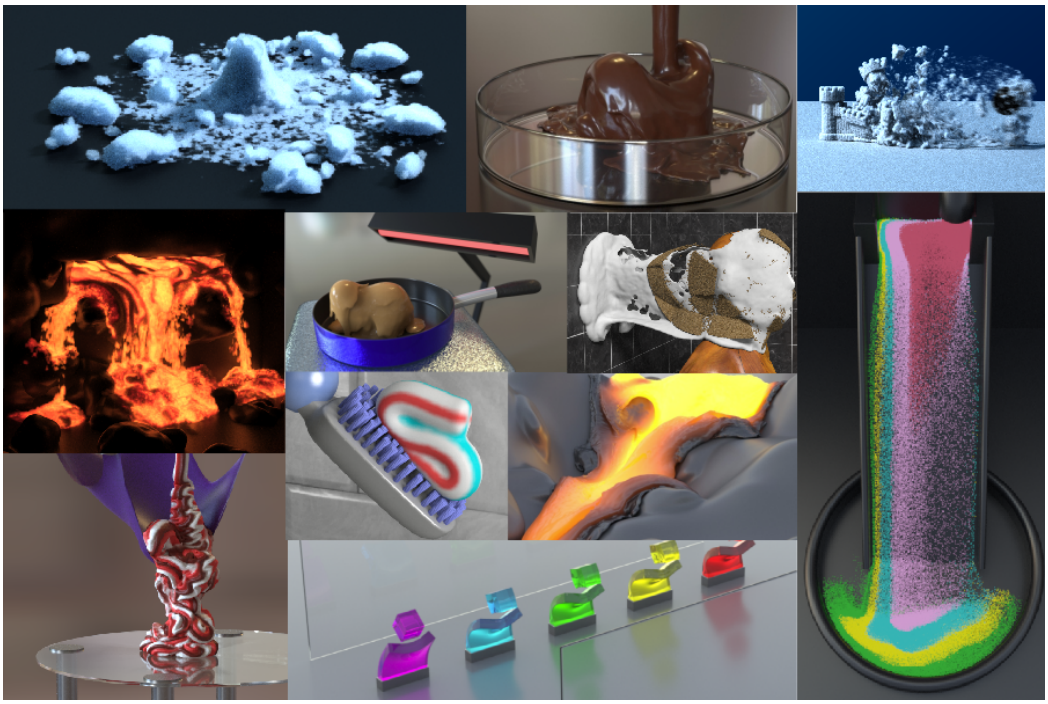
The Material Point Method for Simulating Continuum Materials

Chenfanfu Jiang*[1], Craig Schroeder†[2], Joseph Teran‡[1,3], Alexey Stomakhin§[3], and Andrew Selle¶[3]

[1]Department of Mathematics, University of California, Los Angeles
[2]Department of Computer Science, University of California, Riverside
[3]Walt Disney Animation Studios

SIGGRAPH 2016 Course Notes Version 1 (May 2016)



* cffjiang@math.ucla.edu
† snubdodecahedron@gmail.com
‡ jteran@math.ucla.edu
§ alexey.stomakhin@disneyanimation.com
¶ andrew.selle@disneyanimation.com

SIGGRAPH '16 Courses, July 24-28, 2016, Anaheim, CA,
ACM 978-1-4503-4289-6/16/07.
http://dx.doi.org/10.1145/2897826.2927348

ABSTRACT

Simulating the physical behaviors of deformable objects and fluids has been an important topic in computer graphics. While the Lagrangian Finite Element Method (FEM) is widely used for elasto-plastic solids, it usually requires additional computational components in the case of large deformation, mesh distortion, fracture, self-collision and coupling between materials. Often, special solvers and strategies need to be developed for a particular problem. Recently, the hybrid Eulerian/Lagrangian Material Point Method (MPM) was introduced to the graphics community. It uses a continuum description of the governing equations and utilizes user-controllable elasto-plastic constitutive models. The hybrid nature of MPM allows using a regular Cartesian grid to automate treatment of self-collision and fracture. Like other particle methods such as Smoothed Particle Hydrodynamics (SPH), topology change is easy due to the lack of explicit connectivity between Lagrangian particles. Furthermore, MPM allows a grid-based implicit integration scheme that has conditioning independent of the number of Lagrangian particles. MPM also provides a unified particle simulation framework similar to Position Based Dynamics (PBD) for easy coupling of different materials. The power of MPM has been demonstrated in a number of recent papers for simulating various materials including elastic objects, snow, lava, sand and viscoelastic fluids. It is also highly integrated into the production framework of Walt Disney Animation Studios and has been used in featured animations including *Frozen*, *Big Hero 6* and *Zootopia*.

CONTENTS

1 ABOUT THE LECTURERS

Chenfanfu Jiang
Mathematics Department
University of California, Los Angeles
cffjiang@math.ucla.edu

Chenfanfu Jiang received his Ph.D. in Computer Science at UCLA in 2015, awarded UCLA Engineering School Edward K. Rice Outstanding Doctoral Student. He is currently a postdoctoral researcher at UCLA, jointly appointed to the departments of Mathematics and Computer Science. His primary research interests include solid/fluid mechanics, physics based simulation and their applications to medical simulation and scene understanding. He actively collaborates with Walt Disney Animation Studios and Center for Advanced Surgical and Interventional Technology (CASIT).

Craig Schroeder
Computer Science Department
University of California, Riverside
snubdodecahedron@gmail.com

Craig Schroeder is currently an assistant professor in computer science at University of California Riverside. He received his Ph.D. in computer science from Stanford University in 2011, followed by a postdoc at University of California Los Angeles, where he received the Chancellor's Award for Postdoctoral Research in 2013, recognizing research impact and value to the UCLA community. He actively publishes in both computer graphics and computational physics. His primary areas of interest are solid mechanics and computational fluid dynamics and their applications to physically based animation for computer graphics. He began collaborating with Pixar Animation Studios during his Ph.D. and later collaborated with Walt Disney Animation Studios during his postdoctoral studies. For his research contributions he received screen credits in Pixar's "Up" and Disney's "Frozen."

Joseph Teran
Mathematics Department
University of California, Los Angeles
jteran@math.ucla.edu

Joseph Teran is a professor of applied mathematics at UCLA. His research focuses on numerical methods for partial differential equations in classical physics, including computational solids and fluids, multi-material interactions, fracture dynamics and computational biomechanics. He also works with Walt Disney Animation applying scientific computing techniques to simulate the dynamics of virtual materials like skin/soft tissue, water, smoke and recently, snow for the movie "Frozen". Teran received a 2011 Presidential Early Career Award for Scientists and Engineers (PECASE) and a 2010 Young Investigator award from the Office of Naval Research.

Alexey Stomakhin
Walt Disney Animation Studios
alexey.stomakhin@disneyanimation.com

Alexey Stomakhin is a Senior Software Engineer at Walt Disney Animation Studios. He is responsible for developing tools for simulation of environmental effects. He is the lead developer of the Disney in-house Material Point Method engine (a.k.a. Matterhorn) which was used extensively for snow simulation in Frozen (2013), and also in Big Hero 6 (2014) and Zootopia (2016). He also does research and works extensively on the simulation of fluids, multi-material interactions and parallel/distributed computing. He holds a Ph.D. degree in Mathematics from University of California, Los Angeles (2013).

Andrew Selle
Walt Disney Animation Studios
andrew.selle@disneyanimation.com

Andrew Selle, Principal Software Engineer, is responsible for developing tools and techniques for simulation and rendering at Walt Disney Animation Studios. He focused on fluid simulation techniques in "Tangled", rigid bodies and volumetric rendering on "Wreck-It-Ralph" and Snow simulation and rendering on "Frozen." He also was a major developer on Disney's Hyperion Renderer used in Big Hero 6, Feast, Zootopia, and all upcoming Disney Animation Films. As part of his work he has contributed to the open source community by releasing and maintaining the SeExpr and Partio libraries. Besides development, he remains active in research oriented publication, continuing to publish articles in refereed journals and conferences. Prior to his current position, he was a Research and Development Software Engineer at Industrial Light + Magic. He holds a B.S. in Mathematics in Computer Science from the University of Wisconsin Madison and a M.S. and Ph.D. in Computer Science from Stanford University.

2 SYLLABUS

2.1 Intended Audience

These notes are intended for industry professionals and academic researchers interested in recent advances in the Material Point Method for simulating various materials for computer animation and visual effects.

2.2 Prerequisites

This course requires minimal concepts of continuum mechanics. Familiarity with multivariable calculus, linear algebra and common numerical algorithms is assumed. No experience of MPM is required. Some previous knowledge and experience with the Finite Element Method (FEM) and continuum mechanics would benefit.

2.3 Level of Difficulty

Easy/Intermediate.

2.4 Tentative Schedule

1. Introduction and Welcome (*All speakers*) (5 min)
    - Introduction of Speakers
    - Course overview
    - MPM introduction: advantages and limitations
    - Research demos, Disney production clips
2. MPM in Disney (*Alexey Stomakhin, Andrew Selle*) (35 min)
3. Continuum Mechanics Concepts (*Joseph Teran*) (15 min)
    - Continuum description of material motion
    - Kinematics, deformation gradient, strain
    - Stress and hyperelasticity
    - Governing equations, conservation of mass/momentum
4. MPM Algorithm (Explicit integration) (*Chenfanfu Jiang*) (15 min)
    - Particle-Grid transfers

- Deformation gradient update
- Force computations
- Symplectic Euler time integration
- The full MPM algorithm

5. MPM Algorithm (Implicit integration) (*Craig Schroeder*) (15 min)
    - Force derivative computations
    - Backward Euler time integration
    - Force linearization
    - Newton's method for MPM
6. Conclusion (*Joseph Teran*) (5 min)
    - Difficulties and workarounds
    - Open problems, interesting research directions
    - Conclusion, Q&A

3 INTRODUCTION

Simulating natural phenomena for virtual worlds and characters is an important application that remains extremely challenging. An artist's need to manipulate and comprehend physical simulations imposes a significant constraint, all but requiring simulation methods to involve Lagrangian particles. In addition, the need for computational efficiency, topology change and numerical stability has led engineers toward hybrid Lagrangian/Eulerian methods. In this course, we focus on the Material Point Method (MPM), which rises as the generalization of Particle In Cell (PIC) and Fluid Implicit Particle Method (FLIP) to solid mechanics [Sulsky et al., 1995]. MPM methods combine Lagrangian material particles (points) with Eulerian Cartesian grids. Notably, there is no inherent need for Lagrangian mesh connectivity.

Many researchers in graphics have experimented with hybrid grid and particle methods. While FLIP has been known in graphics community as a useful liquid simulation method for a while [Zhu and Bridson, 2005; Bridson, 2008; Ando and Tsuruno, 2011], MPM is only introduced and studied recently.

MPM has been shown to be a very effective hybrid particle/grid method for simulating various solid materials in computer graphics. Stomakhin et al. [2013] and Disney's *Frozen* use MPM to simulate snow. Hegemann et al. [2013] uses impulses derived from MPM to resolve colliding embedded deformable object pieces. Stomakhin et al. [2014] augments MPM for simulating incompressible materials and melting/freezing. Ram et al. [2015] and Yue et al. [2015] show that MPM is also suitable for complex fluids. Gast et al. [2015] presents an optimization based integrator to accelerate the nonlinear MPM solver. Jiang et al. [2015]; Jiang [2015]; Jiang et al. [2016] propose a stable and angular momentum conserving scheme to improve the particle/grid transfers in MPM. Klar et al. [2016] and Daviet and Bertails-Descoubes [2016] use MPM to discretize the engineering favored Drucker-Prager elastoplastity to simulate sand dynamics.

As with PIC/FLIP, MPM implicitly handles self-collision and fracture with the use of the background Eulerian grid. As a hybrid Lagrangian/Eulerian approach, MPM has the following advantages when compared with traditional Lagrangian methods (such as FEM solids) and Eulerian methods (such as grid-based fluids):

- Just like Lagrangian FEM, MPM can be derived from the weak form of conservation of momentum, allowing for physically accurate discretization of physical laws.
- Boundary conditions, solid wall collisions and external forces can be easily applied on the grid and particles.
- Automatic self collision/contact due to the fact that particle movements are interpolated from undistortable nodal movement on a grid.
- Automatic splitting and merging behaviors because of particle based material representation. This is remarkably useful for fluids and granular materials.

- Automatic multi-material and multiphase coupling can be easily done solely by giving particles different material properties or constitutive models.
- **MPM can also be used to simulate mesh-based Lagrangian forces without loosing its advantages.** (See Section 12.2.) This provides the opportunity of coupling point based objects and mesh based objects with a single solve on the grid that handles collisions automatically.

More rigorously speaking, MPM is a Lagrangian method, but with an Eulerian grid used for computing derivatives. This alleviates the need for a Lagrangian mesh (for derivative computation) that would get tangled when the material is highly deformed from its original configuration. This lets you simulate a wider class of materials than with a purely Lagrangian method. The Eulerian aspects allow for natural treatment of topological changes and collisions (self and with external objects). There is a sacrifice of some accuracy in doing this though and materials like e.g. hyperelasticity are not going to be simulated as effectively. On the other hand, you get self collisions and topology changes for free.

4 MPM IN PRODUCTION

MPM is adopted in Walt Disney Animation Studios for simulating various materials including most of the dynamic snow in *Frozen*. It is also highly integrated into the production framework of *Big Hero 6* and *Zootopia*.

5 KINEMATICS THEORY

The contents in this section mostly follow [Bonet and Wood, 2008]. We summarize the important concepts that are helpful for understanding the essence of MPM.

First and foremost, MPM particles are not individual particles, molecules, atoms or little spheres as one may naturally think when seeing a "particle" method. Each MPM particle actually represents a continuous piece of material, or really a subset of the whole simulated material domain. For those familiar with FEM style weak forms of equations, material points can be thought of as quadrature points for the discretization of spatial stress derivatives (we will talk about the discretization in Section 9).

Such a view is very common in computational mechanics. When we talk about continuum bodies or continuum mechanics, we have adopted the **continuum assumption**. This means the studied material (either solid, liquid or gas) is treated as continuous pieces of matter. Such a view is very practical in engineering and graphics applications (as well as in everyday life) where it is really not necessary to deal with the microscopic inter-

actions between molecules and atoms. Note that a continuum assumption can be made for almost all solids and fluids that are extensively simulated for graphics, including deformable (elastic and plastic) objects, muscle, flesh, cloth, hair, liquids, smoke, gas and granular materials (sand, snow, mud, soil etc.). A continuum body defines quantities such as density, velocity, and force as continuous functions of position. Equations of motion are solved in the spatial domain, and evolved in time to simulate the behaviors of the simulated materials.

5.1 Continuum Motion

Kinematics refers to the study of motion occurred in continuum materials. The main focus is the change of shape, or the deformation, either locally or globally in different coordinate systems of interest. Describing the motion qualitatively and quantitatively is very essential for deriving governing equations of dynamics and mechanical responses. Luckily in most cases, we can describe kinematics without introducing the meaning of force, stress or even mass.

In continuum mechanics, the deformation is usually represented with the material (or undeformed) space $\mathbf{X}$, the world (or deformed) space $\mathbf{x}$ and a deformation map $\phi(\mathbf{X}, t)$. You can simply treat $\mathbf{X}$ as the "initial position" and $\mathbf{x}$ as the "current position" for any point in the simulated material. In particular, at time $t = 0$, $\mathbf{X}$ and $\mathbf{x}$ have the same value.

Here is a more detailed definition. We consider the motion of material to be determined by a mapping $\phi(\cdot, t) : \Omega^0 \to \Omega^t$ for $\Omega^0, \Omega^t \subset \mathbb{R}^d$ where $d = 2$ or 3 is the dimension of the simulated problem (or domain). The mapping ϕ is sometimes called the flow map or the deformation map. Points in the set Ω^0 are referred to as material points and are denoted as $\mathbf{X}$. Points in Ω^t represent the location of material points at time t. They are referred to as $\mathbf{x}$. In other words, ϕ describes the motion of each material point $\mathbf{X} \in \Omega^0$ over time

$$\mathbf{x} = \mathbf{x}(\mathbf{X}, t) = \phi(\mathbf{X}, t). \tag{1}$$

For example, if our object is moving with a constant speed v along direction $\mathbf{n}$, then we have

$$\mathbf{x} = \mathbf{X} + tv\mathbf{n}. \tag{2}$$

If an object went through some rigid motion after time t (compared to time 0), we will have

$$\mathbf{x} = \mathbf{R}\mathbf{X} + \mathbf{b}, \tag{3}$$

where $\mathbf{R}$ is a rotation matrix, $\mathbf{b}$ is some translation. $\mathbf{R}$ and $\mathbf{b}$ will probably be some function with respect to time t and initial position $\mathbf{X}$, depending on the actual motion.

This mapping can be used to quantify the relevant continuum based physics. For example, the velocity of a given material point $\mathbf{X}$ at time t is

$$\mathbf{V}(\mathbf{X}, t) = \frac{\partial \phi}{\partial t}(\mathbf{X}, t) \tag{4}$$

also the acceleration is

$$\mathbf{A}(\mathbf{X}, t) = \frac{\partial^2 \phi}{\partial t^2}(\mathbf{X}, t) = \frac{\partial \mathbf{V}}{\partial t}(\mathbf{X}, t). \tag{5}$$

I.e. $\mathbf{V}(\cdot, t) : \Omega^0 \to \mathbb{R}^d$ and $\mathbf{A}(\cdot, t) : \Omega^0 \to \mathbb{R}^d$.

The velocity $\mathbf{V}$ and acceleration $\mathbf{A}$ defined above are based on the "Lagrangian view", where they are functions of the material configuration $\mathbf{X}$ and time t. Physically, this means we are measuring them on a fixed particle. This particle has its mass and occupies some volume since the beginning. This is an important concept, because soon we will encounter the "Eulerian view", where we are sitting at a fixed position in the space and measuring the velocity of whichever particle that is passing by that position. For example, the flow velocity in a grid based fluid simulation is a typical Eulerian viewed quantity. For solid simulation and continuum mechanics, it is often more natural (but not necessary) to start from the Lagrangian view for deriving stuff.

5.2 Deformation

Now we have $\mathbf{X}$ and $\mathbf{x}$ being material coordinates and world coordinates, and they belong to domain Ω_0 and Ω_t respectively. For any point $\mathbf{X}$ in Ω_0, we also have Φ to map it to Ω_t for a given time t via $\mathbf{x} = \Phi(\mathbf{X}, t)$.

