmatrix}^{\mathrm{T}} \tag{E.13}$$

Implementation of the material tangent stiffness is identical to linear FEM while the geometric tangent stiffness requires a further elaboration. We use a concrete example of constant strain elements to illustrate the implementation. The geometric tangent stiffness of a three-node triangular element is given by

$$\mathbf{K}^{\mathrm{geo}} = \begin{bmatrix} H_{11} & 0 & H_{12} & 0 & H_{13} & 0 \\ 0 & H_{11} & 0 & H_{12} & 0 & H_{13} \\ H_{21} & 0 & H_{22} & 0 & H_{23} & 0 \\ 0 & H_{21} & 0 & H_{22} & 0 & H_{23} \\ H_{31} & 0 & H_{32} & 0 & H_{33} & 0 \\ 0 & H_{31} & 0 & H_{32} & 0 & H_{33} \end{bmatrix} \tag{E.14}$$

where $\mathbf{H}$, a 3×3 matrix, is given by

$$\mathbf{H} = \int_{\Omega} \begin{bmatrix} N_{1,x} & N_{1,y} \\ N_{2,x} & N_{2,y} \\ N_{3,x} & N_{3,y} \end{bmatrix} [\boldsymbol{\sigma}] \begin{bmatrix} N_{1,x} & N_{2,x} & N_{3,x} \\ N_{1,y} & N_{2,y} & N_{3,y} \end{bmatrix} \mathrm{d}\Omega \tag{E.15}$$

which is computed using a Gauss quadrature scheme and components of $\mathbf{H}$ are distributed properly to determine $\mathbf{K}^{\mathrm{geo}}$ via Eq. (E.14). The provided development is sufficient for implementing other 2D elements.

E.1.2 Linear Case

For the case of linear elastic materials, the internal force is simply the multiplication of $\mathbf{K}^{\mathrm{mat}}$ and $\mathbf{d}^{t+\Delta t}$. Thus, Eq. (E.7) becomes

$$\frac{1}{\beta \Delta t^2} \mathbf{M}(\mathbf{d}^{t+\Delta t} - \tilde{\mathbf{d}}^{t+\Delta t}) + \mathbf{K}^{\mathrm{mat}} \mathbf{d}^{t+\Delta t} = \mathbf{f}^{\mathrm{ext}} \tag{E.16}$$

or in the following simpler form

$$\left(\frac{1}{\beta \Delta t^2} \mathbf{M} + \mathbf{K}^{\mathrm{mat}} \right) \mathbf{d}^{t+\Delta t} = \mathbf{f}^{\mathrm{ext}} + \frac{1}{\beta \Delta t^2} \mathbf{M} \tilde{\mathbf{d}}^{t+\Delta t} \tag{E.17}$$

which can be solved for the updated nodal displacements. Next one computes the acceleration $\mathbf{a}^{t+\Delta t}$ by Eq. (E.6) and finally, the velocity $\mathbf{v}^{t+\Delta t}$ is determined through Eq. (E.5) which completes the solution phase.

The procedure given in Algorithm 28 is simplified to the one given in Algorithm 29 which, except line 1, corresponds to the solution phase for the time step t to $t + \Delta t$. It should be emphasized that in a general implementation one would store both the old and new nodal quantities, e.g. $\mathbf{a}^t$ is the old accelerations (at the end of the previous step or the beginning of the current step) and $\mathbf{a}^{t+\Delta t}$ for the new accelerations at the end of the time step. At the end of a time step the old values and the new values are swapped. One reason of this implementation is to allow to resolve the current time step due to configuration changes caused by e.g. new crack initiation.

Algorithm 29 Implicit linear dynamics FEM with Newmark integration scheme

1: Form the (constant) mass matrix $\mathbf{M}$
2: **Advance from** t **to** $t + \Delta t$
3: Compute $\tilde{\mathbf{d}}^{t+\Delta t} = \mathbf{d}^t + \Delta t \mathbf{v}^t + 0.5\Delta t^2(1 - 2\beta)\mathbf{a}^t$
4: Compute $\mathbf{K}^{\text{mat}}$
5: Compute $\mathbf{K} = 1/(\beta \Delta t^2)\mathbf{M} + \mathbf{K}^{\text{mat}}$
6: Compute RHS vector $\mathbf{f} = \mathbf{f}^{\text{ext}} + 1/(\beta \Delta t^2)\mathbf{M}\tilde{\mathbf{d}}^{t+\Delta t}$
7: Modify $\mathbf{K}$ for Dirichlet boundary conditions
8: Solve for updated displacement $\mathbf{d}^{t+\Delta t} = \mathbf{K}^{-1}\mathbf{f}$
9: Compute $\mathbf{a}^{t+\Delta t} = 1/(\beta \Delta t^2)(\mathbf{d}^{t+\Delta t} - \tilde{\mathbf{d}}^{t+\Delta t})$
10: Compute $\mathbf{v}^{t+\Delta t} = \mathbf{v}^t + (1 - \gamma)\Delta t \mathbf{a}^t + \gamma \Delta t \mathbf{a}^{t+\Delta t}$
11: **end**
12: **Advance to next time step** $t = t + \Delta t$
13: Swap acceleration $\mathbf{a}^t = \mathbf{a}^{t+\Delta t}$
14: **end**

E.2 Implementation

We present herein the computer implementation of the implicit UL finite element formulation for solid mechanics. Data structures for nodal quantities (accelerations, velocities and displacements) are given in Listing E.1. In contrast to explicit FEs, the nodal quantities are stored as column vectors (not matrices). But this decision is just for implementation convenience. Code of the processing step is given in Listing E.2. To simplify the argument list of the two functions `computeTangentMatrix` and `computeExternalForce` we store the mesh (node coordinates, element connectivity etc..) in variable `mesh`—a structure. We refer to the source code for details on these two functions.

Listing E.1 Nodal quantities.

```
ndof      = 2;
dofCount  = nodeCount*ndof;
% nodal accelerations, velocities, displacements at time 't'
nacce0    = zeros(dofCount,1);  % nodal  acceleration
nvelo0    = zeros(dofCount,1);  % nodal  velocity vector
ndisp0    = zeros(dofCount,1);  % nodal displacement vector
% nodal accelerations, velocities, displacements at time 't+dtime'
nacce     = zeros(dofCount,1);  % nodal  acceleration
nvelo     = zeros(dofCount,1);  % nodal  velocity vector
ndisp     = zeros(dofCount,1);  % nodal displacement vector
% consistent mass matrix
massMat   = zeros(dofCount,dofCount); % consistent mass matrix
```

Listing E.2 Processing step with Newmark method and Newton-Raphson method.

```
while ( t < time )
  dtilde = ndisp + dtime*nvelo + 0.5*dtime2*(1-2*beta)*nacce;
  % Newton-Raphson iterations to solve for d(t+dtime)
  error     = 1;
  while error > tol % tol: user-selected value
    iiter     = iiter + 1;
    nacce     = 1/(beta*dtime2)*(ndisp-dtilde);
    % compute the tangent and internal force
    [geoMat, stiffMat, fint] = computeTangentMatrix(mesh, material, ndisp);
    % compute the external force
    [fext] = computeExternalForce(mesh);
    % modified stiffness matrix
    K       = 1/(beta*dtime2)*massMat + stiffMat + geoMat;
    % Residual vector
    res     = massMat*nacce + fint - fext;
    % Boundary conditions
    [K,res]   = applyDirichletBCs(K,res,udofs,vdofs,uFixed,vFixed);
    % Solving for displacement increment
    ddu       = -K\res;
    % Update displacements and do not forget to update the mesh
    ndisp     = ndisp + ddu;
    mesh.node = mesh.node + [ddu(1:2:dofCount) ddu(2:2:dofCount)];
    error     = norm(ddu); % convergence criterion
  end
  % update acceleration/velocity
  nacce     = 1/(beta*dtime2)*(ndisp-dtilde);
  nvelo     = nvelo0 + (1-gamma)*dtime*nacce0 + gamma*dtime*nacce;
  % advance to the next time step
  t         = t + dtime; istep     = istep + 1;
  % swap old/new quantities
  nacce0    = nacce;
  nvelo0    = nvelo;
  ndisp0    = ndisp;
end
```

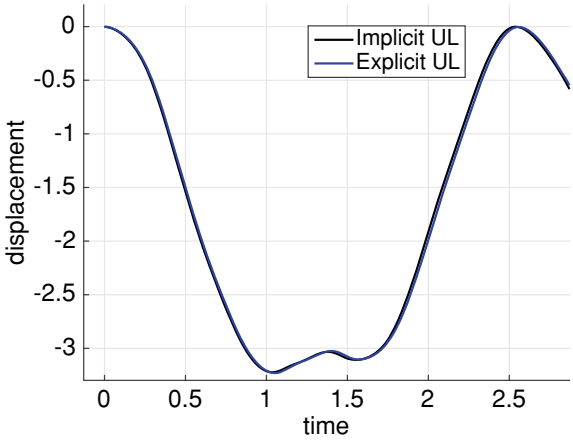


Fig. E.1 Cantilever beam problem: vertical displacement of the marked node in time with implicit method ($\Delta t = 2h/c$) and explicit method ($\Delta t = 0.5h/c$)

E.3 Examples

We reconsider the compliant cantilever beam studied in Sect. D.4.3 using explicit finite element methods. The M-file is **femImpVibratingCantilever.m**. Evolution of the displacement of the marked point in time is given in Fig. E.1, which verifies the implementation of the implicit ULFEM.

Appendix F
Implementing the Material Point Method Using Julia

Many researchers, including us, today do their day-to-day work in dynamic languages such as `Matlab`, `Python`, `Mathmematica`. The reasons are several: (i) these languages are easy to use, (ii) they provide a friendly user interface that integrates computing and graphics into one single platform, (iii) they are ideal for rapid prototyping and (iv) they can be used perfectly for educational purposes. However, the resulting codes are usually slow and not suitable for computationally intensive problems; problems that are suitable for static languages such as `Fortran` and `C/C++`. To solve this problem, one usually resorts to the two language programming paradigm in which a high level programming language is used for certain portion of the code and a low level language, e.g. `Fortran` or `C++`, is used for another portion of the code: the hotspot.

