Modelling cracks in solid materials using the Material Point Method

Joel Wretborn

Examiner, Peter Münger
Supervisor, Rickard Armiento

Department of Physics, Chemistry and Biology
Linköping University, SE-581 83 Linköping, Sweden
Abstract

This thesis investigates a novel way to simulate cracks as an extension of the Material Point Method (MPM). Previous methods, like CRAMP (CRAcks with Material Points), often use an explicit crack representation to define the material crack. We use an implicit crack representation defined as the intersection between pieces of the original specimen created by a pre-fracture process. Material chunks are thereafter forced together using massless particle constraints. The method has proven successful in tearing scenarios, and the main benefits are: (1) minor computational overhead compared to the initial MPM algorithm; (2) simple to implement and scales well in 3 dimensions; (3) gives easy and controllable setup phase for desired material failure mode.

The development of the crack extension has required a fully general MPM solver that can handle arbitrarily many distinct bodies connected in the same simulation. Current collision schemes for MPM exists ([Huang et al. 2011]), however these are often focused on two-body collisions and does not scale well for additional objects due to inaccuracies in contact normal calculations. We present a method that uses an iterative pair-wise comparison scheme to resolve grid collisions that extends to any number of collision objects.

Keywords: Material Point Method, cracks
Acknowledgements

My deep appreciation goes to previous DreamWorks R&D FX Software Engineer Mihai Aldén and DreamWorks Manager of R&D Ken Museth without whom this thesis would not have been possible.

I also want to thank my supervisor Rickard Armiento for giving insightful and valuable feedback, and my examiner Peter Münger, for your interest and commitment in this project. Lastly, thanks Gergely Klar, for sharing your expertise in all areas related to the topic.
Contents

1 Introduction ............................................ 1
  1.1 Thesis structure .................................... 1

2 Background ............................................ 3
  2.1 Motion .................................................. 3
  2.2 Lagrangian and Eulerian view .......................... 3
  2.3 Deformation gradient .................................. 5
  2.4 Volume change ........................................ 5
  2.5 Strain ................................................... 6
  2.6 Stress ................................................. 8
  2.7 Notation ............................................... 9

3 The Material Point Method .............................. 11
  3.1 Governing equations ................................. 11
  3.2 Constitutive model .................................... 12
    3.2.1 Hyperelasticity ................................... 12
    3.2.2 Fixed corotated model ............................ 13
  3.3 Material points ...................................... 14
    3.3.1 Particle initialization ........................... 14
    3.3.2 Particle update .................................. 15
  3.4 The grid ............................................. 16
    3.4.1 Construction ...................................... 16
    3.4.2 Transfer to grid .................................. 17
    3.4.3 Force calculations ................................. 18
    3.4.4 Grid update ...................................... 19
    3.4.5 Transfer to particles ............................. 19
  3.5 Collisions ............................................ 20
  3.6 Algorithm overview .................................. 20

4 Multi-body solver .................................... 23
  4.1 Two-body systems .................................... 24
    4.1.1 Contact conditions ................................. 24
    4.1.2 Normal calculation ................................ 25
    4.1.3 Resolving collision ............................... 26
  4.2 N-body systems ....................................... 27
1 Introduction

The topic of numerical simulations of the mechanical behaviour of materials is interesting due to its vast range of applicability. It is used heavily in academia and science to investigate natural phenomenas, and it is a necessary addition to computer graphics where natural realism is desired.

The Material Point Method (MPM) is a simulation method that has received increased traction in commercial applications in recent years. The method works by discretizing the initial material into particles that carry information like mass and velocity through the simulation. As these material points are unconnected, a grid is introduced to simplify calculations. The equations of motion are solved on this grid, and the solutions are transferred back to the particles that get updated accordingly. Simulation methods that mix these approaches (particles/grids) are usually denoted hybrid Lagrangian/Eulerian methods.