The Jacobian of the deformation map ϕ is useful for a number of reasons described below. E.g. the physics of elasticity is naturally described in terms of this Jacobian. It is standard notation to use $\mathbf{F}$ to refer to the Jacobian of the deformation mapping

$$\mathbf{F}(\mathbf{X}, t) = \frac{\partial \phi}{\partial \mathbf{X}}(\mathbf{X}, t) = \frac{\partial \mathbf{x}}{\partial \mathbf{X}}(\mathbf{X}, t). \tag{6}$$

$\mathbf{F}$ is often simply called the deformation gradient. Discretely it is often a small 2×2 or 3×3 matrix. One special case is for a cloth/thin shell in 3D, $\mathbf{F}$ is 3×2 because the material space is really just 2D. It can be thought of as $\mathbf{F}(\cdot, t) : \Omega^0 \to \mathbb{R}^{d \times d}$. In other words, for every material point $\mathbf{X}$, $\mathbf{F}(\mathbf{X}, t)$ is the $\mathbb{R}^{d \times d}$ matrix describing the deformation Jacobian of the material at time t. We can also use the index notation

$$F_{ij} = \frac{\partial \phi_i}{\partial X_j} = \frac{\partial x_i}{\partial X_j}, \quad i, j = 1, \ldots, d. \tag{7}$$

Now we can compute the deformation gradient of the deformation map in Equation 2. The result is the identity matrix. For the one in Equation 3 we get $\mathbf{F} = \mathbf{R}$. In both

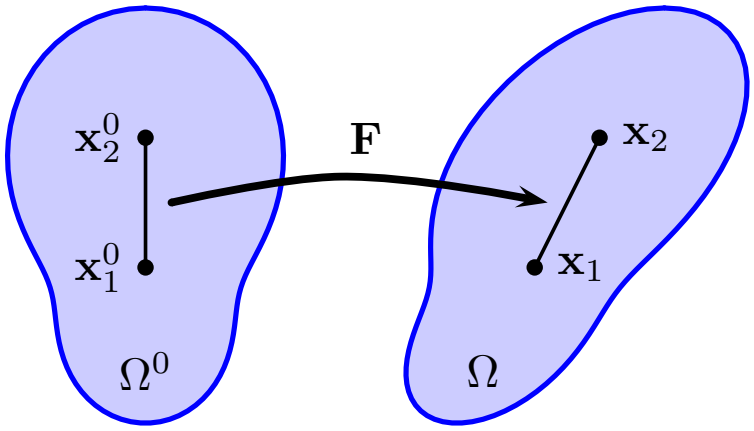


Figure 1: Deformation gradient.

these cases, we know the object does not really deform because all they did are rigid transformations. Such deformation gradients should not result in any internal forces in the material unless artistic effects are desired.

Intuitively, the deformation gradient represents how deformed a material is locally. For example, let $\mathbf{x}_1^0$ and $\mathbf{x}_2^0$ be two nearby points embedded in the material (see Figure 1) at the beginning of the simulation, and let $\mathbf{x}_1$ and $\mathbf{x}_2$ be the same two points in the current configuration. Then $(\mathbf{x}_2 - \mathbf{x}_1) = \mathbf{F}(\mathbf{x}_2^0 - \mathbf{x}_1^0)$.

The determinant of $\mathbf{F}$ (commonly denoted with J) is also often useful because it characterizes infinitesimal volume change. It is commonly denoted with $J = \det(\mathbf{F})$. J is the ratio of the infinitesimal volume of material in configuration Ω^t to the original volume in Ω^0. As an example, it is easy to see for rigid motions (rotations and translations), $\mathbf{F}$ is a rotation matrix and $J = 1$. Note that identity matrix is also a rotation matrix. $J > 1$ means volume increase and $J < 1$ means decrease.

$J = 0$ means the volume becomes zero. In the real world, this will never happen. However, numerically it is possible to achieve such an $\mathbf{F}$. In 3D, that suggests the material is so compressed that it becomes a plane or a line or a single volumeless point. $J < 0$ means the material is inverted. Consider a triangle in 2D, $J < 0$ means one vertex passes through its opposing edge, and the area of this triangle becomes negative. Invertible elasticity [Irving et al., 2004; Stomakhin et al., 2012] is one of the popular methods for resolving these cases. The model we will talk about in Section 6 for snow will be fine with these degenerate cases due to its nice numerical properties.

5.3 Push Forward and Pull Back

So far we have assumed quantities are in terms of $(\mathbf{X}, t)$, this is called the Lagrangian view. The mapping ϕ is assumed to be bijective. And since we will assume it is smooth,

this means that the sets Ω^0 and Ω^t are homeomorphic/diffeomorphic under ϕ. This is associated with the assumption that no two different particles of material ever occupy the same space at the same time. This means that $\forall \mathbf{x} \in \Omega^t$, $\exists! \mathbf{X} \in \Omega^0$ such that $\phi(\mathbf{X}, t) = \mathbf{x}$. In other words, there exist an inverse mapping $\phi^{-1}(\cdot, t) : \Omega^t \to \Omega^0$. This means that any function over one set can naturally be thought of as a function over the other set by means of change of variables. We denote this interchange of independent variable as either push forward (taking a function defined over Ω^0 and defining a counterpart over Ω^t) or vice versa (pull back). For example, given $G : \Omega^0 \to \mathbb{R}$ the push forward $g(\cdot, t) : \Omega^t \to \mathbb{R}$ is defined as $g(\mathbf{x}, t) = G(\phi^{-1}(\mathbf{x}, t))$. Similarly, the pull back of g is $G(\mathbf{X}) = g(\phi(\mathbf{X}, t), t)$ which can be seen to be exactly $G(\mathbf{X})$ by definition of the inverse mapping. ??

The push forward of a function is sometimes referred to as Eulerian (a function of $\mathbf{x}$) and the pull back function is sometimes referred to as Lagrangian (a function of $\mathbf{X}$). As previously defined in Equation 4 and 5, the velocity and acceleration functions are Lagrangian. Let's rewrite them here:

$$\mathbf{V}(\mathbf{X}, t) = \frac{\partial \phi}{\partial t}(\mathbf{X}, t) \tag{8}$$

$$\mathbf{A}(\mathbf{X}, t) = \frac{\partial^2 \phi}{\partial t^2}(\mathbf{X}, t) = \frac{\partial \mathbf{V}}{\partial t}(\mathbf{X}, t). \tag{9}$$

It is also useful to define Eulerian counterparts. That is, using push forward,

$$\boldsymbol{\nu}(\mathbf{x}, t) = \mathbf{V}(\phi^{-1}(\mathbf{x}, t), t), \tag{10}$$

$$\mathbf{a}(\mathbf{x}, t) = \mathbf{A}(\phi^{-1}(\mathbf{x}, t), t). \tag{11}$$

With this, we can also see that the pull back formula are

$$\mathbf{V}(\mathbf{X}, t) = \boldsymbol{\nu}(\phi(\mathbf{X}, t), t), \tag{12}$$

$$\mathbf{A}(\mathbf{X}, t) = \mathbf{a}(\phi(\mathbf{X}, t), t). \tag{13}$$

With this notion of $\mathbf{a}$ and $\boldsymbol{\nu}$ we can see that (using chain rule)

$$\mathbf{A}(\mathbf{X}, t) = \frac{\partial}{\partial t}\mathbf{V}(\mathbf{X}, t) = \frac{\partial \boldsymbol{\nu}}{\partial t}(\phi(\mathbf{X}, t), t) + \frac{\partial \boldsymbol{\nu}}{\partial \mathbf{x}}(\phi(\mathbf{X}, t), t)\frac{\partial \phi}{\partial t}(\mathbf{X}, t). \tag{14}$$

Using index notation, this can be written as

$$A_i(\mathbf{X}, t) = \frac{\partial}{\partial t}V_i(\mathbf{X}, t) = \frac{\partial \nu_i}{\partial t}(\phi(\mathbf{X}, t), t) + \frac{\partial \nu_i}{\partial x_j}(\phi(\mathbf{X}, t), t)\frac{\partial \phi_j}{\partial t}(\mathbf{X}, t). \tag{15}$$

where summation is implied on the repeated index j.

Combining Equation 8 and 10, we have

$$\nu_j(\mathbf{x}, t) = \frac{\partial \phi_j}{\partial t}(\phi^{-1}(\mathbf{x}, t), t). \tag{16}$$

Combining Equation 11 and 15, we have

$$a_i(\mathbf{x},t) = A_i(\phi^{-1}(\mathbf{x},t),t) = \frac{\partial v_i}{\partial t}(\mathbf{x},t) + \frac{\partial v_i}{\partial x_j}(\mathbf{x},t)v_j(\mathbf{x},t) \tag{17}$$

where we used $\mathbf{x} = \phi(\phi^{-1}(\mathbf{x},t),t)$ (by definition).

We thus get a seemingly non-intuitive result:

$$a_i(\mathbf{x},t) \neq \frac{\partial v_i}{\partial t}(\mathbf{x},t). \tag{18}$$

5.4 Material Derivative

Although the relationship between the Eulerian $\mathbf{a}$ and $\boldsymbol{v}$ is not simply via partial differentiation with respect to time, the relationship is a common one and it is often called the material derivative. The notation

$$\frac{D}{Dt}v_i(\mathbf{x},t) = \frac{\partial v_i}{\partial t}(\mathbf{x},t) + \frac{\partial v_i}{\partial x_j}(\mathbf{x},t)v_j(\mathbf{x},t) \tag{19}$$

is often introduced so that

$$\mathbf{a} = \frac{D}{Dt}\boldsymbol{v}. \tag{20}$$

For a general **Eulerian** function $f(\cdot,t) : \Omega^t \to \mathbb{R}$, we use this same notation to mean

$$\frac{D}{Dt}f(\mathbf{x},t) = \frac{\partial f}{\partial t}(\mathbf{x},t) + \frac{\partial f}{\partial x_j}(\mathbf{x},t)v_j(\mathbf{x},t). \quad !!!! \tag{21}$$

Note that $\frac{D}{Dt}f(\mathbf{x},t)$ is the push forward of $\frac{\partial}{\partial t}F$ where F is a **Lagrangian** function with $F(\cdot,t) : \Omega^0 \to \mathbb{R}$. F is the pull back of f. This rule of pushing forward $\frac{\partial}{\partial t}F(\mathbf{X},t)$ to $\frac{D}{Dt}f(\mathbf{x},t)$ is very useful, and should always be kept in mind.

The deformation gradient is usually thought of as Lagrangian. That is, most of the time when this comes up in the physics of a material, the Lagrangian view is the dominant one. There is however a useful evolution of the Eulerian (push forward) of $\mathbf{F}(\cdot,t) : \Omega^0 \to \mathbb{R}^{d\times d}$. Let $\mathbf{f}(\cdot,t) : \Omega^t \to \mathbb{R}^{d\times d}$ be the push forward of $\mathbf{F}$, then

$$\frac{D}{Dt}\mathbf{f} = \frac{\partial \boldsymbol{v}}{\partial \mathbf{x}}\mathbf{f} \quad \text{or} \quad \frac{D}{Dt}f_{ij} = \frac{\partial v_i}{\partial x_k}f_{kj} \tag{22}$$

with summation implied on the repeated index k. We can see this because

$$\frac{\partial}{\partial t}F_{ij}(\mathbf{X},t) = \frac{\partial}{\partial t}\frac{\partial \phi_i}{\partial X_j}(\mathbf{X},t) = \frac{\partial V_i}{\partial X_j}(\mathbf{X},t) = \frac{\partial v_i}{\partial x_k}(\phi(\mathbf{X},t),t)\frac{\partial \phi_k}{\partial X_j}(\mathbf{X},t), \quad !!!! \tag{23}$$

where the last equality comes from differentiating Equation 12. In some literature (including [Bonet and Wood, 2008] and [Klar et al., 2016]), Equation 22 is written using symbol $\mathbf{F}$ instead of $\mathbf{f}$ as

$$\dot{\mathbf{F}} = (\nabla \boldsymbol{v})\mathbf{F} \quad \text{or} \quad \frac{D\mathbf{F}}{Dt} = (\nabla \boldsymbol{v})\mathbf{F}, \tag{24}$$

while the formula $\dot{\mathbf{F}} = \frac{\partial}{\partial \mathbf{X}}\left(\frac{\partial \phi}{\partial t}\right)$ also appears. When written in such ways, $\mathbf{F}$ and $\mathbf{f}$ are undistinguished and $\dot{\mathbf{F}}$ is used for both time derivatives in the two spaces. It is fine to do so as long as $\mathbf{F} = \mathbf{F}(\mathbf{X}, t)$ or $\mathbf{F} = \mathbf{F}(\mathbf{x}, t)$ is clearly specified in the context. Otherwise, we prefer to keep using $\mathbf{F}$ to denote the Lagrangian one, and $\mathbf{f}$ for the Eulerian one to avoid confusion.

Equation 23 will play an important role in deriving the discretized $\mathbf{F}$ update on each MPM particle (Section 9.4).

5.5 Volume and Area Change

Assume there is a tiny volume dV at the material space, what is the corresponding value of dv in the world space? Consider dV being defined over the standard basis vectors $\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3$ with $dV = dL_1\boldsymbol{e}_1 \cdot (dL_2\boldsymbol{e}_2 \times dL_3\boldsymbol{e}_3)$, where dL_i are tiny numbers, $d\mathbf{L}_i = dL_i\boldsymbol{e}_i$. Then we have

$$dV = dL_1 dL_2 dL_3. \tag{25}$$

The corresponding deformed vectors in the world space are

$$d\mathbf{l}_i = \mathbf{F} d\mathbf{L}_i. \tag{26}$$

It can be shown that $dl_1 dl_2 dl_3 = J dL_1 dL_2 dL_3$ or $dv = J dV$ where $J = \det(\mathbf{F})$.

Given this property, for any function $G(\mathbf{X})$ or $g(\mathbf{x}, t)$, it is very common to use the push forward/pull back when changing variables for integrals defined over subsets of either Ω^0 or Ω^t. That is

$$\int_{B^t} g(\mathbf{x}) d\mathbf{x} = \int_{B^0} G(\mathbf{X}) J(\mathbf{X}, t) d\mathbf{X}, \tag{27}$$

where B^t is an arbitrary subset of Ω^t, B^0 is the pre-image of B^t under $\phi(\cdot, t)$, G is the pull back of g and $J(\mathbf{X}, t)$ is the deformation gradient determinant.

Similar analysis can be done for areas. Consider an arbitrary tiny area dS in Ω^0, denote the corresponding area in Ω^t with ds. Assuming their normals are $\mathbf{N}$ and $\mathbf{n}$ respectively,

$$d\mathbf{S} = (dS)\mathbf{N}, \tag{28}$$

$$d\mathbf{s} = (ds)\mathbf{n}. \tag{29}$$

Consider another tiny vector $d\mathbf{L}$ (with corresponding deformed version $d\mathbf{l}$) that determines a tiny volume when combined with $d\mathbf{S}$ ($d\mathbf{s}$), we have

$$dV = d\mathbf{S} \cdot d\mathbf{L}, \tag{30}$$

$$dv = d\mathbf{s} \cdot d\mathbf{l}. \tag{31}$$

Combining this with the previous result $dv = JdV$, we get

$$Jd\mathbf{S} \cdot d\mathbf{L} = d\mathbf{s} \cdot (\mathbf{F}d\mathbf{L}), \tag{32}$$

where we have used $d\mathbf{l} = \mathbf{F}d\mathbf{L}$. Equation 32 needs to be true for any $d\mathbf{L}$. That results in the relationship

$$d\mathbf{s} = \mathbf{F}^{-T}Jd\mathbf{S}, \tag{33}$$

or

$$\mathbf{n}ds = \mathbf{F}^{-T}J\mathbf{N}dS. \tag{34}$$

We can then use this relation ship to write the surface integrals as

$$\int_{\partial B^t} \mathbf{h}(\mathbf{x}, t) \cdot \mathbf{n}(\mathbf{x}) ds(\mathbf{x}) = \int_{\partial B^0} \mathbf{H}(\mathbf{X}) \cdot \mathbf{F}^{-T}(\mathbf{X}, t)\mathbf{N}(\mathbf{X})J(\mathbf{X}, t)dS(\mathbf{X}) \tag{35}$$

where $\mathbf{H} : \Omega^0 \to \mathbb{R}^d$ is the pull back of $\mathbf{h} : \Omega^t \to \mathbb{R}^d$, $\mathbf{n}(\mathbf{x})$ is the unit outward normal of ∂B^t at $\mathbf{x}$ and $\mathbf{N}(\mathbf{X})$ is the unit outward normal of ∂B^0 at $\mathbf{X}$. These relationships are very useful when deriving the equations of motion.

6 HYPERELASTICITY

It is more natural to introduce the physical meaning of stress when deriving the governing equations (conservation of momentum) in Section 7. For now we simply introduce the fact that stress is related to strain (or deformation gradient in our case) through some "constitutive relationship". The stress is a field that exists in the whole domain. There are multiple stress definitions available. For example, the Cauchy stress is $\sigma(\cdot, t) : \Omega^t \to \mathbb{R}^{d\times d}$. Discretely stress is a small tensor (matrix) at each evaluated point.

A constitutive model relating the state (such as deformation gradient $\mathbf{F}$) to the stress is needed for governing the material responses under deformations. For perfectly hyperelastic materials the constitutive relation is defined through the potential energy, which increases with non-rigid deformation from the initial state.

In this section we focus on elastic materials as well as an inexact but practically easy-to-use plastic model. The models described below have been successfully used for elastic objects, sand, snow, lava and many other materials in the MPM publications and movies.

6.1 First Piola-Kirchoff Stress

For traditional solids, we prefer to express strain stress relationship using deformation gradient and first Piola-Kirchoff stress because they are more naturally expressed in the material space. Hyperelastic materials are those elastic solids whose first Piola-Kirchoff stress $\mathbf{P}$ can be derived from an strain energy density function $\Psi(\mathbf{F})$ via

$$\mathbf{P} = \frac{\partial \Psi}{\partial \mathbf{F}}. \tag{36}$$

With index notation, this means

$$P_{ij} = \frac{\partial \Psi}{\partial F_{ij}}. \tag{37}$$

$\psi(\mathbf{F})$ is the elastic energy density function (a scalar function) designed to penalize non-rigid $\mathbf{F}$. $\mathbf{P}$ is discretely a small matrix with the same dimensions as $\mathbf{F}$.