The 'ideal' programming language, from the point of view of a researcher, is the one that is as easy to use as `Matlab/Python` and as fast as `Fortran/C++` so that time would be spent on testing new scientific ideas rather than on studying difficult programming topics and code optimization techniques. This 'ideal' programming language would allow researchers to not only do prototypes but also solve their large-scale models within the same language, instead of resorting to two language solutions when performance is needed. In the search for such a language, `Julia` was created in 2012 (Bezanson et al. 2012, 2014). `Julia` is designed to be easy and fast thanks to the LLVM-based just-in-time (JIT) compilation (Lattner and A. 2004). In other words, with `Julia`, one can have machine performance without sacrificing human convenience (Bezanson et al. 2012).

Sinaie et al. (2017) programmed a (UL)MPM code written in `Julia` to verify the efficiency claims of the Julia community. They showed that a quick `Julia` implementation was eight times faster than the Matlab code presented in Chap. 6. That MPM code written in `Julia` 0.7.0 is, however, not compatible to the current version, at the time of writing this book, of `Julia` (`Julia` 1.7.2).

The aim of this appendix are multi-fold. First, we re-write that code using `Julia` 1.7.2. Second, we program another code which is more efficient and general than Sinaie's code; it also supports the GPIC. We verify its efficiency compared with our

N. V. Phu et al., *The Material Point Method*, Scientific Computation,
https://doi.org/10.1007/978-3-031-24070-6

Matlab code and Karamelo, a C++ code. Third, this appendix serves as a concise introduction to Julia in the context of computational mechanics. We refer to Xiao et al. (2021), Vigliotti and Auricchio (2021) for the use of the Julia language in computational mechanics.

We start with a short presentation of Julia in Sect. F.1. We did not attempt to be exhaustive and refer to the documentation of Julia for detail. We focus on Julia features required for the implementation of an MPM code. Then, we present our revised MPM code in Sect. F.2. This code is 5× faster than the Matlab code described in Sect. 6. Next, a much more efficient code (15× faster than the old code or 75× faster than the Matlab code) is described in Sect. F.3. Yet, this code is still slower than Karamelo. Therefore, an optimized code is given in Sect. F.4.

By presenting various versions of the code, we demonstrate the common practice of writing codes using Julia: a program is first quickly implemented similar to Matlab/Python. Next, performance is examined and modifications are made to have a really fast code. These codes are available at https://github.com/vinhphunguyen/jump. Note that we do not discuss parallel computing in this appendix. And our Julia codes only implement the ULMPM (with hat, quadratic and cubic B-splines weighting functions) and GPIC. In other words, the codes do not support the TLMPM.

F.1 A Short Introduction to Julia

We begin this introduction with installation and code editing in Sect. F.1.1. Next, we discuss the comparison between for-loops and vectorized operators (Sect. F.1.2). We then describe how a new type can be defined in Julia (Sect. F.1.3). Arrays are discussed in Sect. F.1.4, sets and dictionaries in Appendix F.1.5 and memory allocation in Sect. F.1.6. Next, we present types and multiple dispatch—a key feature of Julia in Sect. F.1.7. Type stability is treated in Sect. F.1.8. Modules, a way to avoid name collision in Julia, are treated in Sect. F.1.9.

F.1.1 Julia: Installation and Code Editor

Julia can be downloaded for free, under the MIT license, from the web page http://julialang.org/downloads. There are a couple of ways to interact with Julia. First, Julia comes with a full-featured interactive command-line REPL (read-eval-print loop) built into the julia executable (Fig. F.1). If an IDE (integrated development environment) is preferred, there is Juno at http://junolab.org, see Fig. F.2. A good document on how to get Juno and Atom is given at https://techytok.com/atom-and-juno-setup-for-julia/. Alternatively, Julia can be launched from a web browser using a Jupyter notebook, and Listing F.1 presents how to install Jupyter with Julia.

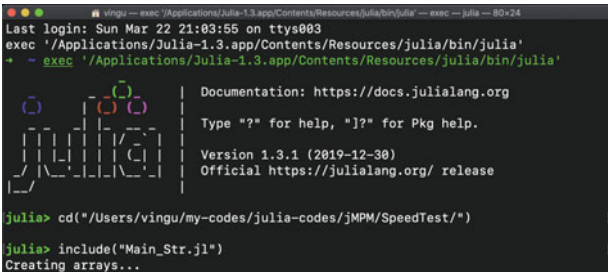


Fig. F.1 `Julia`'s command-line REPL (read-eval-print loop). We also show commands to run a certain `Julia` script. Alternatively, a script can be run directly in a terminal by tying: julia –optimize=3 script-name.jl. To remove `Julia`, rm -rf /.julia/

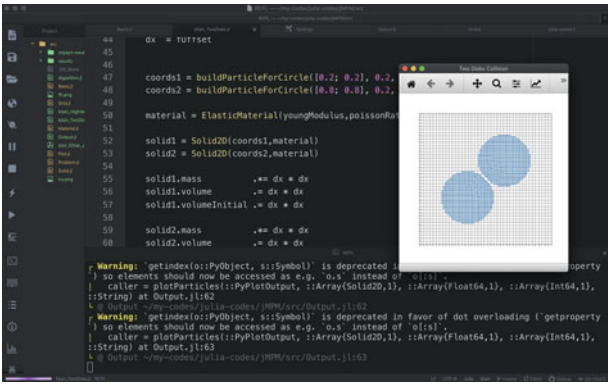


Fig. F.2 Juno IDE atom

F.1.2 Using 'for' Loops Versus Vectorization

Performing a fixed set of operations on large arrays of data (consisting of nodal data and material point data) encompasses a significant part of an MPM implementation. In `Julia`, similar to `Matlab`, such operations can be carried out using either *for-loops* or *vectorized operators*. While vectorized operators are efficient in `Matlab`, *for-loops* tend to be faster than vectorized operators in `Julia`. However, this depends on how the *for-loop* is implemented.

To check the performance of for-loops versus vectorized operators, three large arrays are created and assigned with initial values (Listing F.2). The initial values assigned to the arrays are arbitrary. Three functions in which a summation is performed on the first two arrays and assigned to the third array are coded (Listing F.3). A simple summation has been selected, so that the run time is more representative of the iterations, rather than the calculations. Each function is iterated 1000 times, whereby the time duration is monitored using the `@time` function (Listing F.4).

Table F.1 Performance comparison of *for-loops* against vectorized operators on large data arrays in `Julia`. Results obtained with single thread on an Intel Core i7-6700 CPU with a RAM of 16 GB

Array size	Type	Iterations/Second		
		i-j loop	j-i loop	Vectorized
10^6	`Real`	2.70	–	2.52
10^6	`Float64`	668.90	–	293.00
$10^6 \times 10$	`Float64`	33.39	63.29	38.05
10×10^6	`Float64`	9.17	35.57	32.21

The performance is evaluated in terms of "Iterations/run-time second" i.e., the more iterations per second, the more efficient the code is. The first observation made by these results is the poor performance of abstract data types. Although `Julia` allows the use of abstract data types such as `Real`, which has attractive design implication towards polymorphism, it is always best to declare variables using concrete types when performance is an issue. This is readily seen by comparing the first two rows of Table F.1. As a result, the `Float64` concrete data type is used for our implementation. Actually, for things that hold data, such as arrays, dictionaries, or fields in composite types, it is best to explicitly define the type that it will hold.

From Table F.1, in `Julia`, *for-loops* can perform better than vectorized operators. For one-dimensional arrays, the speed at which for loops are carried out is more than twice the speed of vectorized operators. However, for two-dimensional arrays, to have noticeable advantage over

Listing F.1 Installation of IJulia and Jupyter.

```
julia> using IJulia
julia> notebook()
install Jupyter via conda, y/n? [y]: y
```

Listing F.2 Creating three two dimensional arrays of size (10, 1000000) of `Float64`. Note that there is no semicolon ';' at the end of a statement.

```
using Printf # this is to use package Printf
println("Creating arrays...")
array1 = Array{Float64}(undef,10,1000000) # this array is not initialized
array2 = Array{Float64}(undef,10,1000000)
array3 = Array{Float64}(undef,10,1000000)
# Arrays index starts from 1
for i = 1:1:size(array1,1)
for j = 1:1:size(array1,2)
  array1[i,j] = 1.0*i
  array2[i,j] = -1.0*i
  array3[i,j] = 0
end
end
println("...done\n")
```

vectorized operators, nested *for-loops* have to be consistent with the *column-major* order of which data is stored in memory (similar to `Matlab` and `Fortran`). In other words, the inner loop should iterate over the first index of the two-dimensional array.

Such order is termed as 'j-i loop' in Listing F.3 and Table F.1. The poor performance of vectorized operators is due to array allocations which are expensive.

All the codes presented in Listing F.2–F.4 are put in a script file `Main_Vec.jl`. By convention, `Julia` scripts have names that end with `.jl`. Launch `Julia`, and type the following in the REPL to run this script

Listing F.3 Three sum functions. Note that there is no space between the function name and the argument list. Each of the sum operation is called 1000 times. Function parameters are passed by reference; if a function modifies an array, the changes will be visible in the caller. There is no need to annotate the type of arguments in Julia functions, unless if needed.

```
function sum1(r, x1, x2 )     # vectorized version
#function sum1(r::Array{Float64}, x1, x2 ) # with type declaration
  for count=1:iterations
   r = x1 + x2
  end
end
function sum2(r, x1, x2)      # loop version i-j
 ni = size(x1,1)
 nj = size(x2,2)
 for count=1:iterations
  for i = 1:ni
    for j = 1:nj
      r[i,j] = x1[i,j] + x2[i,j]
    end
  end
 end
end
function sum3(r, x1, x2)       # loop version j-i
 for count=1:iterations
   for j = 1:nj
    for i = 1:ni
      r[i,j] = x1[i,j] + x2[i,j]
    end
   end
 end
end
```

Listing F.4 Measuring the time of three functions.

```
@time sum1(array3, array1, array2)
@time sum2(array3, array1, array2)
@time sum3(array3, array1, array2)
```

```
julia> cd("juMP/SpeedTest/")
julia> include("Main_Vec.jl")
```

Albeit irrelevant here, `Julia` is very well suited for non-vectorized problems such as those arising from stochastic processes modeled by Monte Carlo methods.