Our goal with this thesis is to extend the MPM algorithm to handle simulations involving material fracture. Existing methods like CRAMP [Nairn 2003] and enriched constitutive models [Guamatsia et al. 2006], [Nguyen 2013] exist, however these are considered to be both computationally slow and cumbersome to setup.

This report is intended for people with limited experience in fluid dynamics and numerical simulations, including previous knowledge of MPM. The mathematical framework provided is at times rigorous and the physical intuition can sometimes be lost in detail. However, our goal has been to provide enough detail for someone uninitiated to be able to implement their own MPM solver while still providing enough context to create a broad understanding of numerical simulations in general.

1.1 Thesis structure

Chapter 2 aims to give a rough conceptual overview of continuum mechanics that will be required for a full understanding of MPM. This chapter is intended for readers without prior background in continuum mechanics and discusses important concepts like strain and stress.

Chapter 3 continues by giving a detailed layout of the basic MPM algorithm using an explicit time step integration scheme. The algorithm is presented as implementable steps.
Chapter 4 extends the basic MPM algorithm by a contact algorithm, enabling multiple bodies to be coupled into the same solver. This is initially presented by outlining a collision scheme between two objects, but it is thereafter extended to arbitrarily many collision objects.

In chapter 5 the multi-body solver is leveraged to enable material cracks in the MPM framework. An object is pre-fractured into chunks, represented as distinct bodies, and glued together by glue particles containing information regarding the material failure. When a threshold is reached, the glue particles are removed from the simulation. A couple of example scenarios will be presented to show the applicability of the algorithm.
2 Background

This chapter aims to introduce some essential concepts that will prove helpful in understanding the underlying mathematical framework of the Material Point Method. We will provide some basic insight into continuum mechanics using the notation and structure provided by [Bonet and Wood 1997]. For a more complete introduction to the subject we refer the reader to the book.

2.1 Motion

We will consider the motion of a deformable body represented by material points, and we need to track these particles as they move in time. This is usually expressed as a mapping from the particle’s initial position $X$ to its current position $x$ observed at some later time $t$ by

$$x = \phi(X, t).$$

(2.1)

The collection of all particles at a time $t > 0$ is referred to as their current configuration $\Omega_t$, and the initial configuration, also called the rest configuration, is denoted $\Omega_0$. For this relation to be physically meaningful this has to be a one-to-one mapping between $\Omega_0$ and $\Omega_t$, or else it would not be possible to distinguish between individual particles. It is thus safe to assume that the inverse

$$X = \phi^{-1}(x, t)$$

(2.2)

also exists.

2.2 Lagrangian and Eulerian view

The introduction of $\Omega_0$ and $\Omega_t$ brings with it two sets of coordinate systems, one for each frame of reference. In practice these two are often taken to be identical, but when choosing one or another the physical interpretation changes. By representing a quantity as a function of the initial configuration $q(X, t)$, we describe how the state of the particles change during the simulation. This is known as the Lagrangian view, or material description. Instead, by using the current configuration $q(x, t)$ the change refers to fixed points in space, and it is referred to as the Eulerian view or spatial description.
2. Background

Consider the figure above, and suppose we are interested in the mass $m$. We can choose to define it in the Lagrangian or Eulerian view as $m_L(X, t)$ or $m_E(x, t)$ respectively. A subscript $L,E$ was added to make the distinction clearer, however it is sufficiently defined by specifying the variable $x$ or $X$. As $X_p$ refers to the red material point, the value of $m_L(X_p, t)$ will be the mass of that particle and

$$m_L(X_p, t_1) = m_L(X_p, t_2) = m_L(X_p, t_3).$$

Instead, $x_i$ is fixed in space and will vary in time, depending on if there is a particle nearby or not. Judging from the picture,

$$m_E(x_p, t_1) = m_E(x_p, t_2) \neq m_E(x_p, t_3)$$

as at time $t_3$ a particle is occupying $x_i$.