Note that it is easy to relate $\mathbf{P}$ to the Cauchy stress $\boldsymbol{\sigma}$ which is sometimes more common in the engineering literature via

$$\boldsymbol{\sigma} = \frac{1}{J}\mathbf{P}\mathbf{F}^T = \frac{1}{\det(\mathbf{F})}\frac{\partial \psi}{\partial \mathbf{F}}\mathbf{F}^T. \tag{38}$$

The material behavior is defined via the interaction of ϕ and the stress σ or $\mathbf{P}$. For hyperelastic materials, the stress is a function of the change in shape, as expressed via the deformation gradient. Note that the motion of the material is rigid if

$$\boldsymbol{\phi}(\mathbf{X}, t) = \mathbf{R}(t)\mathbf{X} + \mathbf{t}(t) \tag{39}$$

where $\mathbf{R}^T\mathbf{R} = \mathbf{I}$, $\det(\mathbf{R}) = 1$ and $\mathbf{t} : [0, \infty) \to \mathbb{R}^d$. I.e. $\mathbf{R}$ is the rotation and $\mathbf{t}$ is the translation. Note that in this case, $\mathbf{F} = \mathbf{R}$. Hyperelastic materials penalize deformation via stress that arises from an energy that penalizes deviation of $\mathbf{F}$ from orthogonal. This can be written as

$$\mathbf{P} = \frac{\partial \Psi}{\partial \mathbf{F}}, \quad \Psi(\mathbf{F}) = \tilde{\Psi}(\mathbf{F}^T\mathbf{F}). \tag{40}$$

In other words, the energy does not change (and has a minimum) if $\mathbf{F}$ is orthogonal. $\mathbf{F}^T\mathbf{F}$ is often denoted with $\mathbf{C}$ (right Cauchy-Green tensor). If the material is isotropic (meaning that response to deformation is material direction independent), then we can further simplify the energy by writing it as a function of the invariants of $\mathbf{C}$:

$$\Psi = \tilde{\Psi}(I_1, I_2, I_3) \tag{41}$$

where I_i are the coefficients of the characteristic polynomial of $\mathbf{F}^T\mathbf{F}$ (often called the isotropic invariants) as

$$I_1 = \mathrm{tr}(\mathbf{C}), \tag{42}$$

$$I_2 = \mathrm{tr}(\mathbf{C}\mathbf{C}), \tag{43}$$

$$I_3 = \det(\mathbf{C}) = J^2. \tag{44}$$

In graphics, it has been convenient to further write this as

$$\Psi(\mathbf{F}) = \hat{\Psi}(\mathbf{\Sigma}(\mathbf{F})) \tag{45}$$

where $\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ is the graphics/mechanics favored "Polar SVD" [Irving et al., 2004; McAdams et al., 2011; Gast et al., 2016] (where both [McAdams et al., 2011] and [Gast et al., 2016] release open source code for rapidly computing it). It is called "Polar SVD" for historical reasons. Mainly, $\mathbf{U}$ and $\mathbf{V}$ are rotations, and the Polar decomposition $\mathbf{F} = \mathbf{R}\mathbf{S}$ can be reconstructed via $\mathbf{R} = \mathbf{U}\mathbf{V}^T$ and $\mathbf{S} = \mathbf{V}\Sigma\mathbf{V}^T$, where $\mathbf{R}$ is the closest rotation to $\mathbf{F}$ [Gast et al., 2016] and $\mathbf{S}$ is symmetric.

The isotropic invariants can be written in terms of the singular values so this is always possible. This construction of the constitutive response to deformation can be done intuitively in terms of $\mathbf{\Sigma}(\mathbf{F})$, however it does require differentiating the singular values as a function of $\mathbf{F}$ to get the stress (and stress derivatives) and this requires careful derivation.

6.2 Neo-Hookean

Neo-Hookean is one of the most common nonlinear hyperelastic models for predicting large deformations of elastic materials. The energy density function for this model is

$$\Psi(\mathbf{F}) = \frac{\mu}{2}\left(\mathrm{tr}(\mathbf{F}^T\mathbf{F}) - d\right) - \mu\log(J) + \frac{\lambda}{2}\log^2(J), \tag{46}$$

where $d = 2$ or 3 denotes the problem dimension, μ and λ are related to Young's modulus E and Poisson's ratio ν via

$$\mu = \frac{E}{2(1+\nu)}, \quad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}. \tag{47}$$

It is easy to see that when $\mathbf{F}$ is a rotation, $\Psi(\mathbf{F}) = 0$. For a non-inverted $\mathbf{F}$ (meaning $J > 0$), $\Psi(\mathbf{F}) \geqslant 0$. The energy density function is adequate to describe a hyperelastic solid. For force computations, Equation 36 is needed to provide $\mathbf{P}$ as a function of $\mathbf{F}$. For Neo-Hookean, the result is

$$\mathbf{P} = \mu(\mathbf{F} - \mathbf{F}^{-T}) + \lambda\log(J)\mathbf{F}^{-T}. \tag{48}$$

Note that $\frac{\partial \mathbf{P}}{\partial \mathbf{F}}$ is further needed for implicit integration. We provide in Section 6.4 a practical way to do so.

6.3 Fixed Corotated Constitutive Model

Another simple and widely used model that is defined from the Singular Value Decomposition (SVD) is the so called fixed corotated model. This is called "fixed" because it is a

small modification to a commonly used model called corotated linear elasticity common in the computer graphics literature. Assuming the polar SVD $\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, the energy for fixed corotated model is

$$\Psi(\mathbf{F}) = \hat{\Psi}(\mathbf{\Sigma}(\mathbf{F})) = \mu \sum_{i=1}^{d} (\sigma_i - 1)^2 + \frac{\lambda}{2}(J-1)^2 \tag{49}$$

where of course $J = \prod_{i=1}^{d} \sigma_i$. Expanding the μ term in the formula we have

$$\mu \sum_{i=1}^{d} (\sigma_i - 1)^2 = \mu \left(\sum_{i=1}^{d} \sigma_i^2 - 2 \sum_{i=1}^{d} \sigma_i + d \right). \tag{50}$$

It can be shown that

$$\frac{\partial}{\partial \mathbf{F}} \sum_{i=1}^{d} \sigma_i^2 = 2\mathbf{F} \quad \text{and} \quad \frac{\partial}{\partial \mathbf{F}} \sum_{i=1}^{d} \sigma_i = \mathbf{R} \qquad ! \ ! \ ! \tag{51}$$

where $\mathbf{F} = \mathbf{R}\mathbf{S}$ is the polar decomposition of $\mathbf{F}$ ($\mathbf{R}$ a rotation matrix and $\mathbf{S}$ symmetric). This can of course be defined from the SVD of $\mathbf{F}$ as $\mathbf{F} = \mathbf{U}\mathbf{V}^T\mathbf{V}\mathbf{\Sigma}\mathbf{V}^T$. I.e. $\mathbf{R} = \mathbf{U}\mathbf{V}^T$ and $\mathbf{S} = \mathbf{V}\mathbf{\Sigma}\mathbf{V}^T$. Combining all of this, we have

$$\mathbf{P}(\mathbf{F}) = \frac{\partial \psi}{\partial \mathbf{F}}(\mathbf{F}) = 2\mu(\mathbf{F} - \mathbf{R}) + \lambda(J-1)J\mathbf{F}^{-T}. \tag{52}$$

The second derivatives require a bit more care but can be computed relatively easily.

We first do it by computing differentials (which will results in $\delta\mathbf{P}$). This will be useful for a matrix free implementation of the implicit solver.

$$\begin{aligned}
\frac{\partial^2 \Psi}{\partial \mathbf{F} \partial \mathbf{F}} : \delta\mathbf{F} &= \delta\left(\frac{\partial \Psi}{\partial \mathbf{F}}\right) && (53)\\
&= \delta(2\mu(\mathbf{F} - \mathbf{R}) + \lambda(J-1)J\mathbf{F}^{-T}) && (54)\\
&= 2\mu\delta\mathbf{F} - 2\mu\delta\mathbf{R} + \lambda J\mathbf{F}^{-T}\delta J + \lambda(J-1)\delta(J\mathbf{F}^{-T}) && (55)\\
&= 2\mu\delta\mathbf{F} - 2\mu\delta\mathbf{R} + \lambda J\mathbf{F}^{-T}(J\mathbf{F}^{-T} : \delta\mathbf{F}) + \lambda(J-1)\delta(J\mathbf{F}^{-T}) && (56)
\end{aligned}$$

Since $J\mathbf{F}^{-T}$ is a matrix whose entries are polynomials in the entries of $\mathbf{F}$, $\delta(J\mathbf{F}^{-T}) = \frac{\partial}{\partial \mathbf{F}}(J\mathbf{F}^{-T}) : \delta\mathbf{F}$ can readily be computed directly. That leaves the task of computing $\delta\mathbf{R}$.

$$\begin{aligned}
\delta\mathbf{F} &= \delta\mathbf{R}\mathbf{S} + \mathbf{R}\delta\mathbf{S} && (57)\\
\mathbf{R}^T\delta\mathbf{F} &= (\mathbf{R}^T\delta\mathbf{R})\mathbf{S} + \delta\mathbf{S} && (58)\\
\mathbf{R}^T\delta\mathbf{F} - \delta\mathbf{F}^T\mathbf{R} &= (\mathbf{R}^T\delta\mathbf{R})\mathbf{S} + \mathbf{S}(\mathbf{R}^T\delta\mathbf{R}) && (59)
\end{aligned}$$

Here we have taken advantage of the symmetry of $\delta\mathbf{S}$ and the skew symmetry of $\mathbf{R}^T\delta\mathbf{R}$. There are three independent components of $\mathbf{R}^T\delta\mathbf{R}$, which we can solve for directly. The equation is linear in these components, so $\mathbf{R}^T\delta\mathbf{R}$ can be computed by solving a 3×3 system. Finally, $\delta\mathbf{R} = \mathbf{R}(\mathbf{R}^T\delta\mathbf{R})$.

6.4 A Practical Differentiation Strategy for Isotropic Elasticity

Here we provide a practical way (originated from [Stomakhin et al., 2012]) to compute $\mathbf{P}$ and $\frac{\partial \mathbf{P}}{\partial \mathbf{F}}$ (either the tensor or the differential $\delta\mathbf{P}$) for any general isotropic elastic material. This method will utilize a symbolic software package. We will discuss the *Mathematica* implementation. A *Maple* or any other version is straightforward to produce following the same logic. For a more thorough discussion on implementing derivative computations that happen a lot in computer graphics applications, we refer to [Schroeder, 2016].

It is worth noting that this strategy can be used for computing other derivatives in diagonal space that looks similar to $\frac{\partial \hat{\mathbf{P}}}{\partial \mathbf{F}}$. For example in some models, the Kirchoff stress τ (instead of first Piola-Kirchoff stress $\mathbf{P}$) is used:

$$\tau = \mathbf{U}\hat{\tau}\mathbf{U}^T, \tag{60}$$

where $\hat{\tau}$ is a diagonal stress measure with each entry being a function of Σ. To compute $\frac{\partial \tau}{\partial \mathbf{F}}$, almost exactly the same method discussed below for computing $\frac{\partial \mathbf{P}}{\partial \mathbf{F}}$ can be used.

6.4.1 *Computing* $\mathbf{P}$

Let's start with $\mathbf{P}$. For an isotropic material, stress can be computed as

$$\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T, \tag{61}$$

$$\Psi(\mathbf{F}) = \hat{\Psi}(\mathbf{\Sigma}), \tag{62}$$

$$\mathbf{P} = \mathbf{U}\hat{\mathbf{P}}\mathbf{V}^T. \tag{63}$$

Here, $\hat{\mathbf{P}}$ is diagonal with entries $\frac{\partial \hat{\Psi}}{\partial \sigma_i}$.

Here is the proof for $\mathbf{P}$ in diagonal space (Equation 63): First, we want to show $\mathbf{P}(\mathbf{RF}) = \mathbf{R}\mathbf{P}(\mathbf{F})$ for any rotation $\mathbf{R}$. Consider any model (doesn't even need to be isotropic), rotation after deformation shouldn't change the energy. This means $\Psi(\mathbf{F}) = \Psi(\mathbf{RF})$. Take the differentials to this equation, we get

$$\begin{aligned}
\delta\Psi = \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}) : \delta\mathbf{F} &= \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{RF}) : \delta(\mathbf{RF}) \\
(\mathbf{P}(\mathbf{F})) : (\delta\mathbf{F}) &= (\mathbf{P}(\mathbf{RF})) : \delta(\mathbf{RF}) \\
(\mathbf{P}(\mathbf{F})) : (\delta\mathbf{F}) &= (\mathbf{P}(\mathbf{RF}))_{ij} R_{ik} \delta F_{kj} \\
(\mathbf{P}(\mathbf{F})) : (\delta\mathbf{F}) &= (\mathbf{R}^T\mathbf{P}(\mathbf{RF})) : \delta\mathbf{F} \\
\mathbf{P}(\mathbf{F}) &= \mathbf{R}^T\mathbf{P}(\mathbf{RF}) \\
\mathbf{R}\mathbf{P}(\mathbf{F}) &= \mathbf{P}(\mathbf{RF})
\end{aligned}$$

Similar, if we assume an isotropic material, we can use $\Psi(\mathbf{FR}) = \Psi(\mathbf{F})$ to prove $\mathbf{P}(\mathbf{FR}) = \mathbf{P}(\mathbf{F})\mathbf{R}$ for any rotation $\mathbf{R}$. Using these two qualities for $\mathbf{P}$, we have

$$\mathbf{P}(\mathbf{F}) = \mathbf{P}(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T) = \mathbf{U}\mathbf{P}(\Sigma)\mathbf{V}^T = \mathbf{U}\hat{\mathbf{P}}\mathbf{V}^T. \tag{64}$$

6.4.2 *Computing* $\partial \mathbf{P}/\partial \mathbf{F}$ *or* $\delta \mathbf{P}$

The idea is similar to the one for $\mathbf{P}$. Take any two rotations $\mathbf{R}$ and $\mathbf{Q}$, use the result for $\mathbf{P}$, we have

$$\mathbf{P}(\mathbf{F}) = \mathbf{P}(\mathbf{R}\mathbf{R}^T\mathbf{F}\mathbf{Q}\mathbf{Q}^T) = \mathbf{R}\mathbf{P}(\mathbf{R}^T\mathbf{F}\mathbf{Q})\mathbf{Q}^T. \tag{65}$$

Call $\mathbf{K} = \mathbf{R}^T\mathbf{F}\mathbf{Q}$, we have

$$\mathbf{P}(\mathbf{F}) = \mathbf{R}\mathbf{P}(\mathbf{K})\mathbf{Q}^T \tag{66}$$

Now take the differential (and keep in mind that $\mathbf{R}$ and $\mathbf{Q}$ should be treat as constants):

$$\delta\mathbf{P} = \mathbf{R}\left[\frac{\partial \mathbf{P}}{\partial \mathbf{F}}(\mathbf{K}) : \delta(\mathbf{K})\right]\mathbf{Q}^T \tag{67}$$

$$= \mathbf{R}\left[\frac{\partial \mathbf{P}}{\partial \mathbf{F}}(\mathbf{K}) : (\mathbf{R}^T\delta\mathbf{F}\mathbf{Q})\right]\mathbf{Q}^T. \tag{68}$$

Since $\mathbf{R}$ and $\mathbf{Q}$ are freely chosen, we choose $\mathbf{R} = \mathbf{U}$ and $\mathbf{Q} = \mathbf{V}$, then $\mathbf{K} = \Sigma$. The formula then becomes

$$\delta\mathbf{P} = \mathbf{U}\left[\frac{\partial \mathbf{P}}{\partial \mathbf{F}}(\Sigma) : (\mathbf{U}^T\delta\mathbf{F}\mathbf{V})\right]\mathbf{V}^T. \tag{69}$$

This gives us $\delta\mathbf{P}$. For the tensor $\partial\mathbf{P}/\partial\mathbf{F}$, we adopt the index notation:

$$(\delta\mathbf{P})_{ij} = U_{ik}\left(\frac{\partial \mathbf{P}}{\partial \mathbf{F}}(\Sigma)\right)_{klmn} U_{rm}\delta F_{rs} V_{sn} V_{jl} \tag{70}$$

$$(\delta\mathbf{P})_{ij} = \left(\frac{\partial \mathbf{P}}{\partial \mathbf{F}}(\mathbf{F})\right)_{ijrs}\delta F_{rs} \tag{71}$$

These two equations need to hold for any $\delta\mathbf{F}$, revealing

$$\left(\frac{\partial \mathbf{P}}{\partial \mathbf{F}}(\mathbf{F})\right)_{ijrs} = \left(\frac{\partial \mathbf{P}}{\partial \mathbf{F}}(\Sigma)\right)_{klmn} U_{ik}U_{rm}V_{sn}V_{jl} \tag{72}$$

So the remaining problem is computing $\frac{\partial \mathbf{P}}{\partial \mathbf{F}}(\Sigma)$. We show how to do it in 3D.

First, we need to use Rodrigues' rotation formula. It says any rotation matrix can be written in terms of a unit vector $\mathbf{k}$ and an rotation angle θ:

$$\mathbf{R} = \mathbf{I} + \sin(\theta)\mathbf{K} + (1 - \cos(\theta))\mathbf{K}^2, \tag{73}$$

where $\mathbf{K}$ is the skew-symmetric cross-product matrix of $\mathbf{k}$. This means every rotation matrix has only 3 degree of freedoms $r1, r2, r3$, then $\mathbf{k} = \frac{\mathbf{r}}{|\mathbf{r}|}, \theta = |\mathbf{r}|$. So, we can parametrize rotation matrix $\mathbf{U}$ and $\mathbf{V}$ with 3 numbers for each.

Now we have the following code for defining $\mathbf{F}$ in terms of $s1, s2, s3, u1, u2, u3, v1, v2, v3$, where $\mathbf{U}$ and $\mathbf{V}$ are defined by u_i and v_i with Rodrigues' rotation formula, s_i are the singular values from Σ.

```
id=IdentityMatrix[3];
var={s1,s2,s3,u1,u2,u3,v1,v2,v3};
Sigma=DiagonalMatrix[{s1,s2,s3}];
cp[k1_,k2_,k3_]={{0,-k3,k2},{k3,0,-k1},{-k2,k1,0}};
vV={v1,v2,v3};
vU={u1,u2,u3};
nv=Sqrt[Dot[vV,vV]];
nu=Sqrt[Dot[vU,vU]];
UU=cp[u1,u2,u3]/nu;
VV=cp[v1,v2,v3]/nv;
U=id+Sin[nu]*UU+(1-Cos[nu])*UU.UU;
V=id+Sin[nv]*VV+(1-Cos[nv])*VV.VV;
F=U.Sigma.Transpose[V];
```

where cp is a function for generating the cross-product matrix (corresponding to computing $\mathbf{K}$ in Equation 73).