F.1.3 Composite Types Versus Arrays

A composite type is a collection of named fields, an instance of which can be treated as a single value. In many languages, composite types are the only kind of user-defined type. In mainstream object oriented languages, such as `C++`, `Java`, and `Python`, composite types also have named functions associated with them. However, in `Julia`, functions are not bundled with the objects they operate on. Thus, a

composite type is similar to a *struct* in C and can be thought of as roughly equivalent to a class without behavior in object-oriented languages.

Given the common use of composite types, the performance of an array of such types are compared to the performance of multi-dimensional arrays.

First, we define a new type named `testDataStructure` (lines 4–9 in Listing F.5). Then, we create three large arrays containing one million `testDataStructure` (lines 7–9). Finally, we assign concrete values to these three arrays. To see built in types in `Julia`, in the REPL, type: `typeof(3)`, `typeof(3.0)`, you will get `Int64` and `Float64`, respectively. Note that, Julia's standard numeric types e.g. `Float64, Int64`, etc. are concrete subtypes of the abstract type `Number`. We refer to Sect. F.1.7 for a discussion on abstract types and subtypes.

Listing F.5 Definition of a new type named `testDataStructure`, and creating three arrays of this type and initializing them. '::' is the type-assertion operator; it specifies a concrete type for any variables.

```
1  #= Define a type 'testDataStructure'
2     with one member 'vMember' which is an array of 100 Float64
3  =#
4  struct testDataStructure
5    vMember::Array{Float64}
6    function testDataStructure() # this is an inner constructor
7      new(zeros(100))
8    end
9  end
10 thisDataStructure01 = Array{testDataStructure}(undef,1000000)
11 thisDataStructure02 = Array{testDataStructure}(undef,1000000)
12 thisDataStructure03 = Array{testDataStructure}(undef,1000000)
13 for i = 1:1000000
14   thisDataStructure01[i] = testDataStructure()
15   thisDataStructure02[i] = testDataStructure()
16   thisDataStructure03[i] = testDataStructure()
17   for j = 1:100
18     thisDataStructure01[i].vMember[j] = 1.0*i
19     thisDataStructure02[i].vMember[j] = -1.0*i
20     thisDataStructure03[i].vMember[j] = 0
21   end
22 end
```

Next, we define three methods to access data members shown in Listing F.6. Note that, based on previous finding, for-loops are used over vectorized operators. The results, given in Table F.2, indicate a consistent performance advantage of multi-dimensional arrays over composite types. However, Table F.2 also demonstrates that removing redundant access calls to the members of a composite type improves the performance of operations on composite types (compare column 5 with columns 3 and 4).

Listing F.6 Three methods to access data members of a composite type.

```
1  function fun1(x::Array{testDataStructure},
2                y::Array{testDataStructure},
3                z::Array{testDataStructure})
4    for count in 1:1000
5      for i = 1:1000000
6        for j = 1:100
7          z[i].vMember[j] = x[i].vMember[j] + y[i].vMember[j]
8        end
9      end
10   end
11 end
12 function fun2(x::Array{testDataStructure},
13               y::Array{testDataStructure},
```

Table F.2 Performance comparison of accessing elements of an array as opposed to members of a composite type. Methods 1–3 refer to different methods of access specified in Listing F.6

Array size	Iterations/Second			
	Array	Method 1	Method 2	Method 3
$10^6 \times 10$	58.51	17.85	22.15	25.36
$10^6 \times 100$	5.77	1.88	2.50	3.01

```
                  z::Array{testDataStructure})
  for count in 1:1000
    for i = 1:1000000
        this01 = x[i]
        this02 = y[i]
        this03 = z[i]
        for j = 1:100
          this03.vMember[j] = this01.vMember[j] + this02.vMember[j]
        end
    end
  end
end
function fun3(x::Array{testDataStructure},
                  y::Array{testDataStructure},
                  z::Array{testDataStructure})
  for count in 1:1000
    for i = 1:1000000
      this01 = x[i].vMember
      this02 = y[i].vMember
      this03 = z[i].vMember
      for j = 1:100
        this03[j] = this01[j] + this02[j]
      end
    end
  end
end
```

While the performance advantage of multi-dimensional arrays as opposed to composite types is significant enough to be taken advantage of, one also has to consider the benefits derived from composite types in terms of code organization and readability. Taking this into account, composite types are used the code presented in Appendix F.2.

Funtions. `Julia` function arguments follow a convention sometimes called "pass-by-reference", which means that values are not copied when they are passed to functions. Function arguments themselves act as new variable bindings (new locations that can refer to values), but the values they refer to are identical to the passed values. Modifications to mutable values (such as Arrays) made within a function will be visible to the caller. Listing F.7 gives a summary of `Julia` functions.

Listing F.7 Functions in `Julia`.

```
# one-line function
play(::Type{Rock}, ::Type{Paper}) = "Paper wins"
# full form
function play(a::Type{Rock}, b::Type{Paper})
  return "Paper wins"
end
# anonymous function
data["pressure"] = [(1,"force",t -> 400*exp(-10000*t))]
# modify float inputs by returning a tuple
# a tuple = immutable fixed-length container that can hold any values
```

```
function func(damage,damage_init,...)
  damage     += 1.
  damage_init += 1.
  return (damage,damage_init)
end
# function with keyword arguments
function plot(x, y; style="solid", width4=1., color="black") #=> plot(x,y,width=2)
# types of keyword arguments can be made explicit
function plot(x, y; style="solid", width::Float64=1., color="black")
```

F.1.4 Arrays

Multi-dimensional arrays are popular in engineering and sciences. This section briefly presents their use in `Julia`. See Listing F.8 for many ways of array creations and Listing F.9 for common operations on them. In `Julia`, array slices create a copy of the slice, and this can be avoided by using the macro `@view` (see line 14 in Listing F.9).

Listing F.8 Arrays in Julia.

```
using LinearAlgebra
using StaticArrays
# declare one vector and one 2D matrix of known dimensions
v1 = Array{Float64,1}(undef,3)    # un-initialized array of 3 Float64
m1 = Array{Float64,2}(undef,2,3)  # un-initialized matrix of 2x3 Float64
# initialize the vector v1
[v1[i] = 2*i for i=1:3 ]          # using comprehension
for i = 1: 3 v1[i] = 2*i end      # using for-loop
# declare one vector with unknown length
v2 = Array{Float64,1}
# fill v2
v2 = fill(0,3)                    # => v2= [3 3 3]
# declare one vector with 0 element, then add numbers to it
v3 = Array{Float64,1}()
push!(v3,1.)
# declare arrays directly
v4 = [1. 2. 3.]                   # => 2D array: 1x3 Array{Float64,2}
v5 = [1., 2., 3.]                 # => 1D array: 3-element Array{Float64,1}
a  = [1. 2.;3. 4.]                #
a6 = zeros(Int64, 5, 4)           # => 2D array: 5x4 of 0 (int)
a7 = Array{Float64}(undef, 0, 2)  # 0 row and 2 cols
a7 = [a7;[1 2]]
```

Elementwise operations. `Julia` supports methods for carrying out an operation on every element of a vector. To do this we add a period or dot before the operator.

Listing F.9 Some remarks on array operations.

```
a    = [1 2 3]
a    = a + 1    # ERROR: MethodError: no method matching +(::Array{Int64,2}, ::Int64)
# we need to use '.+' to get element-wise addition
a   .= a .+ 1   # .= : in-place assignment operator
a  .+= 1        # shorter notation of a .= a .+ 1
x    = zeros(3) # x = [0.,0.,0.]
y    = x        # b points to x, change b, change x!!!
y[1] = 1.       # x = y = [1.0,0.,0.]
meds = ["FEM","MPM","SPH"]
for m in meds
  println(m)    # "FEM", then "MPM", then "SPH"
end
b = a[1:2]      # slicing makes copy, b[1] = 10 => a=[10,2,3]
b = @view a[1:2] # using view, now b is just another name for a[1:2]
map(x-> x/sum(vector),vector)
```

F.1.5 Sets and Dictionaries

Sets and dictionaries are common datastructures. The former are best to store arrays of unique items. We demonstrate their basic usage in Listing F.10.

Listing F.10 Sets and dictionaries in Julia.

```
set1 = Set()
push!(1,set1)
push!(2,set1)
push!(1,set1)          # => set1 = [1,2]
Dict([("A", 1), ("B", 2)]) # Dict{String,Int64} with 2 entries: "B" => 2, "A" => 1
Dict("A"=>1, "B"=>2)
mutable struct FEMesh
  nodes        :: Dict{Int, Vector{Float64}} # node Id => coords
  node_sets    :: Dict{String, Set{Int}}
  elements     :: Dict{Int, Vector{Int}}     # elem id => connectivity
  element_sets :: Dict{String, Set{Int}}
end
mesh = FEMesh()
mesh.nodes[nid]           = ncoords
mesh.node_sets["boundary1"] = [1,5,7]
if haskey(mesh.element_sets, "force") end
```

F.1.6 Memory Allocation

Even though `Julia` code can be as fast as `C/Fortran` code, you have to make sure the least amount of memory allocation. There exists packages to track allocation such as `BenchmarkTools` and `TimerOutputs`. In Listing F.11, we demonstrate the usage of the former package and illustrate memory allocation occurs in common matrix operations. Use of `TimerOutputs` is shown in Listing F.12 and Fig. F.3. When unexpected allocation is present, one can use the macro `@code_warntype` to examine the code (see Sect. F.1.8). When looking at the output of this macro, pay attention to things in red such as Any, Union etc.