It is possible to move between these different representations using equations 2.1 and 2.2. When using the Eulerian representation $m_E(x, t)$, we can get the Lagrangian quantity by the pull back operator defined as

$$m_L(X, t) = m_E(\phi(X, t), t). \quad (2.3)$$

Similarly, starting from a Lagrangian frame of reference $m_L(X, t)$ the corresponding Eulerian quantity is obtained by the push forward operator

$$m_E(x, t) = m_L(\phi^{-1}(x, t), t). \quad (2.4)$$

Figure 2.1: *Difference between the Lagrangian view and the Eulerian.* The $X_p$ coordinate refers to the red particle, and moves in space (i.e. its world space coordinates $(x, y, z)$ will change over time). The $x$ coordinate is a world space coordinate, and is fixed in time.
2. Background

The reason for choosing one view or another depends on what problem one is trying to solve. For example, most fluids simulations use the Eulerian view partly due to the fact that it allows for arbitrarily large deformations. This is because the underlying mesh (which is represented by the points in the example above) does not deform during the simulation (compare $\mathbf{X}$ with $\mathbf{x}$—the latter stays fixed). Conversely, when working with solid materials one usually has to consider the constitutive relationship between strain and stress for the material particle, and the Lagrangian view is then usually preferred. As we will see in the next chapter, MPM utilizes both of these descriptions.

2.3 Deformation gradient

When working with elastic materials, it is essential that we can relate the initial configuration of the body with its currently deformed state. For this purpose the deformation gradient $\mathbf{F}$ is defined as

$$\mathbf{F}(\mathbf{X}, t) = \frac{\partial \phi(\mathbf{X}, t)}{\partial \mathbf{X}}. \quad (2.5)$$

Implicit in the notation above is that the differentiation is made with respect to all permutations, and it could just as well be written

$$F_{ij} = \frac{\partial \phi(\mathbf{X}_i, t)}{\partial X_j}$$

for $i, j = 1, 2, 3$ in three dimensions. The result is that the deformation gradient is a second order tensor, and it will play a central role in measuring material strain. To understand why this definition is useful, it helps to look at the differential of the current position for a fixed time:

$$d\mathbf{x} = \frac{\partial \phi}{\partial \mathbf{X}} d\mathbf{X} + \frac{\partial \phi}{\partial t} dt = \frac{\partial \phi}{\partial \mathbf{X}} d\mathbf{X} = \mathbf{F} d\mathbf{X} \quad (2.6)$$

Equation 2.6 says that neighbouring particles in the spatial description are related to their relative material position, and that the deformation gradient is what defines this mapping. By virtue of the existence of $\phi^{-1}$, the inverse

$$\mathbf{F}^{-1} = \frac{\partial \mathbf{X}}{\partial \phi} \quad (2.7)$$

must also exist.

2.4 Volume change

Consider an infinitesimal volume element defined by the three vectors

$$d\mathbf{X}_1 = \begin{pmatrix} dX_1 \\ 0 \\ 0 \end{pmatrix}, d\mathbf{X}_2 = \begin{pmatrix} 0 \\ dX_2 \\ 0 \end{pmatrix}, d\mathbf{X}_3 = \begin{pmatrix} 0 \\ 0 \\ dX_3 \end{pmatrix} \quad (2.8)$$
in an orthonormal basis $E$ defined in the initial configuration. The volume of a rectangular parallelepiped is given by the product of each entry

$$dV = dX_1dX_2dX_3. \quad (2.9)$$

Our goal is to find the size of this volume element in the spatial view. By equation 2.6 we push $dX_i$ forward as

$$dx_i = F dX_i = \frac{\partial \phi}{\partial X_i} dX_i E_i = \frac{\partial \phi}{\partial X_i} dX_i. \quad (2.10)$$