From now on, we write the $3\times3\times3\times3$ tensor $\frac{\partial \mathbf{P}}{\partial \mathbf{F}}(\Sigma)$ and any other such tensors to 9×9 matrices. That means each 3×3 matrix is now a size-9 vector. It is easy to see the old $\frac{\partial P_{ij}}{\partial F_{kl}}$ is now $\frac{\partial P_{3(i-1)+j}}{\partial F_{3(k-1)+l}}$. We further call vector $\mathbf{S}=\{s1,s2,s3,u1,u2,u3,v1,v2,v3\}$ being the parametrization of $\mathbf{F}$. Then we can apply the chain rule

$$\frac{\partial \mathbf{P}}{\partial \mathbf{F}}(\Sigma)=\frac{\partial \mathbf{P}}{\partial \mathbf{S}}(\Sigma)\frac{\partial \mathbf{S}}{\partial \mathbf{F}}(\Sigma) \tag{74}$$

Here are the Mathematica code for computing them. Note that we achieve $\mathbf{F}=\Sigma$ by taking the limit $\{u1,u2,u3,v1,v2,v3\}=+\epsilon$, which correspond to nearly zero rotations.

```
dFdS=D[Flatten[F],{var}];
dFdSo=dFdS/.{u1->e,u2->e,u3->e,v1->e,v2->e,v3->e};
dFdS1=Limit[dFdSo,e->0,Direction->-1];
dSdFo=Inverse[dFdS1];
Phat=DiagonalMatrix[{t1[s1,s2,s3],t2[s1,s2,s3],t3[s1,s2,s3]}];
P=U.Phat.Transpose[V];
dPdS=D[Flatten[P],{var}];
dPdSo=dPdS/.{u1->e,u2->e,u3->e,v1->e,v2->e,v3->e};
dPdS1=Limit[dPdSo,e->0,Direction->-1];
dPdF=Simplify[dPdS1.dSdFo];
```

Note 'Direction->-1' in Mathematica means taking the limit from large values to the small limit value. The Mathematica computation result will be given in terms of the singular values and $\hat{\mathbf{P}}$. One can then take the formula for implementing them in the code. [Stomakhin et al., 2012] gives the result where $\partial\mathbf{P}/\partial\mathbf{F}$ (size 9×9 matrix) is permuted to be a block diagonal matrix with diagonal blocks $\mathbf{A}^{3\times3}, \mathbf{B}_{12}^{2\times2}, \mathbf{B}_{13}^{2\times2}, \mathbf{B}_{23}^{2\times2}$, where

$$\mathbf{A}=\begin{pmatrix}\hat{\psi}_{,\sigma_1\sigma_1} & \hat{\psi}_{,\sigma_1\sigma_2} & \hat{\psi}_{,\sigma_1\sigma_3}\\ \hat{\psi}_{,\sigma_2\sigma_1} & \hat{\psi}_{,\sigma_2\sigma_2} & \hat{\psi}_{,\sigma_2\sigma_3}\\ \hat{\psi}_{,\sigma_3\sigma_1} & \hat{\psi}_{,\sigma_3\sigma_2} & \hat{\psi}_{,\sigma_3\sigma_3}\end{pmatrix} \tag{75}$$

and

$$\mathbf{B}_{ij} = \frac{1}{\sigma_i^2 - \sigma_j^2} \begin{pmatrix} \sigma_i \hat{\psi}_{,\sigma_i} - \sigma_j \hat{\psi}_{,\sigma_j} & \sigma_j \hat{\psi}_{,\sigma_i} - \sigma_i \hat{\psi}_{,\sigma_j} \\ \sigma_j \hat{\psi}_{,\sigma_i} - \sigma_i \hat{\psi}_{,\sigma_j} & \sigma_i \hat{\psi}_{,\sigma_i} - \sigma_j \hat{\psi}_{,\sigma_j} \end{pmatrix}. \tag{76}$$

The division by $\sigma_i^2 - \sigma_j^2$ is problematic when two singular values are nearly equal or when two singular values nearly sum to zero. The latter is possible with a convention for permitting negative singular values (as in invertible elasticity [Irving et al., 2004; Stomakhin et al., 2012]).

Expanding $\mathbf{B}_{ij}$ in terms of partial fractions yields the useful decomposition

$$\mathbf{B}_{ij} = \frac{1}{2} \frac{\hat{\psi}_{,\sigma_i} - \hat{\psi}_{,\sigma_j}}{\sigma_i - \sigma_j} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} + \frac{1}{2} \frac{\hat{\psi}_{,\sigma_i} + \hat{\psi}_{,\sigma_j}}{\sigma_i + \sigma_j} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}. \tag{77}$$

Note that if $\hat{\psi}$ is invariant under permutation of the singular values, then $\hat{\psi}_{,\sigma_i} \to \hat{\psi}_{,\sigma_j}$ as $\sigma_i \to \sigma_j$. Thus, the first term can normally be computed robustly for an isotropic model if implemented carefully. The other fraction has deeper implications. This term can be computed robustly if $\hat{\psi}_{,\sigma_i} + \hat{\psi}_{,\sigma_j} \to 0$ as $\sigma_i + \sigma_j \to 0$. This property is unfavorable, as it means the constitutive model will have difficulty recovering from many inverted configurations. Since we are usually interested in models with robust behavior under inversion, this term will necessarily be unbounded under some circumstances. We address this by clamping the magnitude of the denominator to not be smaller than 10^{-6} before division to bound the derivatives.

For 2D, a rotation matrix is now simply paremetrized with a single θ where the reconstruction is

$$\mathbf{R} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}.$$

The 2D version of the whole Mathematica code is

```
id=IdentityMatrix[2];
var={s1,s2,u1,v1};
S=DiagonalMatrix[{s1,s2}];
U={{Cos[u1],-Sin[u1]},{Sin[u1],Cos[u1]}};
V={{Cos[v1],-Sin[v1]},{Sin[v1],Cos[v1]}};
F=U.S.Transpose[V];
dFdS=D[Flatten[F],{var}];
dFdSo=dFdS/.{u1->e,v1->e};
dFdS1=Limit[dFdSo,e->0,Direction->-1];
dSdFo=Inverse[dFdS1];
Phat=DiagonalMatrix[{t1[s1,s2],t2[s1,s2]}];
P=U.Phat.Transpose[V];
dPdS=D[Flatten[P],{var}];
dPdSo=dPdS/.{u1->e,v1->e};
dPdS1=Limit[dPdSo,e->0,Direction->-1];
dPdF=Simplify[dPdS1.dSdFo];
```

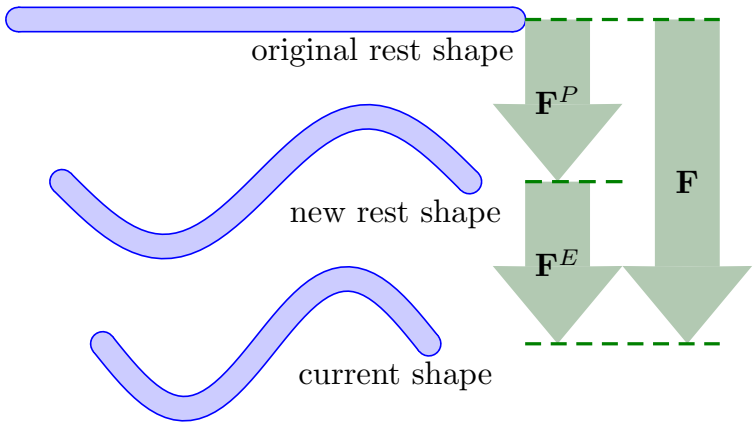


Figure 2: Multiplicative decomposition of the deformation gradient.

6.5 Snow Plasticity

Snow constitutive behavior depends on a very wide range of complex factors. There are many different models used depending on the conditions of interest. Much of the dynamic behavior can be approximated with a relatively simple elasto-plastic assumption. We will use a simple large-strain plastic flow model combined with a hardening effect.

We represent plasticity by factoring deformation gradient into elastic and plastic parts as

$$\mathbf{F} = \mathbf{F}_E \mathbf{F}_P. \tag{78}$$

The deformation gradient is a measure of how a material has locally rotated and deformed due to its motion. By factoring the deformation gradient in this way, we divide this deformation history into two pieces. The plastic part, $\mathbf{F}_P$, represents the portion of the material's history that has been forgotten. If a metal rod is bent into a coiled spring, the rod *forgets* that it used to be straight; the coiled spring behaves as though it was always coiled (see Figure 2). The twisting and bending involved in this operation is stored in $\mathbf{F}_P$. If the spring is compressed slightly, the spring will feel strain (deformation). This is elastic deformation, which is stored in $\mathbf{F}_E$. The spring remembers this deformation. In response, the material exerts stress to try to restore itself to its coiled shape. In this way, we see that only $\mathbf{F}_E$ should be used to compute stress. The full history of the metal rod consists of being bent into a spring shape ($\mathbf{F}_P$) and then being compressed ($\mathbf{F}_E$).

The elastic response is only a function of $\mathbf{F}_E$. Intuitively, this states that the deformation in the local transition to $\mathbf{F}_P$ is permanent. In a sense, $\mathbf{F}_p$ comes to represent a new local rest state of the material. This transition to permanent deformation is typically in response to large deformation. A simple example of this would be denting an aluminum can. Once dented, all elastic response will be to displacement from the dented configuration. Most models define the decomposition in response to a stress based condition. However, for our case it will be more intuitive to think of the constraint as defined on the

singular values of $\mathbf{F}$ itself. This will give us more visual control over the plasticity effect. Specifically, we will enforce that the singular values σ_{Ei} of $\mathbf{F}_E$ are in $[1-\theta_c, 1+\theta_s]$ for some small constants θ_c and θ_s. This will be done with the following procedure. Given

$$\mathbf{F}^n = \mathbf{F}_E^n \mathbf{F}_P^n, \tag{79}$$

(where the singular values σ_{Ei}^n of $\mathbf{F}_E^n$ satisfy the constraint of lying in $[1-\theta_c, 1+\theta_s]$) and a new $\mathbf{F}^{n+1}$, we will first assume that all new deformation introduced in the transition from $\mathbf{F}^n$ to $\mathbf{F}^{n+1}$ was elastic. That is we will first assume that given a new $\mathbf{F}^{n+1}$, it can be decomposed as

$$\mathbf{F}^{n+1} = \tilde{\mathbf{F}}_E^{n+1} \mathbf{F}_P^n. \tag{80}$$

In so doing, this defines $\tilde{\mathbf{F}}_E^{n+1}$ as

$$\tilde{\mathbf{F}}_E^{n+1} = \mathbf{F}^{n+1} \left(\mathbf{F}_P^n\right)^{-1}. \tag{81}$$

In practice it is often more convenient to store $\mathbf{F}_E$ and $\mathbf{F}_P$ instead of storing the full $\mathbf{F}$. The tentative update of $\mathbf{F}$ can then be applied to $\mathbf{F}_E$ directly to give $\tilde{\mathbf{F}}_E^{n+1}$. The next step is to enforce the constraint that the singular values $\tilde{\sigma}_{Ei}^{n+1}$ of $\tilde{\mathbf{F}}_E^{n+1}$ satisfy the constraint of lying in $[1-\theta_c, 1+\theta_s]$. That is, we define

$$\sigma_{Ei}^{n+1} = \text{clamp}(\tilde{\sigma}_{Ei}^{n+1}, 1-\theta_c, 1+\theta_s), \quad i = 1, \ldots, d \tag{82}$$

Now, assuming the singular value decomposition of $\tilde{\mathbf{F}}_E^{n+1}$ is

$$\tilde{\mathbf{F}}_E^{n+1} = \mathbf{U}_E^{n+1} \tilde{\Sigma}_E {\mathbf{V}_E^{n+1}}^T, \tag{83}$$

we can define $\mathbf{F}_E^{n+1}$ from the clamped singular values Σ_E^{n+1} as

$$\mathbf{F}_E^{n+1} = \mathbf{U}_E^{n+1} \Sigma_E^{n+1} {\mathbf{V}_E^{n+1}}^T. \tag{84}$$

Of course with this definition of $\mathbf{F}_E^{n+1}$ we still need to maintain the same decomposition of $\mathbf{F}$ so we would need to determine a new $\mathbf{F}_P^{n+1}$ such that

$$\mathbf{F}^{n+1} = \mathbf{F}_E^{n+1} \mathbf{F}_P^{n+1}. \tag{85}$$

But of course, given that we know $\mathbf{F}^{n+1}$ and $\mathbf{F}_E^{n+1}$, the new plastic component of the deformation gradient is

$$\mathbf{F}_P^{n+1} = \left(\mathbf{F}_E^{n+1}\right)^{-1} \mathbf{F}^{n+1}. \tag{86}$$

Snow will tend to get more rigid under compression. This phenomenon is often called hardening. We use a simple modification to our constitutive model to add this effect. Specifically, we let the Lame coefficients μ and λ increase under compression and decrease under extension. The reduction in material strength under extension facilitates break-up and fracture of the snow. This is an important property for a wide range of visual phenomena. We quantify this hardening effect as

$$\mu(\mathbf{F}_P) = \mu_0 e^{\xi(1-J_P)}, \quad \lambda(\mathbf{F}_P) = \lambda_0 e^{\xi(1-J_P)} \tag{87}$$

where μ_0 and λ_0 are the Lame parameters as set from the original Young's modulus and Poisson ratio. ξ is a hardening parameter that we typically use something between 3 and 10. Also, $J_P = \det(\mathbf{F}_P)$. It is also important to include some safeguards to prevent μ and λ becoming too large. This can be simply done by requiring a numerical clamping bound on $\xi(1 - J_P)$.

It can also be seen from the derivation that instead of storing $\mathbf{F}_p$, keeping track of J_p is enough as long as we know how to get $\tilde{\mathbf{F}}_E^{n+1}$ from the time n quantities. For MPM this is indeed the case.

This hardening model is designed with the intuition that when snow is stretched, it becomes softer (to allow fracture). When it is compressed, it becomes stiffer like packing a snow ball. The rule in Equation 87 is just an empirical formula. Being creative on the rules will help produce more versatile and artistic material behaviors.

A more rigorously derived plasticity model needs to obey the second law of thermodynamics as well as enforcing $J_P = 1$ (plasticity deformation should not change the material volume). We refer to the technical document of [Klar et al., 2016] for a more thorough discussion on this matter.

7 GOVERNING EQUATIONS

The governing equations of interest are conservation of mass and conservation of momentum. We'll list the result below and provide their derivations in Section 7.1 and 7.2. We further derive the weak form of the force balance in Section 7.3. The weak form is essential for deriving the final temporal and spatial discretization of the equations in Section 9. It is recommended to review Section 5 before looking into the derivations.

Letting $\mathbf{V}(\mathbf{X}, t) = \frac{\partial \Phi(\mathbf{X},t)}{\partial t} = \frac{\partial \mathbf{x}(\mathbf{X},t)}{\partial t}$ be the velocity defined over $\mathbf{X}$, the Lagrangian view of the equations are [Gonzalez and Stuart, 2008]

$$R(\mathbf{X}, t)J(\mathbf{X}, t) = R(\mathbf{X}, 0) \quad \text{Conservation of mass,} \tag{88}$$

$$R(\mathbf{X}, 0)\frac{\partial \mathbf{V}}{\partial t} = \nabla^{\mathbf{X}} \cdot \mathbf{P} + R(\mathbf{X}, 0)\mathbf{g} \quad \text{Conservation of momentum,} \tag{89}$$

where $\mathbf{X} \in \Omega_0, t > 0$. Here R is the Lagrangian mass density which is related to the more commonly used Eulerian mass density ρ as $R(\mathbf{X}, t) = \rho(\Phi(\mathbf{X}, t), t)$. Note that the mass conservation can also be written as $\frac{\partial}{\partial t}(R(\mathbf{X}, t)J(\mathbf{X}, t)) = 0$.

In Eulerian view, the governing equations are

$$\frac{D}{Dt}\rho(\mathbf{x}, t) + \rho(\mathbf{x}, t)\nabla^{\mathbf{x}} \cdot \mathbf{v}(\mathbf{x}, t) = 0 \quad \text{Conservation of mass,} \tag{90}$$

$$\rho(\mathbf{x}, t)\frac{D\mathbf{v}}{Dt} = \nabla^{\mathbf{x}} \cdot \boldsymbol{\sigma} + \rho(\mathbf{x}, t)\mathbf{g} \quad \text{Conservation of momentum,} \tag{91}$$

where $\boldsymbol{v} = \boldsymbol{v}(\boldsymbol{x}, t)$ is the Eulerian velocity, $\frac{D}{Dt} = \frac{\partial}{\partial t} + \boldsymbol{v} \cdot \nabla^x$ is the material derivative.

As shown in the conservation of momentum, it is more convenient to use the first Piola-Kirchoff stress $\mathbf{P}$ in the Lagrangian view, and the Cauchy stress σ in the Eulerian view.

7.1 Conservation of Mass

Let $\rho(\boldsymbol{x}, t)$ be the Eulerian mass density and let $R(\mathbf{X}, t)$ be the Lagrangian counterpart (pull back), we have the relationship

$$R(\mathbf{X}, t) = \rho(\phi(\mathbf{X}, t), t) \tag{92}$$

$$\rho(\boldsymbol{x}, t) = R(\phi^{-1}(\boldsymbol{x}, t), t). \tag{93}$$

We can think of ρ to be naturally defined over Ω^t as

$$\rho(\boldsymbol{x}, t) = \lim_{\epsilon \to +0} \frac{\text{mass}(B_\epsilon^t)}{\int_{B_\epsilon^t} d\boldsymbol{x}} \tag{94}$$

where B_ϵ^t is the ball of radius ϵ surrounding an arbitrary $\boldsymbol{x} \in \Omega^t$. This is arguably a natural definition since $\text{mass}(B_\epsilon^t)$ should be a measurable quantity. Conservation of mass can be expressed as

$$\text{mass}(B_\epsilon^t) = \int_{B_\epsilon^t} \rho(\boldsymbol{x}, t) d\boldsymbol{x} = \int_{B_\epsilon^0} R(\mathbf{X}, t) J d\mathbf{X} = \int_{B_\epsilon^0} R(\mathbf{X}, 0) d\mathbf{X} = \text{mass}(B_\epsilon^0) \tag{95}$$

for all $B_\epsilon^t \subset \Omega^t$ (as before, B_ϵ^0 is the pre-image of B_ϵ^t under $\phi(\cdot, t)$). The second equality comes from using the change of variable result from Section 5.5.