Listing F.11 Memory allocations arise when array operations are not properly used.

```
a = ones(10000)
b = ones(10000)
c = ones(10000)
function func1(a,b,c)
  a .= b .+ 2 .* c
end
function func2(a,b,c)
  a = b + 2*c
end
@btime func1(a,b,c) # => 27.957 ns (0 allocations: 0 bytes)
@btime func2(a,b,c) # => 108.846 ns (2 allocations: 224 bytes)
```

Listing F.12 Usage of package `TimerOutputs`.

```
reset_timer!()
@time solve_explicit_dynamics_2D(grid,solids,basis,algo2,output2,fix,Tf,dtime)
print_timer()
```

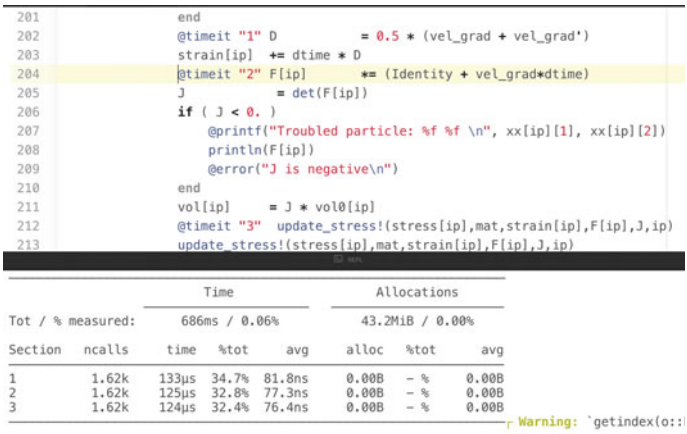


Fig. F.3 Excellent output of the time and memory allocation reported by package `TimerOutputs`

F.1.7 Types and Multiple Dispatch

For many people, the reason why they use `Julia` is "multiple dispatch". Multiple dispatch is a feature where different code is called by a function depending on the types of the arguments. Combined with the JIT (Just-in-time compiler), `Julia` will automatically compile specialized code, depending on the types of the arguments you give it. For instance, when the user writes a generic code, `f(x, y) = x + y`, without annotating the types of `x` and `y`, `Julia` just-in-time compiler will generate efficient and specialized codes for any given pair of `x` and `y` whose addition is well defined.

To demonstrate this concept, we borrow the implementation of the game 'rock-paper-scissors' by Mose Giordano at https://giordano.github.io/blog/2017-11-03-rock-paper-scissors/. The code is shown in Listing F.13. First, an abstract type `Shape` is defined; you cannot use this abstract type to define variables. Then, three concrete sub-types are defined: `Rock`, `Paper` and `Scissors`. After that, three methods (or functions) are defined: they have the same name `play`, but their arguments are different.

Listing F.13 Abstract type and sub-types.

```
abstract type Shape end

struct Rock  <: Shape end
struct Paper <: Shape end
struct Scissors<: Shape end
# methods or functions on these types
play(::Type{Rock}, ::Type{Paper}) = "Paper wins"
play(::Type{Rock}, ::Type{Scissors}) = "Rock wins"
play(::Type{Paper},::Type{Scissors}) = "Scissors wins"
play(::Type{T},  ::Type{T}) where (T<:Shape) = "Tie"
# Note that we did not define play(Paper,Rock) or play(Scissors,Rock)
play(a::Type{<:Shape},b::Type{<:Shape}) = play(b,a)
# Now, start playing
play(Scissors,Paper) # => "Scissors wins"
```

Instead of writing explicit functions for the cases of a tie, line 10 shows a generic code with a parametrized argument `T`. The `play` function as defined above is called a one-line function, its full form is given in Listing F.7, which is more verbose. The primary use for anonymous functions is passing them to functions which take other functions as arguments. A classic example is map, which applies a function to each value of an array and returns a new array containing the resulting values.

F.1.8 Type Stability

`Julia`'s main appeal is speed. But achieving peak performance in `Julia` requires that programmers understand a few subtle concepts that are generally unfamiliar to users of weakly typed languages (e.g. `Python` or `Matlab`).

One particularly subtle performance pitfall is the need to write type-stable code. Code is said to be type-stable if the type of every variable does not vary over time. To clarify this idea, consider the following function:

```
julia> function foo(x)
       x += 1.
       end
```

where the type of x is not specified. `Julia` generates a code for $x = 1$, and using the macro @code_warntype, we will see that

```
julia> @code_warntype foo(1)
Variables
  #self#::Core.Compiler.Const(foo, false)
  x@_2::Int64
  x@_3::Union{Float64, Int64}
```

That means that the type of x was changed from integer to float; type instability has occurred. Similarly, for $x = 1.0$, no type instability occurs:

```
julia> @code_warntype foo(1.)
Variables
  #self#::Core.Compiler.Const(foo, false)
  x@_2::Float64
  x@_3::Float64
```

F.1.9 Modules

To demonstrate how to create a module, we define a module named `MyModule` with one type named `MyStruct` and one function `foo`. They are exported so that they can be used externally (Listing F.14).

Listing F.14 Example of writing a module.

```
module MyModule
  using LinearAlgebra

  struct   MyStruct
    vel  :: Vector{Float64}
  end
  function foo(x::MyStruct) end

  export MyStruct
  export foo
end
```

Then, we can use this module as follows

```
julia> using MyModule
julia> x = MyStruct
julia> foo(x)
```

In the parlance of `Julia`, the struct `MyStruct` is immutable. So, `Julia` forbids directly modify a struct member entirely as follows

```
julia> using MyModule
julia> x = MyStruct(zeros(3))
julia> x.vel  = ones(3) # ERROR: setfield! immutable struct cannot be changed
julia> x.vel .= ones(3) # ok
julia> x.vel .+= 1      # ok
```

F.2 A Simple MPM Code in Julia

Having discussed some implementation choices for common operations in the MPM, we now proceed to a simple implementation of the MPM in `Julia` for solid mechanics. Composite types are used to store the attributes of grid points and material points. We start with a discussion on the code structure in Sect. F.2.1. Then, we present the composite types for the grid and particle in Sect. F.2.2. Solution algorithm is given in Sect. F.2.3 and finally, one example is provided to discuss the performance of the code (Sect. F.2.4).

Table F.3 Organization of the presented MPM code

Main.jl	Main script	379 lines (including plotting)
Basis.jl	Shape functions	53 lines
Grid.jl	Grid and grid points	104 lines
MaterialPoint.jl	Material points	139 lines

F.2.1 Code Organization

Our complete MPM code is organized into four `Julia` files as shown in Table F.3. `Julia` comes with a full-featured interactive command-line REPL built into the julia executable. To run the simulations one launches the `Julia` REPL using the julia command, and change to the directory where the source code is found and type *include("Main.jl")*. We used *@time include("Main.jl")* to measure the runtime of the examples presented in Sect. F.2.4.

F.2.2 Data Structures

We implement the concept of a grid point using the `GridPoint` struct and of a material point using the `MaterialPoint` struct. The declaration of these types are presented in Listings F.15 and F.16, respectively. Note that, for simplicity, the nodal quantity vectors e.g. positions are initialized for 2D problems.

Listing F.15 `Julia` code listing for the declaration of `GridPoint` composite type.

```
1  struct GridPoint
2    Fixed    :: Vector{Bool}    # fixation in different directions
3    Mass     :: Float64
4    Position :: Vector{Float64}
5    Momentum :: Vector{Float64}
6    Force    :: Vector{Float64} # total force=internal force+external force
7    # constructor, set all values to zero, the following is for 2D problems
8    function GridPoint()
9          new (
10              [false;false],
11              0.0,
12              zeros(2),
13              zeros(2),
14              zeros(2)
15             );
16   end
17 end
```

Next, we define an array of grid points, cf. line 2 of Listing F.17 and array(s) of particles (Listing F.17). Note that, for brevity, we just created one particle.

Listing F.16 `Julia` code listing for the declaration of `MaterialPoint` composite type.

```
struct MaterialPoint
  Mass          :: Float64
  VolumeInitial :: Float64
  Volume        :: Float64
  Centroid      :: Vector{Float64} # particle position
  Velocity      :: Vector{Float64}
  Momentum      :: Vector{Float64}
  ExternalForce :: Vector{Float64}
  DeforGradient :: Vector{Float64} # deformation grad F
  Stress        :: Vector{Float64} # strain/stress as vectors (Voigt notation)
  function MaterialPoint()
    # constructor, set all above variables to zero
  end
end
```

Listing F.17 `Julia` code listing for generation of material points. By convention, the exclamation mark "!" is appended to names of functions that modify their arguments: `push!` in this code snippet.

```
# grid creation, Lx, Ly, # nodes x, # nodes y
thisGrid                = mpmGrid(1.0, 1.0, 21, 21)
thisMaterialDomain      = Array{MaterialPoint}(0)
# repeat the following until the desired number of particles is obtained
thisMaterialPoint       = MaterialPoint()
thisMaterialPoint.Centroid = [1.0; 1.0]
# appending to the end of arrays, similar to push_back in C++ STL
push!(thisMaterialDomain, thisMaterialPoint)
```

F.2.3 Solution Phase

Now, we have the grid and some particles, it is ready to move to the solution phase. Listing F.18 presents the implementation of a 2D MPM for elastic bodies. For brevity, only the particle to grid phase was presented.

F.2.4 Examples

This section presents one numerical example to demonstrate the performance advantage of `Julia` as a development tool for the MPM. All simulations are carried out using in-house MPM codes written in `Matlab` and `Julia`. The `Matlab` code is described in Chap. 6.

There are three primary variants affecting the accuracy of MPM solutions: the time-step size, the grid density and the particle density. Decreasing the time-step size will increase the number of time-integration steps needed to complete the analysis, and hence, will increase run-time. Given the trivial relation between time-step size and run-time, i.e. a linear scaling between the two, there is no need to examine it here. However, what is worth examining is the scaling of run-time with grid and particle density. To eliminate the effect of time-step size in this process,

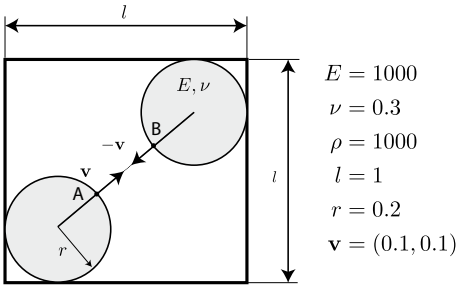


Fig. F.4 Geometry and initial conditions for the impact of two elastic bodies (units in N and mm)

Listing F.18 Particle to grid phase. All grid points and material points are stored into arrays denoted by allGP and allMP, respectively. GP: Grid Point, MP: Material Point, AGP: Adjacent Grid Point. The prefix 'this' indicates the reference of the object currently being processed.

```
for iIndex_MP in 1:length(allMaterialPoint)
  thisMP                = allMaterialPoint[iIndex_MP]
  thisAdjacentGridPoints = getAdjacentGridPoints(thisMP, thisGrid)
  for iIndex in 1:length(thisAdjacentGridPoints)
    thisGridPoint   = thisGrid.GridPoints[thisAdjacentGridPoints[iIndex]]
    N, dN           = getShapeAndGradient(thisMP, thisGridPoint, thisGrid)
    # mass, momentum, internal force and external force
    thisGridPoint.fMass    += N * thisMP.fMass
    thisGridPoint.v2Momentum += N * thisMP.fMass * thisMP.v2Velocity
    end
end
```

performance is evaluated in terms of 'Iterations/run-time seconds'. Note that each iteration constitutes a single time step of the analysis, and run-time refers to the actual time it takes to complete the analysis using a single thread on an Intel Core i7-6700 CPU with a RAM of 16 GB. We also provide the total runtime of all the conducted simulations.