This just says that the mass in B_ϵ^t (as expressed via an integral of the mass density) should not change with time. This set is associated with a subset of the material at time t and as it evolves in the flow, the material will take up more or less space, but there will never be more or less material in the set. Since B_ϵ^t is arbitrary, it must be true that

$$R(\mathbf{X}, t) J(\mathbf{X}, t) = R(\mathbf{X}, 0), \quad \forall \mathbf{X} \in \Omega^0, t \geqslant 0. \tag{96}$$

Note that $J(\mathbf{X}, 0) = 1$. Alternatively, mass conservation can be written as

$$\frac{\partial}{\partial t} (R(\mathbf{X}, t) J(\mathbf{X}, t)) = 0. \tag{97}$$

To switch to the Eulerian view, we first notice that

$$\frac{\partial}{\partial t} (RJ) = \frac{\partial R}{\partial t} J + R \frac{\partial J}{\partial t}. \tag{98}$$

Also,

$$\frac{\partial J}{\partial t} = \frac{\partial J}{\partial F_{ij}} \frac{\partial F_{ij}}{\partial t} = J F_{ji}^{-1} \frac{\partial V_i}{\partial X_j} = J F_{ji}^{-1} \frac{\partial v_i}{\partial x_k} F_{kj} = J \delta_{ik} \frac{\partial v_i}{\partial x_k} = J \frac{\partial v_i}{\partial x_i}, \tag{99}$$

where we have used the determinant differentiation rule

$$\frac{\partial J}{\partial \mathbf{F}} = J\mathbf{F}^{-T}. \tag{100}$$

Combining Equation 97, 98 and 99, we get

$$\frac{\partial R}{\partial t}J + RJ\frac{\partial v_i}{\partial x_i} = 0. \tag{101}$$

Pushing forward on both sides result in the Eulerian view of conservation of mass:

$$\frac{D}{Dt}\rho(\mathbf{x}, t) + \rho(\mathbf{x}, t)\nabla \cdot \boldsymbol{v}(\mathbf{x}, t) = 0, \quad \forall \mathbf{x} \in \Omega^t, t \geqslant 0. \tag{102}$$

7.2 Conservation of Momentum

Continuum forces are classified as either body forces (e.g. gravity) or surface forces (stress-based). Stress-based forces are first defined via a traction field whose existence we will assume. The force per area (or traction) field $\mathbf{t}(\cdot, \mathbf{n}, t) : \Omega^t \to \mathbb{R}^d$ is defined via the relation

$$\text{force}_S(B_\epsilon^t) = \int_{\partial B_\epsilon^t} \mathbf{t}(\mathbf{x}, \mathbf{n}(\mathbf{x}))ds(\mathbf{x}) \tag{103}$$

where $\text{force}_S(B_\epsilon^t)$ is the net force on an arbitrary B_ϵ^t exerted from material on the other side of ∂B_ϵ^t on material inside B_ϵ^t. That is, $\mathbf{t}(\mathbf{x}, \mathbf{n}, t)$ is the force per unit area($d = 3$)/length($d = 2$) that material in the $\mathbf{n}^+$ side of the material at the point $\mathbf{x}$ exerts on material on the $\mathbf{n}^-$ side. It can be shown that this implies the existence of a stress field (Cauchy stress) $\sigma(\cdot, t) : \Omega^t \to \mathbb{R}^{d\times d}$ with

$$\mathbf{t}(\mathbf{x}, \mathbf{n}, t) = \sigma(\mathbf{x}, t)\mathbf{n}. \tag{104}$$

Let $\boldsymbol{v}$ be the Eulerian velocity (with Lagrangian counterpart $\mathbf{V}$). Then conservation of momentum is expressed as

$$\begin{aligned}\frac{d}{dt}\int_{B_\epsilon^t} \rho(\mathbf{x}, t)\boldsymbol{v}(\mathbf{x}, t)d\mathbf{x} = \frac{d}{dt}\int_{B_\epsilon^0} R(\mathbf{X}, t)\mathbf{V}(\mathbf{X}, t)Jd\mathbf{X} = \int_{B_\epsilon^0} R(\mathbf{X}, 0)\mathbf{A}(\mathbf{X}, t)d\mathbf{X} \\ = \int_{\partial B_\epsilon^t} \sigma\mathbf{n}ds(\mathbf{x}) + \int_{B_\epsilon^t} \mathbf{f}^{ext}d\mathbf{x}\end{aligned} \tag{105}$$

for all $B_\epsilon^t \subset \Omega^t$ in the time t configuration of the material. Here we have added the contribution of external body force (such as gravity) $\mathbf{f}^{ext}$ to the change of momentum. $\mathbf{f}^{ext}$ represents the external body force per unit volume. Therefore the rate of change of the momentum in B_ϵ^t is equal to the net force on B_ϵ^t as expressed via the Cauchy stress field plus the external force. It can also be shown that $\sigma(\mathbf{x}, t)$ must be symmetric for conservation of angular momentum [Bonet and Wood, 2008].

The last equality of Equation 105 can also be written as

$$\int_{B_\epsilon^0} R(\mathbf{X},t)J(\mathbf{x},t)\mathbf{A}(\mathbf{X},t)d\mathbf{X} = \int_{\partial B_\epsilon^t} \sigma \mathbf{n} ds(\mathbf{x}) + \int_{B_\epsilon^t} \mathbf{f}^{ext}(\mathbf{x},t)d\mathbf{x}. \tag{106}$$

Pushing forward the volume integral on the left side of Equation 106 results in both sides over B_ϵ^t:

$$\int_{B_\epsilon^t} \rho(\mathbf{x},t)\mathbf{a}(\mathbf{x},t)d\mathbf{x} = \int_{\partial B_\epsilon^t} \sigma \mathbf{n} ds(\mathbf{x}) + \int_{B_\epsilon^t} \mathbf{f}^{ext} d\mathbf{x} = \int_{B_\epsilon^t} \nabla^{\mathbf{x}} \cdot \sigma d\mathbf{x} + \int_{B_\epsilon^t} \mathbf{f}^{ext} d\mathbf{x}, \tag{107}$$

or

$$\rho \mathbf{a} = \nabla^{\mathbf{x}} \cdot \sigma + \mathbf{f}^{ext}, \quad \forall \mathbf{x} \in \Omega^t, t \geqslant 0. \tag{108}$$

Alternatively, we can also choose to pull back the right side of Equation 106 using

$$\int_{\partial B_\epsilon^t} \sigma(\mathbf{x},t)\mathbf{n} ds(\mathbf{x}) = \int_{\partial B_\epsilon^0} J(\mathbf{X},t)\sigma(\phi(\mathbf{X},t),t)\mathbf{F}^{-T}(\mathbf{X},t)\mathbf{N} ds(\mathbf{X}). \tag{109}$$

Recall the first Piola Kirchoff stress is related to the Cauchy stress as $\mathbf{P} = J\sigma\mathbf{F}^{-T}$, we get

$$\int_{\partial B_\epsilon^t} \sigma(\mathbf{x},t)\mathbf{n} ds(\mathbf{x}) = \int_{\partial B_\epsilon^0} \mathbf{P}(\mathbf{X},t)\mathbf{N} ds(\mathbf{X}) = \int_{B_\epsilon^0} \nabla^{\mathbf{x}} \cdot \mathbf{P}(\mathbf{X},t) d\mathbf{X}. \tag{110}$$

Compare this with Equation 106 we get the Lagrangian form of conservation of momentum:

$$\int_{B_\epsilon^0} R(\mathbf{X},0)\mathbf{A}(\mathbf{X},t)d\mathbf{X} = \int_{B_\epsilon^0} \nabla^{\mathbf{x}} \cdot \mathbf{P}(\mathbf{X},t)d\mathbf{X} + \int_{B_\epsilon^0} \mathbf{F}^{ext} J(\mathbf{X},t) d\mathbf{X} \tag{111}$$

or

$$R(\mathbf{X},0)\mathbf{A}(\mathbf{X},t) = \nabla^{\mathbf{x}} \cdot \mathbf{P}(\mathbf{X},t) + \mathbf{F}^{ext}(\mathbf{X},t)J(\mathbf{X},t), \quad \forall \mathbf{X} \in \Omega^0, t \geqslant 0 \tag{112}$$

where $\mathbf{F}^{ext}$ is the pull back of the Eulerian body force per unit volume $\mathbf{f}^{ext}$.

7.3 Weak Form

MPM is like the FEM discretization of the stress based forces over the Eulerian grid. It therefore uses the weak form of force balance. We can think of it as Lagrangian. Let's ignore the external force for simplicity. We can start with the conservation of momentum

$$R(\mathbf{X},0)\mathbf{A}(\mathbf{X},t) = \nabla^{\mathbf{x}} \cdot \mathbf{P}(\mathbf{X},t), \forall \mathbf{X}, t, \tag{113}$$

or

$$R_0 A_i = P_{ij,j}, \quad \forall \mathbf{X}, t, \tag{114}$$

where $R_0 = R(\mathbf{X}, 0)$. So for an arbitrary function $\mathbf{Q}(\cdot, t) : \Omega^0 \to \mathbb{R}^d$, let's compute the dot product to Equation 113 and integrate over Ω^0 to generate the weak form:

$$\int_{\Omega^0} Q_i(\mathbf{X}, t) R(\mathbf{X}, 0) A_i(\mathbf{X}, t) d\mathbf{X} = \int_{\Omega^0} Q_i(\mathbf{X}, t) P_{ij,j}(\mathbf{X}, t) d\mathbf{X} \tag{115}$$

Integration by parts

$$= \int_{\Omega^0} \left(\left(Q_i(\mathbf{X}, t) P_{ij}(\mathbf{X}, t) \right)_{,j} - Q_{i,j}(\mathbf{X}, t) P_{ij}(\mathbf{X}, t) \right) d\mathbf{X} \tag{116}$$

$$= \int_{\partial\Omega^0} Q_i(\mathbf{X}, t) P_{ij}(\mathbf{X}, t) N_j(\mathbf{X}, t) ds(\mathbf{X}) \tag{117}$$

Divergence theorem

$$- \int_{\Omega^0} Q_{i,j}(\mathbf{X}, t) P_{ij}(\mathbf{X}, t) d\mathbf{X}.$$

The quantity $P_{ij}N_j$ would be specified as a boundary condition. If we let $\mathbf{T}(\mathbf{X}, t)$ be the boundary force per unit reference area with $T_i = P_{ij}N_j$, then we can say that force balance implies that $\forall \mathbf{Q}(\cdot, t) : \Omega^0 \to \mathbb{R}^d$

$$\boxed{\int_{\Omega^0} Q_i(\mathbf{X}, t) R(\mathbf{X}, 0) A_i(\mathbf{X}, t) d\mathbf{X} = \int_{\partial\Omega^0} Q_i T_i ds(\mathbf{X}) - \int_{\Omega^0} Q_{i,j} P_{ij} d\mathbf{X}}. \tag{118}$$

For MPM, the stress derivatives will be discretized in the current configuration, so we can push the stress involving integrals to the Eulerian view. Let $\mathbf{q}$ be the push forward of $\mathbf{Q}$ with $\mathbf{Q}(\mathbf{X}, t) = \mathbf{q}(\phi(\mathbf{X}, t), t)$ and $\mathbf{q}(\mathbf{x}, t) = \mathbf{Q}(\phi^{-1}(\mathbf{x}, t), t)$, we have

$$Q_{i,j} = \frac{\partial Q_i}{\partial X_j} = \frac{\partial q_i}{\partial x_k} \frac{\partial x_k}{\partial X_j} = q_{i,k} F_{kj}. \tag{119}$$

Further more, recall that

$$\sigma_{ik} = \frac{1}{J} P_{ij} F_{kj},$$

and define $\mathbf{t}$ to be the external force per unit area in the Eulerian configuration ($\mathbf{t}$ is the push forward of $\mathbf{T}$), Equation 118 becomes

$$\boxed{\int_{\Omega^t} q_i(\mathbf{x}, t) \rho(\mathbf{x}, t) a_i(\mathbf{x}, t) d\mathbf{x} = \int_{\partial\Omega^t} q_i t_i ds(\mathbf{x}) - \int_{\Omega^t} q_{i,k} \sigma_{ik} d\mathbf{x}}, \tag{120}$$

!!!!

where we have pushed forward the volume and surface integrals using the rules described in Section 5.5. Force balance implies the above holds for an arbitrary $\mathbf{q}(\cdot, t) : \Omega^t \to \mathbb{R}^d$. We call Equation 120 the weak form of force balance in the Eulerian view. The MPM discretization on the grid will be based on this equation.

8 MATERIAL PARTICLES

Before discretizing the weak form of the force balance (Equation 120), let's first introduce the material particles (or material points, Lagrangian particles, etc.) that represents the material simulated with MPM.

Recall that the material point method is Lagrangian in the sense that we track actual particles of material. That is we keep track of mass (m_p), velocity (v_p) and position (x_p) for a collection of material particles p. However, all stress based forces are computed on the Eulerian grid, so we have to transfer the material state to the Eulerian configuration to incorporate the effects of material forces. Then, we transfer these effects back to the material particles and move them in the normal Lagrangian way. The Lagrangian nature makes advection very trivial compared to pure Eulerian methods (such as grid-based fluid simulation).

Just like any other PIC/FLIP solvers, MPM solves the governing equation on a background Eulerian grid. The grid acts as a scratch pad. It can be destroyed?? after each solve and reinitialized in the beginning of the next time step. In practice, it is easy to just use a fixed Cartesian grid.

8.1 Eulerian Interpolating Functions

In each time step of MPM, particles transfer their mass and momentum to the grid nodes. After the grid solve, velocities are transferred back to particles for them to perform the advection step. Both transfers require interpolation functions. Taking a Finite Element view, the Eulerian grid is the essential computational mesh while particles act as quadrature points. Therefore, instead of thinking of the interpolation function as a particle 'kernel' as in SPH, it is more natural (but not necessarily) to let the interpolation functions be defined over grid nodes.

We can denote the interpolation function at grid node $\mathbf{i}$ with $N_\mathbf{i}(\mathbf{x})$. Note that $\mathbf{i}$ is bold here because it is usually a multi-index for grid nodes. Specifically, $\mathbf{i} = (i, j)$ in 2D, $\mathbf{i} = (i, j, k)$ in 3D. When $N_\mathbf{i}(\mathbf{x})$ is evaluated at a particle location $\mathbf{x}_p$, a shorter notation $N_\mathbf{i}(\mathbf{x}_p) = w_{\mathbf{i}p}$ is often used instead. Intuitively, we are associating with each particle p and grid node $\mathbf{i}$ a weight $w_{\mathbf{i}p}$ which determines how strongly the particle and node interact. If the particle and grid node are close together, the weight should be large. If the particle and node are farther apart, the weight should be small.

We use dyadic products of one-dimensional interpolation functions as our grid basis functions as in [Steffen et al., 2008]

$$N_\mathbf{i}(\mathbf{x}_p) = N(\frac{1}{h}(x_p - x_i))N(\frac{1}{h}(y_p - y_i))N(\frac{1}{h}(z_p - z_i)), \tag{121}$$

where $\mathbf{i} = (i, j, k)$ is the grid index, $\mathbf{x}_p = (x_p, y_p, z_p)$ is the evaluation position, h is the grid spacing, $\mathbf{x}_i = (x_i, y_i, z_i)$ is the grid node position. For more compact notation, we will use $w_{\mathbf{i}p} = N_\mathbf{i}(\mathbf{x}_p)$ and $\nabla w_{\mathbf{i}p} = \nabla N_\mathbf{i}(\mathbf{x}_p)$. While writing them like these, one should keep in mind both $N_\mathbf{i}$ and $\nabla N_\mathbf{i}$ are still functions of arbitrary $\mathbf{x}$. When writing with w, it means these functions are evaluated at $\mathbf{x} = \mathbf{x}_p$.

Choosing a kernel N leads to trade offs with respect to smoothness, computational efficiency, and the width of the stencil. We prefer tensor product splines for their compu-

tational efficiency, as they are relatively inexpensive to compute, differentiate, and store. The multi-linear kernel typically employed for FLIP fluid solvers is the simplest of these options, but it is not suitable here. There are two reasons for this (see [Steffen et al., 2008]). The first is that ∇w_{ip} would be discontinuous and produce discontinuous forces. The second is that ∇w_{ip} may be far from zero when $w_{ip} \approx 0$, leading to large forces being applied to grid nodes with tiny weights. MPM typically requires C1 continuity of the interpolation function to prevent the so called cell-crossing instability. In practice, either quadratic B splines or cubic B splines may be used. Quadratic B splines is more computational efficient and memory saving due to a smaller transfer stencil. Cubic B splines on the other hand, is more expensive but provides wider coverage, therefore less sensitive to numerical errors such as numerical fracture when they are not desired artistic effects. The cubic kernel is defined with

$$N(x) = \begin{cases} \frac{1}{2}|x|^3 - |x|^2 + \frac{2}{3} & 0 \leqslant |x| < 1 \\ \frac{1}{6}(2-|x|)^3 & 1 \leqslant |x| < 2 \\ 0 & 2 \leqslant |x| \end{cases} \tag{122}$$

where h is the grid spacing. The quadratic kernel is also useful:

$$N(x) = \begin{cases} \frac{3}{4} - |x|^2 & 0 \leqslant |x| < \frac{1}{2} \\ \frac{1}{2}\left(\frac{3}{2} - |x|\right)^2 & \frac{1}{2} \leqslant |x| < \frac{3}{2} \\ 0 & \frac{3}{2} \leqslant |x| \end{cases} \tag{123}$$

The quadratic and cubic kernels are shown in Figure 3. Though theoretically unstable, linear interpolation functions may still work for certain applications in practice. It is

$$N(x) = \begin{cases} 1 - |x| & 0 \leqslant |x| < 1 \\ 0 & 1 \leqslant |x| \end{cases} \tag{124}$$

Computing the gradient is done similarly by differentiating the one dimensional functions:

$$\nabla N_i(\mathbf{x}_p) = \begin{pmatrix} \frac{1}{h}N'(\frac{1}{h}(x_p - x_i))N(\frac{1}{h}(y_p - y_i))N(\frac{1}{h}(z_p - z_i)) \\ N(\frac{1}{h}(x_p - x_i))\frac{1}{h}N'(\frac{1}{h}(y_p - y_i))N(\frac{1}{h}(z_p - z_i)) \\ N(\frac{1}{h}(x_p - x_i))N(\frac{1}{h}(y_p - y_i))\frac{1}{h}N'(\frac{1}{h}(z_p - z_i)) \end{pmatrix}$$

where $N'(x)$ is the derivative of $N(x)$.

8.2 Eulerian/Lagrangian Mass

When representing our material as a finite collection of points, we can assign each point a subset ($B^0_{\Delta x,p} \subset \Omega^0$) of the total material. In this way, we can define the mass of that particle to be

$$m_p^n = \int_{B^{t^n}_{\Delta x,p}} \rho(\mathbf{x}, t^n)\, d\mathbf{x}. \tag{125}$$

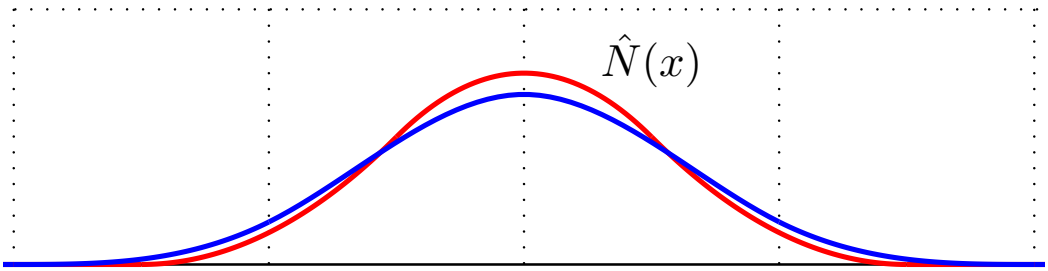


Figure 3: Cubic (blue) and quadratic (red) splines used for computing interpolation weights.

With this convention, we can define the following conservative process for transferring mass and momentum (and then velocity) to the nodes of the Eulerian grid. Define

$$m_i = \sum_p m_p N_i(\boldsymbol{x}_p) \tag{126}$$

as the mass of Eulerian grid node $\boldsymbol{i}$. With this convention, we have

$$\sum_i m_i = \sum_i \sum_p m_p N_i(\boldsymbol{x}_p) = \sum_p m_p \sum_i N_i(\boldsymbol{x}_p) = \sum_p m_p \tag{127}$$

by the partition of unity assumption on the N_i.

Conservation of mass is preserved with the construct of interpolation function.

8.3 Eulerian/Lagrangian Momentum

Similarly, we can transfer particle momentum $m_p \boldsymbol{v}_p$ as (note that we will eventually use a different transfer called APIC described in Section 10.1)

$$(m\boldsymbol{v})_i = \sum_p m_p \boldsymbol{v}_p N_i(\boldsymbol{x}_p) \tag{128}$$

and we can show that

$$\sum_i (m\boldsymbol{v})_i = \sum_p m_p \boldsymbol{v}_p \tag{129}$$

by the same partition of unity logic. The Eulerian velocity $\boldsymbol{v}_i$ is defined as

$$\boldsymbol{v}_i = \frac{(m\boldsymbol{v})_i}{m_i}. \tag{130}$$

Note that in this case, the repeated index does *not* imply summation.

8.4 Eulerian to Lagrangian Transfer

The transfer from Eulerian variables to Lagrangian variables is similar. However, we never need to transfer mass from the grid to the particles since Lagrangian particle mass never changes. Velocity is simply interpolated as

$$\boldsymbol{v}_p = \sum_i \boldsymbol{v}_i N_i(\boldsymbol{x}_p). \tag{131}$$

It is easily verified that this conserves momentum as

$$\sum_p m_p \boldsymbol{v}_p = \sum_p m_p \sum_i \boldsymbol{v}_i N_i(\boldsymbol{x}_p) = \sum_i \boldsymbol{v}_i \sum_p m_p N_i(\boldsymbol{x}_p) = \sum_i m_i \boldsymbol{v}_i. \tag{132}$$

Momentum is conserved with this G2P velocity transfer.

9 DISCRETIZATION

9.1 Discrete Time

We will start by assuming we are at time t^n. Recall that the weak forms of the force balance equation (Equation 118 and 120) imply

$$\int_{\Omega^0} Q_i R_0 A_i d\boldsymbol{X} = \int_{\partial\Omega^{t^n}} q_i t_i ds(\boldsymbol{x}) - \int_{\Omega^{t^n}} q_{i,k}\sigma_{ik} d\boldsymbol{x} \tag{133}$$

for all $\boldsymbol{q}(\boldsymbol{x}, t^n)$ (or $\boldsymbol{Q}(\boldsymbol{X}, t^n)$). We start by replacing the Lagrangian acceleration $A_i(\boldsymbol{X}, t^n)$ with $\frac{1}{\Delta t}\left(V_i^{n+1} - V_i^n\right)$. We can then push the left hand side forward from Ω^0 to Ω^{t^n} to obtain

$$\begin{aligned}&\frac{1}{\Delta t}\int_{\Omega^{t^n}} q_i(\boldsymbol{x}, t^n)\rho(\boldsymbol{x}, t^n)\left(v_i^{n+1}(\boldsymbol{x}) - v_i^n(\boldsymbol{x})\right) d\boldsymbol{x}\\ &= \int_{\partial\Omega^{t^n}} q_i(\boldsymbol{x}, t^n) t_i(\boldsymbol{x}, t^n) ds(\boldsymbol{x}) - \int_{\Omega^{t^n}} q_{i,k}(\boldsymbol{x}, t^n)\sigma_{ik}(\boldsymbol{x}, t^n) d\boldsymbol{x}.\end{aligned} \tag{134}$$

Note that with this notation $\boldsymbol{v}^n : \Omega^{t^n} \to \mathbb{R}^d$, $\boldsymbol{v}^{n+1} : \Omega^{t^n} \to \mathbb{R}^d$ (both of them are defined for $\boldsymbol{x} \in \Omega^{t^n}$). Therefore $v_i^{n+1}(\boldsymbol{x}) = V_i(\phi^{-1}(\boldsymbol{x}, t^n), t^{n+1})$ and $v_i^n(\boldsymbol{x}) = V_i(\phi^{-1}(\boldsymbol{x}, t^n), t^n)$.

We should keep in mind that all the i and k subscripts in Equation 134 are component index for dimensions. I.e., $i = 1, 2, 3$ for 3D and $i = 1, 2$ for 2D. When we are going discrete in space, we probably want to use indices like $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ to denote a grid node. To avoid confusion, from now on, we can require Greek letters like α, β, γ to denote the component index. This results in rewriting Equation 134 as

$$\begin{aligned}&\frac{1}{\Delta t}\int_{\Omega^{t^n}} q_\alpha(\boldsymbol{x}, t^n)\rho(\boldsymbol{x}, t^n)\left(v_\alpha^{n+1}(\boldsymbol{x}) - v_\alpha^n(\boldsymbol{x})\right) d\boldsymbol{x}\\ &= \int_{\partial\Omega^{t^n}} q_\alpha(\boldsymbol{x}, t^n) t_\alpha(\boldsymbol{x}, t^n) ds(\boldsymbol{x}) - \int_{\Omega^{t^n}} q_{\alpha,\beta}(\boldsymbol{x}, t^n)\sigma_{\alpha\beta}(\boldsymbol{x}, t^n) d\boldsymbol{x},\end{aligned} \tag{135}$$

where $\alpha, \beta = 1, 2, .., d$ and d is the problem dimension (2 or 3). Now $q_{i\alpha}$ means the α component of the vector quantity $\mathbf{q}$ that is stored at node i. Also, $q_\alpha(\mathbf{x}, t)$ means the α component of the field $\mathbf{q}(\mathbf{x}, t)$.

9.2 Discrete Space

We can do an FEM style discretization of the spatial terms by replacing q_α, v_α^n and v_α^{n+1} with grid based interpolants as

$$q_\alpha(\mathbf{x}, t^n) = \sum_i q_{i\alpha}(t^n) N_i(\mathbf{x}), \tag{136}$$

$$v_\alpha^n(\mathbf{x}) = \sum_j v_{j\alpha}^n N_j(\mathbf{x}), \tag{137}$$

$$v_\alpha^{n+1}(\mathbf{x}) = \sum_j v_{j\alpha}^{n+1} N_j(\mathbf{x}) \tag{138}$$

or

$$q_\alpha^n = q_{i\alpha}^n N_i, \quad v_\alpha^n = v_{j\alpha}^n N_j, \quad v_\alpha^{n+1} = v_{j\alpha}^{n+1} N_j \tag{139}$$

for short where summation is implied on the repeated index. Force balance (Equation 135) can then be viewed as

$$\begin{aligned} &\frac{1}{\Delta t}\int_{\Omega^{t^n}} q_{i\alpha}^n N_i(\mathbf{x}) \rho(\mathbf{x}, t^n) v_{j\alpha}^{n+1} N_j(\mathbf{x}) d\mathbf{x} - \frac{1}{\Delta t}\int_{\Omega^{t^n}} q_{i\alpha}^n N_i(\mathbf{x}) \rho(\mathbf{x}, t^n) v_{j\alpha}^n N_j(\mathbf{x}) d\mathbf{x} \\ &\qquad = \int_{\partial\Omega^{t^n}} q_{i\alpha}^n N_i(\mathbf{x}) t_\alpha(\mathbf{x}, t^n) ds(\mathbf{x}) - \int_{\Omega^{t^n}} q_{i\alpha}^n N_{i,\beta}(\mathbf{x}) \sigma_{\alpha\beta}(\mathbf{x}, t^n) d\mathbf{x}. \end{aligned} \tag{140}$$

for all $q_{i\alpha}$. This can also be expressed as

$$\begin{aligned} &q_{i\alpha}^n \delta_{\alpha\beta} \frac{m_{ij}^n}{\Delta t} v_{j\beta}^{n+1} - q_{i\alpha}^n \delta_{\alpha\beta} \frac{m_{ij}^n}{\Delta t} v_{j\beta}^n \\ &= \int_{\partial\Omega^{t^n}} q_{i\alpha}^n N_i t_\alpha ds(\mathbf{x}) - \int_{\Omega^{t^n}} q_{i\alpha}^n N_{i,\beta} \sigma_{\alpha\beta} d\mathbf{x}. \end{aligned} \tag{141}$$

where

$$m_{ij}^n = \int_{\Omega^{t^n}} N_i(\mathbf{x}) \rho(\mathbf{x}, t^n) N_j(\mathbf{x}) d\mathbf{x} \tag{142}$$

defines the mass matrix. We can pull it back to the material space to get

$$m_{ij}^n = \int_{\Omega^0} N_i(\mathbf{x}(\mathbf{X})) R(\mathbf{X}, 0) N_j(\mathbf{x}(\mathbf{X})) d\mathbf{X} \approx \sum_p m_p N_i(\mathbf{x}_p) N_j(\mathbf{x}_p). \tag{143}$$

It is symmetric positive semi-definite because it can be written as $\mathbf{B}\mathbf{B}^T$ ($m_{ij} = \sum_p B_{ip} B_{jp}$) where $B_{ip} = \sqrt{m_p} N_{ip}$ so that $z^T \mathbf{M} z \geqslant 0$ for any z. In practice we can not use this

mass matrix because it may be singular. We'll see below a "mass lumping" strategy to approximate the mass matrix with a diagonal and positive definite matrix while keeping consistent with the particle-grid transfers.

Equation 141 must be true for all choices of $q^n_{i\alpha}$. So if we choose them to be

$$q^n_{i\alpha} = \begin{cases} 1, & \alpha = \hat{\alpha} \text{ and } i = \hat{i} \\ 0, & \text{otherwise} \end{cases}$$

then

$$\sum_j \frac{m_{\hat{i}j}}{\Delta t}\left(v^{n+1}_{j\hat{\alpha}} - v^n_{j\hat{\alpha}}\right) = \int_{\partial\Omega^{t^n}} N_{\hat{i}} t_{\hat{\alpha}} ds(x) - \int_{\Omega^{t^n}} N_{\hat{i},\beta}\sigma_{\hat{\alpha}\beta}\, dx. \tag{144}$$

This can be seen as a discrete force balance equation for the $\hat{i}, \hat{\alpha}$ degree of freedom on the grid. We will next show that it can be used as an explicit update rule for the Eulerian momentum and that the right hand side can be thought of as the $\hat{\alpha}^{th}$ component of the force on the $\hat{i}^{th}$ Eulerian grid node.

As we just mentioned, it is often convenient (though less accurate) to use a mass lumping simplification. It is done by replacing the rows in m_{ij} with the corresponding row sum, thus making it a diagonal matrix. The row sums (let's call it $\hat{m}_i$ for row i) are

$$\begin{aligned} \hat{m}_i &= \sum_j \int_{\Omega^{t^n}} N_i(x)\rho(x,t^n)N_j(x)dx = \int_{\Omega^{t^n}} N_i(x)\rho(x,t^n)\sum_j N_j(x)dx \\ &= \int_{\Omega^{t^n}} N_i(x)\rho(x,t^n)(x)dx. \end{aligned} \tag{145}$$

We can also use the following approximation (similarly to what we did in Equation 143)

$$\int_{\Omega^{t^n}} N_i(x)\rho(x,t^n)dx = \int_{\Omega^0} N_i(x(X))R(X,0)dX \approx \sum_p N_i(x_p)m_p \tag{146}$$

since $m_p \approx V^0_p R(X_p, 0)$. In other words $\hat{m}_i$ can naturally be approximated as m_i.

Rewriting $m_i v^n_i$ as the momentum $(mv)^n_i$, we can summarize the discretization as

$$\boxed{\frac{\left((mv)^{n+1}_{i\alpha} - (mv)^n_{i\alpha}\right)}{\Delta t} = \int_{\partial\Omega^{t^n}} N_i(x)t_\alpha(x,t^n)ds(x) - \int_{\Omega^{t^n}} N_{i,\beta}(x)\sigma_{\alpha\beta}(x,t^n)dx}. \tag{147}$$

In other words, since the left hand side is the change in momentum, the right hand side is approximately the force. Here we have assumed that we have an estimate of the stress $\sigma^n_p \approx \sigma(x^n_p, t^n)$ at each Lagrangian particle x^n_p, thus

$$\int_{\Omega^{t^n}} N_{i,\beta}(x)\sigma_{\alpha\beta}(x,t^n)dx \approx \sum_p \sigma^n_{p\alpha\beta} N_{i,\beta}(x^n_p)V^n_p \tag{148}$$

Discretization over volume of particles.

where V^n_p is the volume of $B^{t^n}_{\Delta x,p}$, or the volume particle p occupies at time t^n.

9.3 Estimating the Volume

We can estimate V_p^n with two ways. First, recall that

$$m_p \approx R(\mathbf{X}_p, 0)V_p^0 \approx \rho(\mathbf{x}_p^n, t^n)V_p^n. \tag{149}$$

We can then approximate $\rho(\mathbf{x}_p^n, t^n)$ from m_i^n as

$$\rho_i^n = \frac{m_i^n}{\Delta x^d}, \tag{150}$$

$$\rho(\mathbf{x}_p^n, t^n) \approx \sum_i \rho_i^n N_i(\mathbf{x}_p^n). \tag{151}$$

Therefore

$$V_p^n \approx \frac{m_p}{\sum_i \frac{m_i^n}{\Delta x^d} N_i(\mathbf{x}_p^n)} = \frac{m_p \Delta x^d}{\sum_i m_i^n N_i(\mathbf{x}_p^n)}. \tag{152}$$

Another approach is given as follows. If we have an approximation of the deformation at each Lagrangian particle: $\mathbf{F}_p^n \approx \mathbf{F}(\mathbf{X}_p, t^n)$ (which we will usually need for an elastic material), then you can approximate V_p^n using $J_p^n = \det(\mathbf{F}_p^n)$. Specifically, if we assume we know the initial volume $V_p^0 = \int_{B^0_{\Delta x,p}} d\mathbf{x}$, then

$$V_p^n \approx V_p^0 J_p^n. \tag{153}$$

If we use the formula for the first Piola Kirchoff stress $\sigma = \frac{1}{J}\mathbf{P}\mathbf{F}^T$ then we can further rewrite Equation 147 and 148 in terms of $\mathbf{P}$ instead of σ:

$$\sum_p {\sigma_p}_{\alpha\beta}^n N_{i,\beta}(\mathbf{x}_p^n) V_p^n = \sum_p \frac{1}{J_p^n} {P_p}_{\alpha\gamma}^n {F_p}_{\beta\gamma}^n N_{i,\beta}(\mathbf{x}_p^n) V_p^0 J_p^n = \sum_p {P_p}_{\alpha\gamma}^n {F_p}_{\beta\gamma}^n N_{i,\beta}(\mathbf{x}_p^n) V_p^0. \tag{154}$$

This is the formula that will be more useful in practice because most of our constitutive models are expressed in terms of $\mathbf{P}$. Now,

$$\boxed{\frac{\left((mv)_{i\alpha}^{n+1} - (mv)_{i\alpha}^n\right)}{\Delta t} = \int_{\partial\Omega^{t^n}} N_i(\mathbf{x}) t_\alpha(\mathbf{x}, t^n) ds(\mathbf{x}) - \sum_p {P_p}_{\alpha\gamma}^n {F_p}_{\beta\gamma}^n N_{i,\beta}(\mathbf{x}_p^n) V_p^0}. \tag{155}$$

We will next discuss how to approximate the deformation gradient ($\mathbf{F}_p^n$) at each Lagrangian point ($\mathbf{x}_p^n$).

9.4 Deformation Gradient Evolution

In order to continue the discussion of time stepping schemes, we need to consider the constitutive model. So far, the results hold independent of the type of material. We

will generally consider models where the stress depends primarily on the change of shape in the material as expressed via the deformation gradient (this includes all models discussed in Section 6). Next, we will discuss how this can be computed in the MPM context. The deformation gradient is more difficult to compute for Eulerian methods (or Lagrangian methods where there is no mesh, e.g. particle methods). However, we can use the equation (recall the result from Section 5.4)

$$\frac{\partial}{\partial t}\mathbf{F}(\mathbf{X},t) = \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{x}}(\boldsymbol{\phi}(\mathbf{X},t),t)\mathbf{F}(\mathbf{X},t) \tag{156}$$

to update (or evolve) a deformation gradient on each material particle by discretizing $\frac{\partial v}{\partial x}$ over the Eulerian grid.