We reconsider the problem of impact between two identical elastic disks (Fig. F.4). Details can be found in Sect. 6.15.2. We used the USL algorithm with a cutoff value of 10^{-8} and hat weighting functions. Note that using the MUSL gives the same result (Fig. F.5). AQ1

To demonstrate the use of graphics in `Julia`, see Sect. F.3.5 for more detail, the movement of the two disks and their impact are given in Fig. F.4. We now move to the main point: performance of `Julia` code. Table F.4 shows the performance results for different cases of number of particles and number of grid points. The `Julia` code is consistently faster than the `Matlab` code. Note that increasing the number of particles or making the grid more dense, causes the performance ratio to vary. However, for all cases, the `Julia` code is observed to perform more than 5 times faster than the `Matlab` code. In fact, the performance ratio is rather significant, especially when it comes to simulations that can potentially take hours or days to complete. For example, the simulation given in the last rows took 2.6 h with the `Matlab` code and only 0.46 h with the `Julia` code.

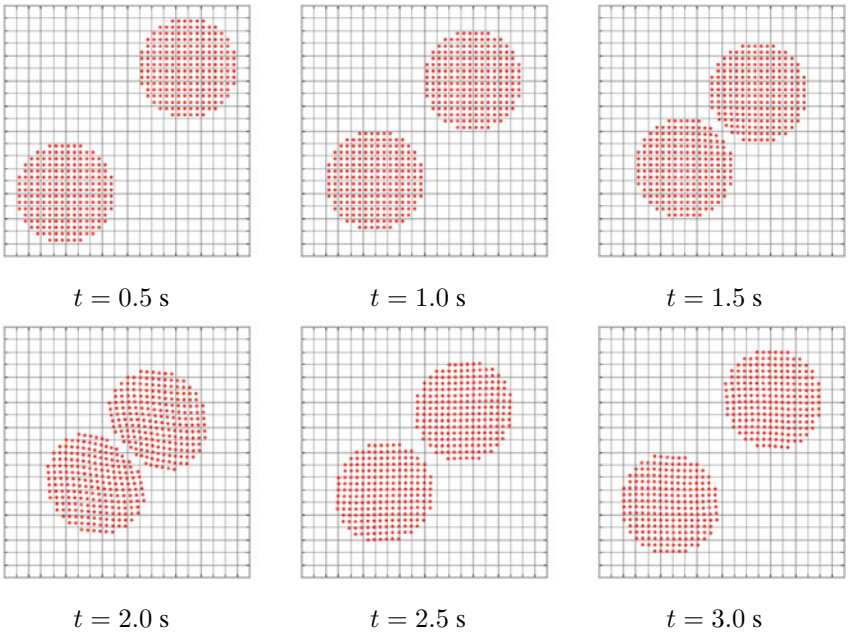


Fig. F.5 Snapshots for the impact of two elastic bodies before, during and after impact (Sinaie et al. 2017). These images were created using the PyPlot graphical package (Johnson 2012)

Table F.4 Performance comparison of `Julia` and `Matlab` code for the two-disk impact problem

Particles	Grid	Iterations/Second		Total time [s]		Ratio
		`Matlab`	`Julia`	`Matlab`	`Julia`	
416	20×20	20.16	132.80	148.81	22.59	6.59
1624	20×20	5.30	33.37	566.36	89.90	6.30
1624	40×40	5.07	26.45	591.25	113.42	5.21
25 784	80×80	0.32	1.82	9375.00	1651.07	5.68

F.3 A More Efficient Julia MPM Code

The code presented in Sect. F.2 was quickly programmed to verify whether `Julia` is fast. Furthermore, the main coder (Sina Sinaie) was new to the language. Thus, that code was not efficient and limited. We present herein a more efficient and better implementation of an explicit dynamics MPM. We name our code `juMP`.

We start with a discussion on grid and particle data structure (Sect. F.3.1). Based on the findings in Sect. F.1.3 about composite type versus array, we decided to organize grid/particle data using large arrays for the whole grid (or the whole solid). Basis functions are described in Sect. F.3.2, followed by materials in Sect. F.3.3. To handle

Table F.5 Organization of `juMP`

Basis.jl	Basis functions
Grid.jl	`Grid1D`, `Grid2D`, `Grid3D`
Solid.jl	`Solid2D`, `Solid3D`
Material.jl	Materials
Output.jl	Particle data output
Problem.jl	The problem
Algorithm.jl	Types for USL, MUSL
Bsplines.jl	Modified quadratic and cubic B-splines
Mesh.jl	Finite elements and shape functions (for GPIC)
Fem.jl	Meshed solids (for GPIC)
Dirichlet.jl	Functions to handle Dirichlet BCs
Neumann.jl	Functions to handle Neumann BCs
Main_*.jl	Input file for each simulation
*.geo, *.msh	Gmsh geo and mesh files
*.inp	Abaqus mesh files (GPIC)

different problem types e.g. solid mechanics or heat transfer, a problem type was coded (Sect. F.3.4). Post-processing is discussed in Sect. F.3.5 and a complete setup for a typical simulation is given in Sect. F.3.6. In Sect. F.3.7, the implementation of GPIC is given. To verify our implementation and test its efficiency, we provide some comparative examples in Sect. F.3.8.

The code organization is shown in Table F.5, where each file (except the input files) defines a module. We also made a git repository for the code, see Listing F.19. This was convenient for code development and we take this opportunity to briefly introduce this powerful tool for code development.

Below is how to get the code `juMP` and run the provided examples:

1. Install `Julia`;
2. Get the source code of `juMP` at https://github.com/vinhphunguyen/jump;
3. Open a terminal and go to the folder that contains the source of `juMP`;
4. Launch `Julia` by typing julia and enter on the terminal;
5. Within the REPL of `Julia`, type include("install_script.jl") to install required packages;
6. Within the REPL of `Julia`, type include("Main_TwoDisks.jl") to run the two disk collision example;
7. A new sub-folder "twodisks-mpm" is created by `juMP` where you can find the output LAMMPS dump files which can be opened using `Ovito` for visualization.

Listing F.19 Some common git commands.

```
# make a local git folder ./git in the folder containing the code
git init
# link it with the github repo
git remote add origin git@github.com:vinhphunguyen/Material-Point-Method-in-Julia.git
# add files
git add *.jl
git commit -m '1st commit'
git push origin master
# work on the code for a while, then
git add *.jl
git commit -m 'your message'
git push origin master
# if you need to go back in time
git checkout <commit hash> <filename> # to revert filename back to commit hash
```

F.3.1 Grid and Particle Data Structure

Limiting to 2D problems, our grid is coded as a composite type `Grid2D` which contains all data for the whole grid (rather than a grid node as in our previous implementation). This type is implemented in module `Grid` (file `Grid.jl`), see Listing F.20. We store grid data as a large array: for example, the grid momentum is a vector (of which length is equal the number of grid points) of a static vector of length 2 i.e., `Vector{SVector{2, Float64}}`.

Listing F.20 Type `Grid2D`.

```
module Grid
   struct Grid2D
      lx       :: Float64            # length in x dir
      ly       :: Float64            # length in y dir
      dx       :: Float64            # cell size in y dir
      dy       :: Float64            # cell size in y dir

      mass     :: Vector{Float64}
      pos      :: Vector{SVector{2,Float64}}
      momentum :: Vector{SVector{2,Float64}}
      force    :: Vector{SVector{2,Float64}}

      # constructor, GL_x is length of the grid in x dir
      # iN_x: number of nodes in x dir
      function Grid2D(fGL_x, fGL_y, iN_x, iN_y)
        dx          = fGL_x / Float64(iN_x - 1.0)
        dy          = fGL_y / Float64(iN_y - 1.0)
        mass        = fill(0,iN_x*iN_y)
        momentum    = fill(zeros(2),iN_x*iN_y)
        force       = fill(zeros(2),iN_x*iN_y)
        pos         = fill(zeros(2),iN_x*iN_y)
        # some code skipped here...
        return new(fGL_x,fGL_y,....)
      end
   end
end
```

We adopt the package `StaticArrays` to use its `SVector`, `SMatrix` and `MMatrix`. Those static arrays are efficient for small arrays with known dimensions than the generic `Array`.

In the same manner, we code a type named `Solid2D` (Listing 21). A solid represents a group of particles. Note that we store a particle stress (and other second order tensors) as a 2×2 matrix, not a 3×1 vector using the Voigt notation. This is to follow `Karamelo`'s implementation and thus code sharing. Also, a solid is attached

to a material. To this end, we coded a type named `MaterialType` to be discussed in Sect. F.3.3.

Listing F.21 Type `Solid2D`.

```
module Solid
  using LinearAlgebra
  using StaticArrays
  using Material

  struct Solid2D
    mass              :: Vector{Float64}
    volumeInitial     :: Vector{Float64}
    volume            :: Vector{Float64}
    pos               :: Vector{SVector{2,Float64}} # position
    velocity          :: Vector{SVector{2,Float64}} # velocity
    strain            :: Vector{SMatrix{2,2,Float64}} # strain, 2x2 matrix
    stress            :: Vector{MMatrix{2,2,Float64}} # stress
    parCount          :: Int64
    mat               :: MaterialType
  end
end
```

To do 1D and 3D simulations, we also code `Solid1D`, `Solid3D` and `Grid1D` and `Grid3D`. It is clear that this design has some code duplication, but we wanted to maximize the speed.

We now want to provide some details on how to find to which nodes a particle $\mathbf{x}_p$ contribute in 3D. Assume that we use a 3D grid as shown in Fig. F.6. We first determine the 3D index (i, j, k) of the lower bottom corner of the cell containing $\mathbf{x}_p$ using

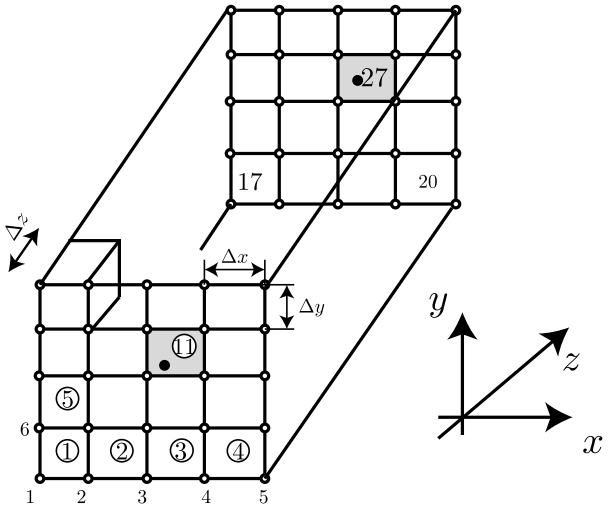


Fig. F.6 A three dimensional structured grid. The nodes are numbered from left to right in the x direction, then from bottom to up in the y direction for the plane $z = z_{\min}$. Then, for the next plane up to $z = z_{\max}$

$$i = \text{int}\left[\text{floor}\left(\frac{x_p}{\Delta_x}\right) + 1.0\right] \tag{F.1}$$

$$j = \text{int}\left[\text{floor}\left(\frac{y_p}{\Delta_y}\right) + 1.0\right] \tag{F.2}$$

$$k = \text{int}\left[\text{floor}\left(\frac{z_p}{\Delta_z}\right) + 1.0\right] \tag{F.3}$$

where in `Julia`, the floor function returns a real, so we needed to convert its output to an integer. Then, we find the 1D index of this point, ii by

$$ii = \texttt{numX} * \texttt{numY} * (k-1) + \texttt{numY} * (j-1) + i \tag{F.4}$$

where `numX/Y` denotes the number of nodes along the x and y direction, respectively. Then, particle p will contribute to the following nodes

$$[ii, ii+1, ii+\texttt{numX}, ii+1+\texttt{numX}, \\ ii+\texttt{numXY}, ii+1+\texttt{numXY}, ii+\texttt{numX}+\texttt{numXY}, ii+\texttt{numX}+1+\texttt{numXY}] \tag{F.5}$$

where `numXY` is short for `numX*numY`. Note that the first two nodes in this equation is for 1D grids, the first four nodes is for 2D grids.

F.3.2 Basis Functions

To cover different basis functions and for different grids (i.e., dimensions), we code a module named `Basis` shown in Listing F.22. We provide different implementations of `getShapeAndGradient`, one for a particular basis e.g. linear or quadratic B-splines and a concrete grid (e.g. 1D or 2D).

Listing F.22 Module `Basis`.

```
1  module Basis
2  using Grid
3  using Solid
4
5  export LinearBasis, CPDIQ4Basis
6  export getShapeAndGradient, getShapeFunctions
7  struct LinearBasis end
8  struct CPDIQ4Basis end
9  function getShapeAndGradient(nearPoints::Vector{Int64}, funcs::Vector{Float64},
10                              ders::Vector{Float64}, p::Int64, grid::Grid1D,
11                              solid,basis::LinearBasis)
12 function getShapeAndGradient(nearPoints::Vector{Int64}, funcs::Vector{Float64},
13                              ders::Vector{Float64}, p::Int64, grid::Grid2D,
14                              solid,basis::LinearBasis)
```

The function `getShapeAndGradient`, for a given particle p, computes: (i) the node indices I to which p contributes, (ii) the basis functions of these nodes ϕ_{Ip} and (iii) their gradients $\nabla\phi_{Ip}$. To reduce memory allocation, the arrays storing these quantities are allocated once for the entire simulation.

F.3.3 Material

To have a MPM code that can adopt different types of materials, we coded an abstract type named `MaterialType` and some sub-types such as `ElasticMaterial` and `ElastoPlasticMaterial`, see Listing F.23. We have made a decision that for inelastic materials, history variables are stored as fields (or members) of a material sub-type, not in the solid particles. This helps to reduce memory storage, but the main reason for our implementation is that it allows a unified interface for the function `update_stress!` for all materials. This has one downside: writing particle data to files is harder as the particles have different data. For example, an elastic solid does not have plastic strain whereas a plastic solid does.

It is clear that our implementation of the `update_stress!` cannot handle all material models. For example, for hyperelastic solids, one needs the gradient deformation tensor $\mathbf{F}$ and its determinant J for the stress update. Implementing a generic interface that can cover all material models is simply beyond our capability.

Listing F.23 Type `MaterialType` and some sub-types for elastic and plastic materials.

```
abstract type MaterialType end

struct ElasticMaterial <: MaterialType
  E      ::Float64
  nu     ::Float64
  density::Float64

  function ElasticMaterial(E,nu,density)
  end
end

function update_stress!(sigma::MMatrix{2,2,Float64},
                        mat::ElasticMaterial,
                        epsilon::SMatrix{2,2,Float64},ip)
  sigma .= mat.lambda*(epsilon[1,1]+epsilon[2,2])*Identity + 2.0 * mat.mu * epsilon
end

struct ElastoPlasticMaterial <: MaterialType
  fy     ::Float64        # yield stress
  k1     ::Float64        # hardening modulus

  pstrain::Vector{MVector{3,Float64}} # plastic strain
  alpha  ::Vector{Float64}          # equivalent plastic strain

  function ElastoPlasticMaterial(E,nu,density,fy,k1,parCount)
  end
end

function update_stress!(sig::MMatrix{2,2,Float64},
                        mat::ElastoPlasticMaterial,
                        eps::SMatrix{2,2,Float64},ip)
end
```

F.3.4 Problem

To handle different problems e.g. explicit dynamics, thermal analysis or thermo-mechanical analyses, a type named `ExplicitSolidMechanics` was coded, see Listing F.24. This type contains a grid and a list of solids and implements a function `solve`.

F.3.5 Graphics

We use PyPlot for real-time graphics in juMP and Ovito for post-processing; we refer to Sect. 5.5 for a discussion on visualization of MPM results. The package PyPlot uses the Julia PyCall package to call Matplotlib directly from Julia with little or no overhead (arrays are passed without making a copy). You will need to have the Python Matplotlib library installed on your machine to use PyPlot.

Listing F.24 Type ExplicitSolidMechanics.

```
struct ExplicitSolidMechanics
  grid     ::Grid2D
  solids   ::Vector{Solid2D}
  basis
  output   ::OutputType
  Tf       ::Float64
  kinEnergy ::Vector{Float64}
  strEnergy ::Vector{Float64}
  recordTime::Vector{Float64}

  function ExplicitSolidMechanics(grid::Grid2D,solids::Vector{Solid2D}, basis,
                                  output::OutputType,Tf::Float64)
  end
end

function solve(problem::ExplicitSolidMechanics,alg::MUSL,dtime)
end
function solve(problem::ExplicitSolidMechanics,alg::USL,dtime)
end
```

To support different graphics, we define an abstract type and two sub-types (one for PyPlot and the other for Ovito) as shown in Listing F.25. There are two functions, plotParticles, specialized for each sub-type. In the constructor of OvitoOutput, we used some Julia features to work with directories and files (Listing F.26). We have used the package Glob to easily get files with a certain extension.

F.3.6 A Complete Example

Having presented all components of juMP, it is now ready to solve a concrete problem. The steps are: (1) creating the grid, (2) choosing a basis, (3) creating some geometries, (4) creating some materials, (5) creating some solids by associating a geometry with a material, (6) assigning initial velocities to the solids (if any), (7) defining the problem, (8) choosing an algorithm (USL or MUSL supported), (9) providing an output e.g. Ovito-based or PyPlot and finally (10) solving that problem. Listing F.27 presents an example for the two disk collision problem.

F.3.7 Implementation of GPIC

This section presents the implementation of GPIC (discussed in Sect. 3.7), which is a version of the MPM that uses a mesh to represent the solids and thus faithfully capture its boundary and GPIC can handle boundary conditions effortlessly. For brevity we do not discuss frictional contact.

A typical simulation set up using GPIC is given in Listing F.28. We will discuss `FEM2D` and `solve_explicit_dynamics_femp_2D`. Check the file FemMPM.jl to see all the supported algorithms. Line 18 in Listing F.28 specifies whether a UL, TL or TL with full quadrature is adopted.

Listing F.25 Graphics supporting `PyPlot` and `Ovito`.

```
module Output
  import PyPlot
  using Solid

  abstract type OutputType end
  struct PyPlotOutput <: OutputType
    interval     ::Int64
    dir          ::String
    function PyPlotOutput(interval::Int64,dir::String)
      if !isdir(dir)  mkdir(dir) end
      new(interval,dir)
    end
  end
  struct OvitoOutput <: OutputType
    interval     ::Int64
    dir          ::String
    function OvitoOutput(interval::Int64,dir::String)
    end
   end
  function plotParticles(plot::PyPlotOutput,solids::Vector{Solid2D},
                         lims::Vector{Float64},ncells::Vector{Int64},
                         counter)
  function plotParticles(plot::OvitoOutput,solids::Vector{Solid2D},
                         lims::Vector{Float64},ncells::Vector{Int64},
                         counter)
end
```

Listing F.26 Working with directories and files.

```
function OvitoOutput(interval::Int64,dir::String,outs)
  if isdir(dir)                                  # if dir exists
    dumpfiles = Glob.glob(string(dir,"*.LAMMPS")) # get all files *.LAMMPS
    if (length(dumpfiles) > 0 )                  # if found, delete them
      [rm(file) for file in dumpfiles]
    end
  else
    mkdir(dir)
  end
  new(interval,dir,outs)
end
# write to dump files to be processed by Ovito
fileName = string(plot.dir,"dump_p.","$(Int(counter)).LAMMPS")
file     = open(fileName, "a")
write(file, "ITEM: TIMESTEP \n")
write(file, "ITEM: NUMBER OF ATOMS\n")
write(file, "$(parCount) \n")
```