Now let's reformulate a similar rule discretely. We assume we have the velocity (defined over Ω^{t^n}) at time t^{n+1} as a a function of $\boldsymbol{x}$ only. Specifically, we consider the function $\boldsymbol{v}^{n+1} : \Omega^{t^n} \to \mathbb{R}^d$ defined as $\boldsymbol{v}^{n+1}(\boldsymbol{x}) = \boldsymbol{V}(\phi^{-1}(\boldsymbol{x}, t^n), t^{n+1})$ and also of course $\boldsymbol{v}^{n+1}(\phi(\mathbf{X}, t^n)) = \boldsymbol{V}(\mathbf{X}, t^{n+1})$. (We have used the same definition in Section 9.1.) With this we have

$$\frac{\partial}{\partial t}\mathbf{F}(\mathbf{X},t^{n+1}) = \frac{\partial \boldsymbol{V}}{\partial \mathbf{X}}(\mathbf{X},t^{n+1}) = \frac{\partial \boldsymbol{v}^{n+1}}{\partial \boldsymbol{x}}(\phi(\mathbf{X},t^n))\mathbf{F}(\mathbf{X},t^n). \tag{157}$$

This is useful because if we further use the approximation

$$\frac{\partial}{\partial t}\mathbf{F}(\mathbf{X}_p,t^{n+1}) \approx \frac{\mathbf{F}_p^{n+1}-\mathbf{F}_p^n}{\Delta t} \tag{158}$$

then

$$\mathbf{F}_p^{n+1} = \mathbf{F}_p^n + \Delta t \frac{\partial \boldsymbol{v}^{n+1}}{\partial \boldsymbol{x}}(\boldsymbol{x}_p^n)\mathbf{F}_p^n = \left(\mathbf{I} + \Delta t \frac{\partial \boldsymbol{v}^{n+1}}{\partial \boldsymbol{x}}(\boldsymbol{x}_p^n)\right)\mathbf{F}_p^n \tag{159}$$

and of course if we use the grid based interpolation formula for $\boldsymbol{v}^{n+1}$, i.e.,

$$\boldsymbol{v}^{n+1}(\boldsymbol{x}) = \sum_i \boldsymbol{v}_i^{n+1} N_i(\boldsymbol{x}), \tag{160}$$

$$\frac{\partial \boldsymbol{v}^{n+1}}{\partial \boldsymbol{x}}(\boldsymbol{x}) = \sum_i \boldsymbol{v}_i^{n+1}\left(\frac{\partial N_i}{\partial \boldsymbol{x}}(\boldsymbol{x})\right)^T, \tag{161}$$

then we have

$$\boxed{\mathbf{F}_p^{n+1} = \left(\mathbf{I} + \Delta t \sum_i \boldsymbol{v}_i^{n+1}\left(\frac{\partial N_i}{\partial \boldsymbol{x}}(\boldsymbol{x}_p^n)\right)^T\right)\mathbf{F}_p^n} \tag{162}$$

as the update rule for $\mathbf{F}_p^{n+1}$ given the $\boldsymbol{v}_i^{n+1}$ and $\mathbf{F}_p^n$.

9.5 Forces as Energy Gradient

The elastic response can be shown to arise from an elastic potential energy. This is true for both the continuous and discrete cases. For the discrete case, consider $\hat{\boldsymbol{x}}_i$ defined to

be the nodes of the Eulerian grid when $\hat{x}_i = x_i$. In other words, consider temporarily that we have allowed the nodes of the Eulerian grid to move with a variable defined as $\hat{x}_i$ denoting the imaginary moved node position. We can then temporarily think of these $\hat{x}_i$ as new Lagrangian degrees of freedom. It is like saying that we switched from having the X_p as our Lagrangian particles and switched over to $X_i = \phi^{-1}(x_i, t^n)$ as our new Lagrangian particles and with this we say that $x_i^{n+1} = \phi(X_i, t^{n+1}) = \hat{x}_i$ defines the new configuration of our material. If we keep this idea in mind, we can show that the forces derived in Section 9.2 are related to a discrete potential energy in the case of a certain class of elastic materials. For hyperelastic materials where the first Piola Kirchoff stress is related to an elastic potential energy density ψ via

$$P(F) = \frac{\partial \psi}{\partial F}(F) \tag{163}$$

as we saw in Section 6, we can define the total potential energy (as a function of the moved node positions $\hat{x}$ as

$$e(\hat{x}) = \sum_p \psi(F_p(\hat{x}))V_p^0 \tag{164}$$

where $\hat{x}$ is the full (assembled) vector of all Eulerian $\hat{x}_i$. Here we think of the deformation gradient as a function of $\hat{x}$. This is because we are temporarily thinking of the motion of the material as defined in terms of the $X_i = \phi^{-1}(x_i, t^n)$ and $\hat{x}_i$. Really, this is an equivalent notion to letting motion be defined via the v_i^{n+1}. Specifically we can think of them defined as

$$v_i^{n+1} = \frac{\hat{x}_i - x_i}{\Delta t} \quad \text{or} \quad \hat{x}_i = x_i + \Delta t v_i^{n+1}. \tag{165}$$

This leads to the particle wise deformation gradient formula of

$$F_{p\beta\gamma}(\hat{x}) = F_{p\beta\gamma}^n + \Delta t \sum_j \left(\frac{\hat{x}_{j\beta} - x_{j\beta}}{\Delta t} \right) \sum_\tau N_{j,\tau}(x_p^n) F_{p\tau\gamma}^n. \tag{166}$$

If we differentiate the energy with respect to $\hat{x}_{i\alpha}$ (the α^{th} component of $\hat{x}_i$) we get

$$\frac{\partial e}{\partial \hat{x}_{i\alpha}}(\hat{x}) = \sum_p \sum_{\beta,\gamma} P_{\beta\gamma}(F_p(\hat{x})) \frac{\partial F_{p\beta\gamma}}{\partial \hat{x}_{i\alpha}}(\hat{x}) V_p^0. \tag{167}$$

From the formula for the deformation gradient above, we can see that

$$\frac{\partial F_{p\beta\gamma}}{\partial \hat{x}_{i\alpha}}(\hat{x}) = \delta_{\alpha\beta} \sum_\tau N_{i,\tau}(x_p^n) F_{p\tau\gamma}^n \tag{168}$$

and if we plug this into Equation 167 we get

$$\frac{\partial e}{\partial \hat{x}_{i\alpha}}(\hat{x}) = \sum_p P_{\alpha\gamma}(F_p(\hat{x})) F_{p\tau\gamma}^n N_{i,\tau}(x_p^n) V_p^0. \tag{169}$$

Therefore, comparing with the result in Section 9.3, we can see that the force on the Eulerian grid node $\mathbf{i}$ is

$$-\frac{\partial e}{\partial \hat{x}_{i\alpha}}\left(\hat{\mathbf{x}} = \begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \vdots \end{pmatrix}\right). \tag{170}$$

In other words, we can write the momentum update as

$$\boxed{(mv)_{i\alpha}^{n+1} = (mv)_{i\alpha}^{n} - \Delta t \frac{\partial e}{\partial \hat{x}_{i\alpha}}\left(\hat{\mathbf{x}} = \begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \vdots \end{pmatrix}\right) + \Delta t \int_{\partial\Omega^{t^n}} N_i(\mathbf{x}) t_\alpha(\mathbf{x}, t^n) ds(\mathbf{x})}. \tag{171}$$

10 EXPLICIT TIME INTEGRATION

In this section we describe the easiest implementation of MPM with Symplectic Euler time integration. Most of the formulas in this section are derived in previous sessions. They are restated here to provide a more concise and practitioner friendly reference for implementing MPM. We first discuss some separate modules and then state the full MPM algorithm.

10.1 APIC Transfers

The Affine Particle-In-Cell (APIC) method [Jiang et al., 2015] is recommended for particle-grid transfers due to its nice numerical properties. We ignore the time superscripts here for notation simplicity.

The transfer from particles to grid is

$$m_i = \sum_p w_{ip} m_p, \tag{172}$$

$$m_i v_i = \sum_p w_{ip} m_p (v_p + \mathbf{B}_p (\mathbf{D}_p)^{-1} (\mathbf{x}_i - \mathbf{x}_p)), \tag{173}$$

where $\mathbf{B}_p$ is a matrix quantity stored at each particle (just like mass, position and velocity), $\mathbf{D}_p$ is given by

$$\mathbf{D}_p = \sum_i w_{ip} (\mathbf{x}_i - \mathbf{x}_p)(\mathbf{x}_i - \mathbf{x}_p)^T \tag{174}$$

and is derived by preserving affine motion during the transfers. The corresponding transfer from the grid back to particles is

$$\mathbf{v}_p = \sum_i w_{ip} \mathbf{v}_i, \tag{175}$$

$$\mathbf{B}_p = \sum_i w_{ip} \mathbf{v}_i (\mathbf{x}_i - \mathbf{x}_p)^T. \tag{176}$$

Conveniently, $\mathbf{D}_p$ takes on a surprisingly simple form in the case of the quadratic ($\mathbf{D}_p = \frac{1}{4}\Delta x^2 \mathbf{I}$) and cubic ($\mathbf{D}_p = \frac{1}{3}\Delta x^2 \mathbf{I}$) interpolation stencils commonly used for MPM. Note that for these interpolating stencils, multiplying by $(\mathbf{D}_p)^{-1}$ amounts to a constant scaling factor.

If (though not recommended) linear spline is used for the interpolation function, to prevent singular $\mathbf{D}_p$, an alternative formulation of APIC particle to grid is given as

$$m_i = \sum_p w_{ip} m_p, \tag{177}$$

$$m_i \mathbf{v}_i = \sum_p w_{ip} m_p (\mathbf{v}_p + \mathbf{C}_p (\mathbf{x}_i - \mathbf{x}_p)), \tag{178}$$

and the grid to particle transfer is

$$\mathbf{v}_p = \sum_i w_{ip} \mathbf{v}_i, \tag{179}$$

$$\mathbf{C}_p = \sum_i \mathbf{v}_i \left(\frac{\partial N_i}{\partial \mathbf{x}} (\mathbf{x}_p) \right)^T. \tag{180}$$

Note that in this case, $\mathbf{C}_p$ is just the Eulerian velocity gradient evaluated at $\mathbf{x}_p$, which we have already computed for evolving $\mathbf{F}_p$.

10.2 Deformation Gradient Update

In the beginning of each time step, the grid node locations are always on a undeformed regular grid. Let's call them $\mathbf{x}_i^n$. The sup-script n is not necessary if we use the same grid all the time but we'll keep it here for notation convention. The deformation gradient of each particle can be updated following the motion of the grid. If we assume a grid velocity field $\mathbf{v}_i^{n+1}$, it can be shown that the update rule for $\mathbf{F}$ is

$$\mathbf{F}_p^{n+1} = \left(\mathbf{I} + \Delta t \sum_i \mathbf{v}_i^{n+1} (\nabla w_{ip}^n)^T \right) \mathbf{F}_p^n. \tag{181}$$

10.3 State Update

Grid nodes move??
No.
The move is only imaginary.
We only need the velocity.

Symplectic Euler time integration on the grid implies

$$x_i^{n+1} = x_i^n + \Delta t v_i^{n+1}, \tag{182}$$
$$v_i^{n+1} = v_i^n + \Delta t f_i(x_i^n)/m_i. \tag{183}$$

One way of thinking about the grid motion is

$$x = x(v), \tag{184}$$

i.e. the grid node positions are simply a function of grid node velocities via Equation 182. Therefore

$$f_i^n = f_i(x_i^n) = f_i(x(0)), \tag{185}$$

i.e., the explicit force is simply the force assuming zero grid motion. Using Equation 182, we can rewrite Equation 181 as

$$F_p(x) = \left(I + \sum_i (x_i - x_i^n)(\nabla w_{ip}^n)^T \right) F_p^n, \tag{186}$$

so that F_p is a function of grid node locations x.

10.4 Forces

MPM Forces are defined on grid nodes. As we've seen earlier, for hyperelastic solids, the force can be derived either from the weak form of momentum equation or as the gradient of total potential energy. The later is simpler and often used as long as there is a well defined potential energy. Here we focus on hyperelastic solids.

As mentioned before, we assume a deformation gradient based hyperelastic energy density. The total elastic potential energy is then

$$e = \sum_p V_p^0 \Psi_p(F_p), \tag{187}$$

where V_p^0 is the material space volume of particle p.

Nodal elastic force is the negative gradient of the total potential energy evaluated at nodal positions. Using Equation 186, the MPM spatial discretization of the stress-based forces is given as

$$f_i(x) = -\frac{\partial e}{\partial x_i}(x) = -\sum_p V_p^0 \left(\frac{\partial \Psi_p}{\partial F}(F_p(x)) \right) (F_p^n)^T \nabla w_{ip}^n. \tag{188}$$

This is the force acting on grid node i resulting from elastic stresses of its nearby particles. In the case of Symplectic Euler, the force is easily written as

$$\mathbf{f}_i^n = \mathbf{f}_i(\mathbf{x}_i^n) = -\sum_p V_p^0 \left(\frac{\partial \Psi_p}{\partial \mathbf{F}}(\mathbf{F}_p^n) \right) (\mathbf{F}_p^n)^T \nabla w_{ip}^n, \tag{189}$$

which fully depends on the existing particle/grid weights and particle attributes. Alternatively if we don't have the energy density, the force can also be written using the Cauchy stress,

$$\mathbf{f}_i^n = \mathbf{f}_i(\mathbf{x}_i^n) = -\sum_p V_p^n \sigma_p^n \nabla w_{ip}^n, \tag{190}$$

this comes from the weak form.

10.5 MPM Scheme: Full Algorithm

Here we outline the full update scheme for Symplectic Euler MPM. The particles are assumed to have already been initialized (i.e. each particle has a mass m_p, volume V_p^0, initial position $\mathbf{x}_p$, initial velocity $\mathbf{v}_p$, affine matrix $\mathbf{B}_p$ and their material-related parameters (such as μ and λ in the case of Neo-Hookean).

1. **Particle to grid transfer (P2G)**. Using the APIC formula from Section 10.1, the first step is to transfer particle quantities to the grid. In particular, this step computes grid mass and momentum.
2. **Compute grid velocities**. $\mathbf{v}_i = \frac{m_i \mathbf{v}_i}{m_i}$. For nodes with mass equals to 0, m_i and $\mathbf{v}_i$ are manually reset to $\mathbf{0}$. One would traditionally think this requires a floating point threshold when implementing this step. However in practice directly comparing against `0.0f` will not produce any unstable behavior or overflow. This is because even though you may get large forces on small mass nodes, the effect is scaled by the weight again when transferrring back to particles. In fact, hard coding a threshold will cause various problems such as drifting of total momentum.
3. **Identify grid degree of freedoms**. This step is important for implementation efficiency. We label the grid nodes with nonzero masses to be the actually degree of freedoms. All other nodes will remain static and are not considered being part of the solver unknowns.
4. **Compute explicit grid forces** $\mathbf{f}_i^n$ using Equation 189.
5. **Grid velocity update** using Equation 183. This step should take the boundary conditions or collision objects into account. In the case of explicit integration, each nodal velocity can be independently set to the desired value due to Dirichlet boundary conditions or rigid object collisions. See Section 12.1 for more details on how to deal with collision objects.

6. **Update particle deformation gradient** using Equation 181. Note that we never actually moved the grid or computed any new grid $\mathbf{x}$. The motion is imaginary and only velocities are explicitly stored.
7. **Grid to particle transfer (G2P)**. This step computes new particle velocities $\mathbf{v}_p^{n+1}$ and affine matrices $\mathbf{B}_p^{n+1}$ with the scheme given in Section 10.1.
8. **Particle advection**. Finally particles are advected with their new velocities: $\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1}$. Note that this is only true when APIC is used. In a FLIP or FLIP-PIC-blending transfer scheme, $\mathbf{x}_p^{n+1} = \sum_i \mathbf{x}_i^{n+1} w_{ip}$ should be used instead (note that these two formulas are equivalent in the PIC or APIC case).

11 IMPLICIT TIME INTEGRATION

For implicit time integration, the main difference from explicit is the grid velocity update step in the MPM algorithm.

11.1 Force Derivative

Recall that the total elastic potential energy can be expressed in terms of the energy density Ψ as

$$\int_{\Omega^0} \Psi(\mathbf{F}(\mathbf{X}))\,d\mathbf{X}, \tag{191}$$

where Ω^0 is the undeformed configuration of the material. The MPM spatial discretization of the stress-based forces is equivalent to differentiation of a discrete approximation of this energy with respect to the Eulerian grid node material positions. However, we do not actually deform the Eulerian grid so we can think of the change in the grid node locations as being determined by the grid node velocities. That is, if $\mathbf{x}_i$ is the position of grid node i, then $\hat{\mathbf{x}}_i = \mathbf{x}_i + \Delta t \mathbf{v}_i$ would be the deformed location of that grid node given the current velocity $\mathbf{v}_i$ of the node. If we refer to the vector of all grid nodes $\hat{\mathbf{x}}_i$ as $\hat{\mathbf{x}}$, then the MPM approximation to the total elastic potential can be written as

pseudo move

$$e(\hat{\mathbf{x}}) = \sum_p V_p^0 \Psi(\hat{\mathbf{F}}_p(\hat{\mathbf{x}})), \tag{192}$$

where V_p^0 is the volume of material originally occupied by particle p, and $\hat{\mathbf{F}}_{Ep}$ is related to $\hat{\mathbf{x}}$ as

$$\hat{\mathbf{F}}_p(\hat{\mathbf{x}}) = \left(\mathbf{I} + \sum_i (\hat{\mathbf{x}}_i - \mathbf{x}_i)(\nabla w_{ip}^n)^T\right)\mathbf{F}_p^n. \tag{193}$$

With this convention, the MPM spatial discretization of the stress-based forces is given as

$$-\mathbf{f}_i(\hat{\mathbf{x}}) = \frac{\partial e}{\partial \hat{\mathbf{x}}_i}(\hat{\mathbf{x}}) = \sum_p V_p^0 \frac{\partial \Psi}{\partial \mathbf{F}}(\hat{\mathbf{F}}_p(\hat{\mathbf{x}}))(\mathbf{F}_p^n)^T \nabla w_{ip}^n. \tag{194}$$

That is, $\mathbf{f}_i(\hat{\mathbf{x}})$ is the force on grid node i resulting from elastic stresses. This can also be written in terms of the Cauchy stress $\boldsymbol{\sigma}_p = \frac{1}{J_p^n}\frac{\partial \Psi}{\partial \mathbf{F}}(\hat{\mathbf{F}}_p(\hat{\mathbf{x}}))(\mathbf{F}_p^n)^T$ as

$$\mathbf{f}_i(\hat{\mathbf{x}}) = -\sum_p V_p^n \boldsymbol{\sigma}_p \nabla w_{ip}^n, \tag{195}$$

where $V_p^n = J_p^n V_p^0$ is the volume of the material occupied by particle p at time t^n.