Listing F.27 A typical simulation setup in `juMP`.

```
grid    = Grid2D(0.0,1.0, 0.0, 1.0, 41, 41) # 1x1 domain with 40x40 cellss
basis   = LinearBasis()
fOffset = 0.2/16 # there are 8 material points over the radius (16 MPs)
coords1 = buildParticleForCircle([0.2; 0.2], 0.2, fOffset)
```

```
coords2 = buildParticleForCircle([0.8; 0.8], 0.2, fOffset)
material = ElasticMaterial(youngModulus,poissonRatio,density)
solid1  = Solid2D(coords1,material)
solid2  = Solid2D(coords2,material)
assign_velocity(solid1, SVector{2,Float64}([0.1 0.1]))
assign_velocity(solid2,-SVector{2,Float64}([0.1 0.1]))
solids  = [solid1, solid2]
Tf      = 3.5
output  = PyPlotOutput(interval,"results/","Two Disks Collision",(4., 4.))
problem = ExplicitSolidMechanics(grid,solids,basis,output,Tf)
algo    = MUSL(0.99)
solve(problem,algo,0.001)
```

Listing F.28 A typical GPIC simulation setup in `juMP`.

```
grid     = Grid2D(0,1.05, 0,1.05, 21, 21)
basis    = LinearBasis()
material = ElasticMaterial(youngModulus,poissonRatio,density,0,0)
solid1   = FEM2D("disk.msh")
solid2   = FEM2D("disk.msh")

v0       = SVector{2,Float64}([ 0.1 0.1])
Fem.assign_velocity(solid1, v0)
Fem.assign_velocity(solid2,-v0)
# as the mesh of the disks was created with the center of the disk at (0,0)
Fem.move(solid1,SVector{2,Float64}([ 0.2+0.05, 0.2+0.05]))
Fem.move(solid2,SVector{2,Float64}([ 0.2+.6,0.2+.6]))

solids  = [solid1 solid2]
mats    = [material, material]
output2 = VTKOutput(interval,"twodisks-femp/",["pressure"])
fix     = EnergiesFix(solids,"twodisks-femp/energies.txt")
algo1   = TLFEM(0.,1.) # other options: ULFEM(0.), TLFEMFull(1.)
body    = ConstantBodyForce2D([0.,0.])
data            = Dict()
data["total_time"] = Tf
data["dt"]      = dtime
solve_explicit_dynamics_femp_2D(grid,solids, mats, basis,body,algo1,output2,fix,data)
```

FEM2D. To represent a 2D solid a structure named `FEM2D` is coded. The main interface of this structure is given in Listing F.29. It is quite similar to `Solid2D` with a few differences. Similar to FEM, we store the velocity, mass, volume, coordinates, internal/external forces at the nodes and stress/strain at Gauss points. The FE mesh for a solid is stored in the variable `mesh`, which is of type `FEMesh`. We do not discuss `FEMesh`; it is mainly used to store all data of a FE mesh. The code only supports two-node line elements, three-node triangle element, four-node quadrilateral elements, four-node tetrahedral elements and eight-node hexahedral elements.

The constructor of `FEM2D` is given in Listing F.30. Its function is to read a mesh from a file name and intialize all data. Currently the code supports Gmsh and Abaqus mesh files.

Similarly we also coded `FEM3D` for 3D solids and `FEMAxis` for axisymmetric solids (see Fig. 7.9 for an axisymmetric simulation using `juMP`).

Listing F.29 Structure `FEM2D`.

```
struct FEM2D
  mass              :: Vector{Float64}
  volume            :: Vector{Float64}
  centerX           :: Vector{Float64}          # centroids of elements
  pos0              :: Vector{SVector{2,Float64}} # node coords initial
  pos               :: Vector{SVector{2,Float64}} # position
  velocity          :: Vector{SVector{2,Float64}} # velocity
  dU                :: Vector{SVector{2,Float64}} # incremental displacements
  fint              :: Vector{SVector{2,Float64}} # internal forces at FE nodes
  ftrac             :: Vector{SVector{2,Float64}} # external forces
  deformationGradient :: Vector{SMatrix{2,2,Float64,4}} # F, 2x2 matrix
```

```
    strain            :: Vector{SMatrix{2,2,Float64,4}} # stress, 2x2 matrix
    stress            :: Vector{SMatrix{2,2,Float64,4}} # strain
    parCount          :: Int64            # same as element count in the mesh
    nodeCount         :: Int64            # number of nodes in the mesh
    elems             :: Array{Int64,2}
    mesh              :: FEMesh
    basis             :: FiniteElement   # finite element basis function
end
```

Solution phase. The function `solve_explicit_dynamics_femp_2D` implements the GPIC algorithm described in Algorithm 5. For every time step, the solution phase consists of 4 steps:

- **M2G**: Listing F.31
- **Updating grid**: this is identical to MPM;
- **G2M1**: update the position/velocity of the solid nodes; similar to MPM
- **G2M2**: update stress at GPs and internal forces at solid nodes, see Listing F.32.

Listing F.30 Constructor of `FEM2D`.

```
function FEM2D(fileName)
  mesh      = read_GMSH(fileName)
  nodeCount = length(mesh.nodes)                  # node count
  parCount  = length(mesh.element_sets["All"])    # element count
  Identity = SMatrix{2,2}(1, 0, 0, 1)
  F         = fill(Identity, parCount)
  strain    = fill(zeros(2,2),parCount)
  x         = fill(zeros(2),nodeCount)
  elems     = Array{Array{Int64,1},1}(undef,0) # element nodes
  velo      = fill(zeros(2),nodeCount)
end
```

Listing F.31 Solution in GPIC: M2G phase. Note that there is no need for the gradient of the grid weighting functions. Instead we need the shape functions and its gradient for the solid mesh.

```
for s = 1:solidCount
  solid = solids[s]
  xx    = solid.pos
  mm    = solid.mass
  vv    = solid.velocity
  fint  = solid.fint
  for ip = 1:solid.nodeCount
    support  = getShapeFunctions(nearPoints,funcs,ip, grid, solid,basis)
    fMass    = mm[ip]
    vp       = vv[ip]
    fp       = fint[ip]
    for i = 1:support
      id               = nearPoints[i]; # index of node 'i'
      Ni               = funcs[i]
      Nim              = Ni * fMass
      nodalForce[id]   -= Ni * fp
    end
  end
end
```

F.3.8 Performance Tests

To test the performance of `juMP` compared with the old `Julia` code, we consider two examples: the two disk collision and the high velocity impact.

Table F.6 Performance comparison of two `Julia` codes for the two-disk impact problem. No output or real-time visualization. Simulations carried out with a MacBook Pro 2018 with 3.1 GHz Quad-Core Intel Core i7 and 16 GB of RAM

Particles	Grid	Total time [s]		
		Old `Julia`	New `Julia`	Ratio
416	20 × 20	69.48	4.46	16
1624	20 × 20	270.99	17.34	15
1624	40 × 40	271.33	18.22	15
25 784	80 × 80	4689.47	327.46	14

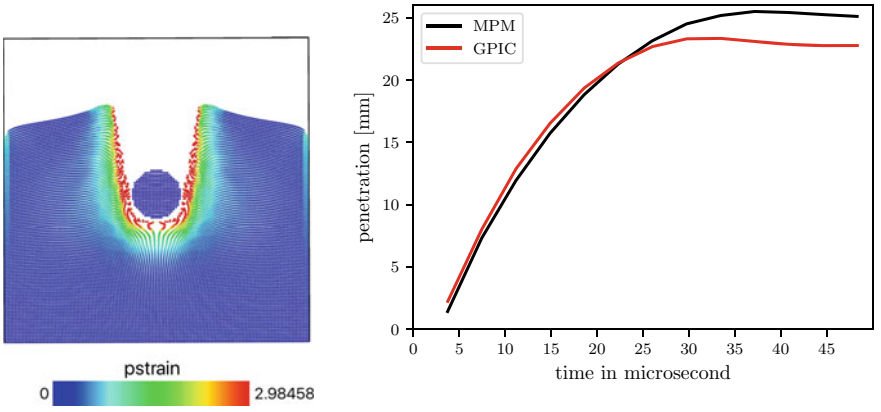
Collision of two elastic disks. We consider again this popular problem, and the performance of the two MPM codes is given in Table F.6. The new one is consistently 15× faster than the old one, and thus it is about 75 × faster than the `Matlab` code.[2] This is not surprising as the performance of large arrays is much better than the use of arrays of composite types (Sect. F.1.3).

Listing F.32 Solution in GPIC: stress update phase.

```
for s = 1:solidCount
  solid = solids[s]
  XX    = solid.pos0; du   = solid.dU
  stress = solid.stress; elems = solid.elems; fint = solid.fint
  for ip = 1:solid.parCount
    elemNodes = @view elems[ip,:]
    coords   =  @view XX[elemNodes]
    vel_grad =  SMatrix{2,2}(0., 0., 0., 0.)
    for gp = 1:noGP
      xieta = @view gpCoords[:,gp]
      detJ     = lagrange_basis_derivatives!(N, dNdx, meshBasis, xieta, coords)
      w        = detJ * weights[gp]
      for i = 1:length(elemNodes)
        in        = elemNodes[i]; # index of node 'i'
        dNi       = @view dNdx[:,i]
        vI        = du[in]
        vel_grad += SMatrix{2,2}(dNi[1]*vI[1], dNi[1]*vI[2],dNi[2]*vI[1], dNi[2]*vI[2])
      end
      D           = 0.5 * (vel_grad + vel_grad')
      F1          = Identity + vel_grad
      J           = det(F1)
      sigma = ...
      P     = J*sigma*inv(F1)' # convert to 1st Piola Kirchoof stress
      for i = 1:length(elemNodes)
        in  = elemNodes[i]; # index of node 'i'
        dNi = @view dNdx[:,i]
        fint[in]    += w * @SVector[P[1,1] * dNi[1] + P[1,2] * dNi[2],
                                    P[2,1] * dNi[1] + P[2,2] * dNi[2]]
      end
    end
  end
end
```

High velocity impact. We consider again the high velocity impact presented in Sect. 6.15.3. The aims are: (1) to demonstrate the flexibility of `juMP` in treating different materials in a simulation and in setting an output type (`PyPlot` or `Ovito`)

[2] This comparison should not be taken seriously as we spent time to optimize the `Julia` code but did not do the same thing with the `Matlab` code. This is because we though that we should use open source and free tools.



(a) Plastic strain (b) Penetration

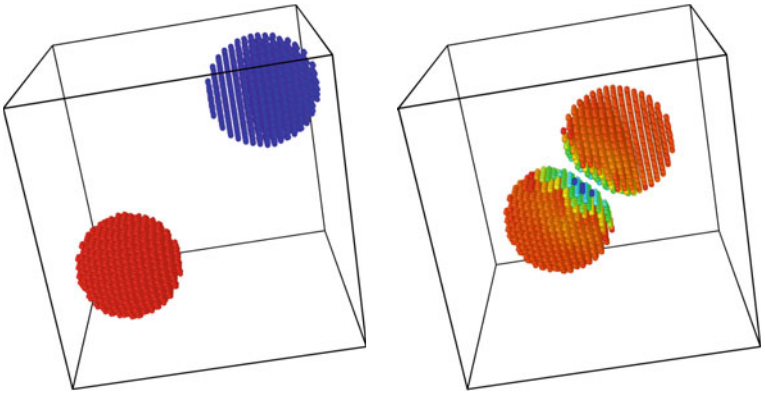
Fig. F.7 High velocity impact problem: plot of the equivalent plastic strain at the end of the simulation in `Ovito` (**a**) and evolution of the penetration (**b**)

and (2) to compare the efficiency of this code with `Karamelo`. The results given in Fig. F.7 verified the implementation. However, our `Julia` code is still less efficient compared to `Karamelo`. We present in Sect. F.4 modifications to increase the efficiency of `juMP`.

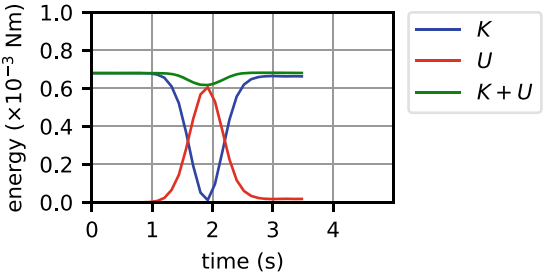
Collision of two elastic spheres. Since `Julia` is quite fast it is possible to do 3D simulations. As a simplest test for 3D, we consider again the popular problem of two disk collision but now extended to 3D. Two spheres with centers at $(0, 2, 0.2, 0.5)$ and $(0.8, 0.8, 0.5)$ are put in a grid of $1 \times 1 \times 1$ size (Fig. F.8a). As the thickness does not play any role, only one layer of cells can be used in the z direction. The initial velocities are $(0.1, 0.1, 0.0)$ and $(-0.1, -0.1, 0.0)$ for the left and right spheres, respectively. The solution must be similar to the 2D problem, except the magnitude of the kinetic and strain energies (Fig. F.8).

F.4 Tweaks for Speed

A careful examination of the previous code using the macro `@code_warntype` reveals that our type `Solid2D` is not concrete i.e., it contains `mat::Material Type` which is an abstract type. And this decreases the efficiency significantly. To avoid this issue, we implemented `Solid2D` as a parameterized type with the parameter being the type of the material. Listing F.33 is the new implementation, which is only a slight modification to the previous one.



(a) Initial setup (b) At collision



(c) Energies in time

Fig. F.8 Collision of two elastic spheres: USL was used with cut-off tolerance of 10^{-9}

Listing F.33 The new `Solid2D`, which is a parameterized type.

```
struct Solid2D{T <: MaterialType}
 deformationGradient :: Vector{SMatrix{2,2,Float64,4}} # F, 2x2 matrix
 strain              :: Vector{SMatrix{2,2,Float64,4}} # strain, 2x2 matrix
 stress              :: Vector{MMatrix{2,2,Float64,4}} # stress
 mat                 :: T
 # inner constructor
 function Solid2D(pts::Vector{SVector{2,Float64}},mat::T) where {T <: MaterialType}
  return new{T}(...)
 end
end
```

Any function working on `Vector{Solid2D}` or `Vector{Solid3D}` needs modification as these types are now parameterized and thus the correct types are `Vector{Solid2D{T}}` and `Vector{Solid3D{T}}`. The modified function is shown in Listing F.34.

Similarly, the type `Problem` is a not a concrete type. So, we no longer use this type, and instead directly implement different `solve()` functions. See Listing F.35 for one of them. To get maximum efficiency, we pre-allocate arrays used in the calculation such as ones to store the basis functions, the derivatives, the strain rate

matrix. And for matrix-matrix multiplication, we explicitly code the operation (lines 23 and 41). Doing so removed allocation for temporary arrays. Finally, we added the macro `@inbounds` to bypass array index bound checking. Macro @view is used to avoid copying when a slice of an array is needed. $L_{ij} = v_{i,j}$ was explicitly coded to avoid temporary arrays.

Listing F.34 Functions with `Vector{Solid2D{T}}`.

```
function plotParticles(plot::OvitoOutput,solids::Vector{Solid2D{T}},
  lims::Vector{Float64},ncells::Vector{Int64},counter::Int64) where {T<:MaterialType}
  // exactly as before
end
function plotParticles(plot::OvitoOutput,solids::Vector{Solid3D{T}},
  lims::Vector{Float64},ncells::Vector{Int64},counter::Int64) where {T<:MaterialType}
  // exactly as before
end
```

Git. It is always a good idea not to modify a working piece of code to experiment something else. So, we made a new branch, work on that branch for the modifications presented in this section. The git commands for this are given in Listing F.36.

Listing F.35 USL algorithm implemented in `juMP`.

```
function solve_explicit_dynamics_2D(grid,solids,basis,alg::MUSL,output,fixes,Tf,dtime)
t            = 0.               # not t = 0 => type instability issue
Identity     = UniformScaling(1.) # identity matrix
nearPoints,funcs, ders = initialise(grid,basis)
D                  = SMatrix{2,2}(0., 0., 0., 0.) #zeros(Float64,2,2)
while t < Tf
  @inbounds for i = 1:grid.nodeCount
    nodalMass[i]     = 0.
    nodalMomentum0[i] = @SVector[0., 0.]
  end
  # P2G step
  for s = 1:solidCount
   solid = solids[s]; xx   = solid.pos
   @inbounds for ip = 1:solid.parCount
   support  = getShapeAndGradient(nearPoints,funcs,ders,ip, grid, solid,basis)
   volp     = vol[ip]
   sigma    = stress[ip]
   @inbounds for i = 1:support
    id             = nearPoints[i]
    dNi            = @view ders[:,i]
    nodalForce[id] -= volp*@SVector[sigma[1,1] * dNi[1] + sigma[1,2] * dNi[2],
                                    sigma[2,1] * dNi[1] + sigma[2,2] * dNi[2]]
   end
   end
  end
  # G2P step
  @inbounds for s = 1:solidCount
    solid = solids[s]; xx  = solid.pos
    @inbounds for ip = 1:solid.parCount
      support = getShapeAndGradient(nearPoints,funcs,ders,ip,grid,solid,basis)
      vel_grad = SMatrix{2,2}(0., 0., 0., 0.)
      for i = 1:support
        dNi= @view ders[:,i]
        m  = nodalMass[id]
        if ( m > 0.)
            vI         = nodalMomentum2[id] /m
            xx[ip]    += (Ni * nodalMomentum[id]/m) * dtime
            vel_grad += SMatrix{2,2}(dNi[1]*vI[1], dNi[2]*vI[1],
                                     dNi[1]*vI[2], dNi[2]*vI[2])
        end
      end
      D          = 0.5 * (vel_grad + vel_grad')
      strain[ip] += dtime * D
      F[ip]      *= (Identity + vel_grad*dtime)
      J           = det(F[ip])
    end
end
end
```

Listing F.36 Making a new git branch and related commands.

```
git branch parameterized-solid    # make a new branch
git checkout parameterized-solid # move to that branch
# edit the code, e.g. Solid.jl
git add Solid.jl
git commit -m 'Solid'
git push origin parameterized-solid
git checkout master              # to work on the original code
```

References

Belytschko, T., Liu, W.K., Moran, B.: Nonlinear Finite Elements for Continua and Structures. Wiley, Chichester, England (2000)

Bezanson, J., Edelman, A., S. K., Shah, V.B.: Julia: a fresh approach to numerical computing. CoRR, abs (2014). arXiv:1411.1607

Bezanson, J., S.K., Shah, V.B., Edelman, A.: Julia: a fast dynamic language for technical computing. CoRR, abs (2012). arXiv:1209.5145

Felippa, C.A.: The linear tetrahedron. http://www.colorado.edu/engineering/CAS/courses.d/AFEM.d/AFEM.Ch09.d/AFEM.Ch09.pdf

Hughes, T.J.R.: The Finite Element Method - Linear Static and Dynamic Finite Element Analysis. Prentice-Hall, London, England (1987)

Johnson, S.G.: PyPlot module for Julia (2012). https://github.com/stevengj/PyPlot.jl

Lattner, C., Vikram A.: LLVM: a compilation framework for lifelong program analysis & transformation. In: Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization, CGO '04, pp. 75–. IEEE Computer Society (2004)

Sadeghirad, A., Brannon, R.M., Burghardt, J.: A convicted particle domain interpolation technique to extend applicability of the material point method for problems involving massive deformations. Int. J. Numer. Meth. Eng. **86**(12), 1435–1456 (2011)

Sinaie, S., Nguyen, V.P., Nguyen, C.T., Bordas, S.: Programming the material point method in Julia. Adv. Eng. Softw. **105**, 17–29 (2017)

Vigliotti, A., Auricchio, F.: Automatic differentiation for solid mechanics. Arch. Comput. Methods Eng. **28**(3), 875–895 (2021)

Wu, S.R., Wu, L.: Introduction to the Explicit Finite Element Method for Nonlinear Transient Dynamics. Wiley (2012)

Xiao, L., Mei, G., Xi, N., Piccialli, F.: Julia language in computational mechanics: a new competitor. In: Archives of Computational Methods in Engineering, pp. 1–14 (2021)

Index

N. V. Phu et al., *The Material Point Method*, Scientific Computation,
https://doi.org/10.1007/978-3-031-24070-6