We highlight this relation of the MPM spatial discretization to the elastic potential because we would like to evolve our grid velocities $\mathbf{v}_i$ implicitly in time. With this convention, we can take an implicit step on the elastic part of the update by utilizing the Hessian of the potential with respect to $\hat{\mathbf{x}}$. The action of this Hessian on an arbitrary increment $\delta\mathbf{u}$ can be expressed as

$$-\delta \mathbf{f}_i = \sum_j \frac{\partial^2 e}{\partial \hat{\mathbf{x}}_i \partial \hat{\mathbf{x}}_j}(\hat{\mathbf{x}})\delta \mathbf{u}_j = \sum_p V_p^0 \mathbf{A}_p (\mathbf{F}_p^n)^T \nabla w_{ip}^n, \tag{196}$$

where

4th order

2nd order $$\mathbf{A}_p = \frac{\partial^2 \Psi}{\partial \mathbf{F} \partial \mathbf{F}}(\hat{\mathbf{F}_p}(\hat{\mathbf{x}})) : \left(\sum_j \delta \mathbf{u}_j (\nabla w_{jp}^n)^T \mathbf{F}_p^n \right)$$ 2nd order tensor (197)

and the notation $\mathbf{A} = \mathcal{C} : \mathbf{D}$ is taken to mean $A_{ij} = \mathcal{C}_{ijkl} D_{kl}$ with summation implied on indices kl.

We can also derive this with the index notation. Recall the force is already written as a function of $\mathbf{x}$ in Equation 188, differentiating with implicitly summed index notations gives

$$\frac{\partial e}{\partial x_{i\alpha} \partial x_{j\tau}} = -\frac{\partial f_{i\alpha}}{\partial x_{j\tau}} = \sum_p V_p^0 \frac{\partial^2 \Psi}{\partial F_{\alpha\beta} \partial F_{\tau\sigma}} (\nabla w_{jp}^n)_\omega (\nabla w_{ip}^n)_\gamma (F_p^n)_{\omega\sigma} (F_p^n)_{\gamma\beta}, \tag{198}$$

where $\frac{\partial^2 \Psi}{\partial \mathbf{F}^2}$ can be derived from $\Psi(\mathbf{F})$.

11.2 Backward Euler System

Backward Euler time integration replaces $\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \Delta t \mathbf{f}_i(\mathbf{x}_i^n)/m_i$ (Equation 183) with

$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \Delta t \mathbf{f}_i(\mathbf{x}_i^{n+1})/m_i, \tag{199}$$

For explicit scheme, force depends on particle stress and weights.

i.e. the force is implicitly dependent on grid motion. This allows for larger time steps and more stable behaviors.

Rearranging the equations of motion reveals a nonlinear system of equations of grid node velocities:

$$h(v^{n+1}) = Mv^{n+1} - \Delta t f(x^n + \Delta t v^{n+1}) - Mv^n = 0. \tag{200}$$

11.3 Newton's Method

Solving Equation 200 can be done with a traditional Newton-Raphson solver. One starts with an initial guess $v^{(0)}$, such as zero or the previous velocity v^n. Then the solution is iteratively improved with

2nd order tensor

$$v^{(i+1)} = v^{(i)} - \left(\frac{\partial h}{\partial v}(v^{(i)})\right)^{-1} h(v^{(i)}), \tag{201}$$

where each step the linear system is solved with a Krylov solver such as MINRES. Note that in each Newton iteration, F_p needs to be updated according to the velocity, an old state F_p^n needs to be stored for each new F_p evaluation as well as the final F_p update after the Newton solve.

In many cases only one iteration of Newton step is taken to already allow time steps orders of magnitude higher than explicit methods. This is almost equivalent to the so called semi-implicit MPM integration.

11.4 Linearized Force

We think of the elasto-plastic response as defined from the material positions of the Eulerian grid nodes $\hat{x}_i = x_i + \Delta t v_i$. However, as noted before, we never deform this grid. Therefore, we can think of $\hat{x} = \hat{x}(v)$ as defined by v. With this in mind, we use the following notation $f_i^n = f_i(\hat{x}(0))$, $f_i^{n+1} = f_i(\hat{x}(v^{n+1}))$ and $\frac{\partial^2 e^n}{\partial \hat{x}_i \partial \hat{x}_j} = -\frac{\partial f_i^n}{\partial \hat{x}_j} = -\frac{\partial f_i}{\partial \hat{x}_j}(\hat{x}(0))$.

Using these derivatives, we form our implicit update using $v_i^{n+1} = v_i^n + \Delta t m_i^{-1}((1-\beta)f_i^n + \beta f_i^{n+1}) \approx v_i^n + \Delta t m_i^{-1}(f_i^n + \beta \Delta t \sum_j \frac{\partial f_i^n}{\partial \hat{x}_j} v_j^{n+1})$. This leads to a (mass) symmetric system to solve for v_i^{n+1}

$$\sum_j \left(I\delta_{ij} + \beta \Delta t^2 m_i^{-1} \frac{\partial^2 e^n}{\partial \hat{x}_i \partial \hat{x}_j}\right) v_j^{n+1} = v_i^\star, \tag{202}$$

where the right hand side is

$$v_i^\star = v_i^n + \Delta t m_i^{-1} f_i^n \tag{203}$$

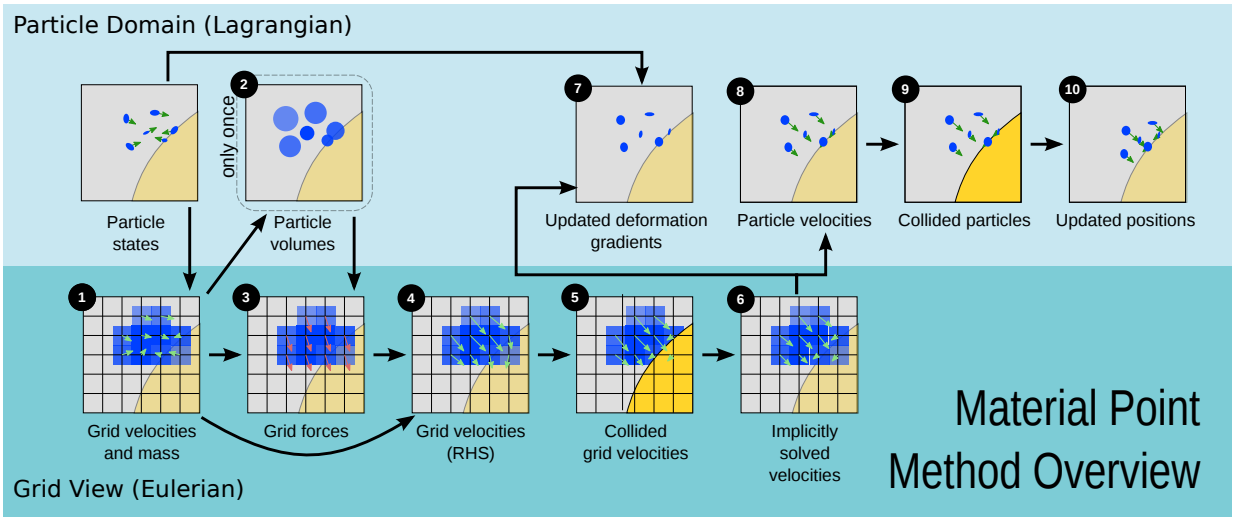


Figure 4: An overview of the implicit time integration MPM scheme.

and β chooses between explicit ($\beta = 0$), trapezoidal ($\beta = \frac{1}{2}$), and backward Euler ($\beta = 1$). It can be shown this scheme is equivalent to taking one step of Newton's method with a zero velocity initial guess. It requires solving one linear system in each time step. An overview of the semi-implicit time integration MPM scheme is shown in Figure 4.

11.5 Optimization based Integrator

Further enlarge the time step!

The traditional Newton-Raphson solver leads to a significant time step improvement over more standard explicit treatments, but still requires a small time step in practice to remain stable.

In fact, when the forces are derived from a potential energy function (which is true for most cases), it is possible to let Newton's method converge reliably under arbitrary Δt on the order of CFL condition by recasting the equation solving problem as an optimization problem, for which robust and efficient methods exist.

We will show that solving

$$h(v^{n+1}) = Mv^{n+1} - \Delta t f(x^n + \Delta t v^{n+1}) - Mv^n = 0 \tag{204}$$

is equivalent to minimizing the following objective function:

$$E(v_i) = \sum_i \frac{1}{2} m_i \|v_i - v_i^n\|^2 + e(x_i^n + \Delta t v_i). \tag{205}$$

Using this formulating, it is possible to take significantly larger time steps, providing a significant computational savings with minimal modification to the original approach. We refer to [Gast et al., 2015] for more details on implementing this integrator.

12 MORE TOPICS

12.1 Collision Objects

We process collisions against collision on the grid velocity v_i immediately after forces are applied to grid velocities. In the case of semi-implicit integration, this contributes to the right hand side of the linear system, and degrees of freedom corresponding to the colliding grid nodes are projected out during the solve. The collision treatments can be applied once more to particle velocities v_p^{n+1} just before updating positions to account for the minor discrepancies between particle and grid velocities due to interpolation. In each case, collision processing is performed the same way. All of our collisions are inelastic. Note that the particle based collision will introduce inconsistency in the deformation gradients on particles. It may produce artifacts for certain simulations and should only be turned on if necessary.

Collision objects are represented as level sets, which makes collision detection ($\phi \leqslant 0$) trivial. In case of a collision the local normal $n = \nabla\phi$ and object velocity v_{co} are computed. First, the particle/grid velocity v is transformed into the reference frame of the collision object, $v_{rel} = v - v_{co}$. If the bodies are separating ($v_n = v_{rel} \cdot n \geqslant 0$), then no collision is applied. Let $v_t = v_{rel} - n v_n$ be the tangential portion of the relative velocity. If a sticking impulse is required ($\|v_t\| \leqslant -\mu v_n$), then we simply let $v'_{rel} = 0$, where the prime indicates that the collision has been applied. Otherwise, we apply dynamic friction, and $v'_{rel} = v_t + \mu v_n v_t / \|v_t\|$, where μ is the coefficient of friction. Finally, we transform the collided relative velocity back into world coordinates with $v' = v'_{rel} + v_{co}$.

We used two types of collision objects: rigid and deforming. In the rigid case, we store a stationary level set and a potentially time-varying rigid transform, which we can use to compute ϕ, n, and v_{co} at any point. In the deforming case, we load level set key frames and interpolate them using $\phi(x, t + \gamma\Delta t) = (1-\gamma)\phi(x - \gamma\Delta t v_{co}, t) + \gamma\phi(x + (1-\gamma)\Delta t v_{co}, t + \Delta t)$, except we compute the velocity as $v_{co} = (1-\gamma)v(x, t) + \gamma v(x, t + \Delta t)$ instead of the average velocity.

Finally, we utilize a sort of *sticky* collision in situations where we want the material to stick to vertical or under-hanging surfaces. In this case, Coulomb friction is insufficient since the normal relative velocity would be zero (vertical) or positive (under-hanging and separating due to gravity). We achieve this effect by setting $v'_{rel} = 0$ unconditionally for collisions against these surfaces. Dirichlet boundary condition on grid nodes is equivalent to sticky collision.

12.2 Lagrangian Forces

In a traditional MPM simulation, forces are computed using we compute forces as in Equation 188. Letting each particle to store a deformation gradient has the advantages of a mesh-free method, such as effortless topology change. When MPM is used for simulating traditional rubber-like elastic solids, it makes more sense to have the connectivity information as in mesh-based FEM solvers because a mesh can provide more accurate deformation gradient computations and easier rendering. Here we show that for objects that are not intended to undergo topology change, a meshed approach is also available and can be easily integrated into the MPM framework.

In this case, we can use any Lagrangian force model (springs, finite elements, etc.) for which we can write down total potential energy $W(\mathbf{x}_p)$. Corresponding to this Lagrangian force model, we compute forces $\mathbf{f}_p = -\frac{\partial W}{\partial \mathbf{x}_p}$. We also assume that given any vector $\delta\mathbf{u}_q$ on particles we can multiply by force derivatives $\frac{\partial \mathbf{f}_p}{\partial \mathbf{x}_q}$ to obtain $\delta\mathbf{f}_p = \sum_q \frac{\partial \mathbf{f}_p}{\partial \mathbf{x}_q}\delta\mathbf{u}_q$. Since these force-related constructs are purely Lagrangian and are computed in the usual way, we will not elaborate on them here.

Although we have defined our mesh-based forces as Lagrangian forces, we must still apply them through the grid. We must describe how particle positions $\mathbf{x}_p$ relate to our (conceptually) moving grid nodes $\mathbf{x}_i$ so that forces can be evaluated. Then, we must compute $\mathbf{f}_i$ from $\mathbf{f}_p$ and find a means of computing $\delta\mathbf{f}_i$ given $\delta\mathbf{u}_i$. Doing this allows us to use Lagrangian forces as Eulerian forces. Comparing our update rules for $\mathbf{x}_p$ and $\mathbf{x}_i$ we find $\mathbf{x}_p = \sum_i w_{ip}^n \mathbf{x}_i$. Using the chain rule,

$$\mathbf{f}_i = \sum_p w_{ip}^n \mathbf{f}_p \qquad \delta\mathbf{f}_i = \sum_{p,q,j} w_{ip}^n \frac{\partial \mathbf{f}_p}{\partial \mathbf{x}_q} w_{jq}^n \delta\mathbf{u}_j. \tag{206}$$

Although this formula is written with three nested summations, the computation can be done efficiently by computing the summations consecutively. Since these forces are applied to the grid, both the MPM and Lagrangian approaches can be employed in the same simulation. Each particle is labeled as an MPM particle or a meshed particle. Note that the deformation gradient $\mathbf{F}_p^n$ stored on meshed particles is never used, since for those particles this quantity is computed using the mesh. This provides an effective means of coupling MPM with mesh-based approaches. This gives the precise surface tracking of Lagrangian techniques coupled with the automatic collision handling of Eulerian grids.

REFERENCES

Ando, R. and Tsuruno, R. (2011). A particle-based method for preserving fluid sheets. In *Proc ACM SIGGRAPH/Eurograph Symp Comp Anim*, SCA '11, pages 7–16.

Bonet, J. and Wood, R. (2008). *Nonlinear continuum mechanics for finite element analysis*. Cambridge University Press.

Bridson, R. (2008). *Fluid simulation for computer graphics*. Taylor & Francis.

Daviet, G. and Bertails-Descoubes, F. (2016). A semi-implicit material point method for the continuum simulation of granular materials. *ACM Trans Graph*, 35(4).

Gast, T., Fu, C., Jiang, C., and Teran, J. (2016). Implicit-shifted symmetric qr singular value decomposition of 3x3 matrices. Technical report, University of California Los Angeles.

Gast, T., Schroeder, C., Stomakhin, A., Jiang, C., and Teran, J. (2015). Optimization integrator for large time steps. *IEEE Trans Vis Comp Graph*, 21(10):1103–1115.

Gonzalez, O. and Stuart, A. (2008). *A first course in continuum mechanics*. Cambridge University Press.

Hegemann, J., Jiang, C., Schroeder, C., and Teran, J. M. (2013). A level set method for ductile fracture. In *Proc ACM SIGGRAPH/Eurograp Symp Comp Anim*, pages 193–201.

Irving, G., Teran, J., and Fedkiw, R. (2004). Invertible finite elements for robust simulation of large deformation. In *Proc ACM SIGGRAPH/Eurograph Symp Comp Anim*, pages 131–140.

Jiang, C. (2015). *The material point method for the physics-based simulation of solids and fluids*. PhD thesis, University of California, Los Angeles.

Jiang, C., Schroeder, C., Selle, A., Teran, J., and Stomakhin, A. (2015). The affine particle-in-cell method. *ACM Trans Graph*, 34(4):51:1–51:10.

Jiang, C., Schroeder, C., and Teran, J. (2016). An Angular Momentum Conserving Affine-Particle-In-Cell Method. *ArXiv e-prints*.

Klar, G., Gast, T., Pradhana, A., Fu, C., Schroeder, C., Jiang, C., and Teran, J. (2016). Drucker-prager elastoplasticity for sand animation. *ACM Trans Graph*, 35(4).

McAdams, A., Selle, A., Tamstorf, R., Teran, J., and Sifakis, E. (2011). Computing the singular value decomposition of 3×3 matrices with minimal branching and elementary floating point operations. Technical report, University of Wisconsin-Madison.

Ram, D., Gast, T., Jiang, C., Schroeder, C., Stomakhin, A., Teran, J., and Kavehpour, P. (2015). A material point method for viscoelastic fluids, foams and sponges. In *Proc ACM SIGGRAPH/Eurograph Symp Comp Anim*, pages 157–163.

Schroeder, C. (2016). Practical notes on implementing derivatives. *ArXiv e-prints*.

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

Stomakhin, A., Howes, R., Schroeder, C., and Teran, J. (2012). Energetically consistent invertible elasticity. In *Proc Symp Comp Anim*, pages 25–32.

Stomakhin, A., Schroeder, C., Chai, L., Teran, J., and Selle, A. (2013). A material point method for snow simulation. *ACM Trans Graph*, 32(4):102:1–102:10.

Stomakhin, A., Schroeder, C., Jiang, C., Chai, L., Teran, J., and Selle, A. (2014). Augmented MPM for phase-change and varied materials. *ACM Trans Graph*, 33(4):138:1–138:11.

Sulsky, D., Zhou, S., and Schreyer, H. (1995). Application of a particle-in-cell method to solid mechanics. *Comp Phys Comm*, 87(1):236–252.

Yue, Y., Smith, B., Batty, C., Zheng, C., and Grinspun, E. (2015). Continuum foam: a material point method for shear-dependent flows. *ACM Trans Graph*, 34(5):160:1–160:20.

Zhu, Y. and Bridson, R. (2005). Animating sand as a fluid. *ACM Trans Graph*, 24(3):965–972.