The vectors $x_i$ might get skewed and scaled. In the general case, the volume of a parallelepiped is given by the triple product $a \cdot (b \times c)$. Using equation 2.10 in combination with the triple product yields the change of volume when going to the spatial view as

$$dv = dx_1 \cdot (dx_2 \times dx_3) = \frac{\partial \phi}{\partial X_1} \cdot (\frac{\partial \phi}{\partial X_2} \times \frac{\partial \phi}{\partial X_3}) dX_1 dX_2 dX_3. \quad (2.11)$$

This triple product above is the determinant of $F$. By using $J = \det(F)$ we can rewrite equation 2.11 to get

$$dv = JdV. \quad (2.12)$$

We thus see that the determinant of $F$ has a clear physical interpretation and expresses change in volume between the two reference frames.

### 2.5 Strain

Strain describes the degree to which a body has deformed. At its core, strain compares the un-deformed body with its current deformed state. Imagine a simple rod with initial length $L$ and area $A$ that has been stretched to some final length $l$ and area $a$. One way of measuring this deformation could be to relate initial length with stretched length, for example

$$\varepsilon_E = \frac{l - L}{L}. \quad (2.13)$$

It is clear that this strain measurement is not unique. Certainly, we could have chosen to divide with the final length $l$ instead of $L$, or made any other potential alteration and still represent some measure of change. The quantity $\varepsilon_E$ above is usually denoted as the engineering strain. The choice of strain measurement varies depending on the use case, and special consideration has to be made for the specific material being modeled.

As noted earlier, $F$ plays an important role in measuring strain. Consider two material vectors $dX_1$ and $dX_2$, deforming into spatial vectors $dx_1$ and $dx_2$. To get a sense of the deformation, the scalar product will be applied. This seems sensible.
since the scalar product encodes both information of change in length (stretching) and potential change of the angle between the vectors (skewing). Pushing the vectors forward with equation 2.6 yields

\[ dx_1 \cdot dx_2 = F dX_1 \cdot F dX_2 = dX_1 \cdot C dX_2. \] (2.14)

Here, \( C \) is the right Cauchy-Green deformation tensor, defined from the deformation gradient as

\[ C = F^T F. \] (2.15)

Looking at the difference \( dx_1 \cdot dx_2 - dX_1 \cdot dX_2 \) and using equation 2.14 yields

\[ \frac{1}{2}(dx_1 \cdot dx_2 - dX_1 \cdot X_2) = dX_1 \cdot E dX_2 \] (2.16)

where \( E \) is the Green strain tensor defined as

\[ E = \frac{1}{2}(C - I). \] (2.17)

Note that \( E \) here is a capital letter to signify that the strain measurement is done in the Lagrangian view. Conversely, using the same reasoning from a spatial perspective would result in a similar expression with the strain tensor instead

\[ e = \frac{1}{2}(I - (FF^T)^{-1}) \]

and used in a similar fashion as

\[ \frac{1}{2}(dx_1 \cdot dx_2 - dX_1 \cdot X_2) = dx_1 \cdot e dx_2. \]

One final part remains to be made if we want to relate this expression with the original example. Let the vectors in each frame coincide so that \( X_1 = X_2 = X \) and \( x_1 = x = x \), and denote the scalar products

\[ X \cdot X = ds^2 \quad \text{and} \quad x \cdot x = ds^2. \] (2.18)

Now, using our previous result from equation 2.16 and dividing by \( ds^2 \) yields

\[ \frac{ds^2 - ds'^2}{2ds^2} = \frac{dX \cdot E dX}{ds}. \] (2.19)

and the similarities with the engineering strain used previously becomes somewhat apparent. Note that \( \frac{dX}{ds} \) is a unit vector in the direction of \( dX \). As a final remark we again reiterate that there are many different strain measurements, and that the Green strain tensor discussed above is one special case.
2.6 Stress

Stress aims to relate the internal body forces that arise in a body due to its current deformation. In the previous section we introduced a number of different strain measurements, and one would have merit to think that this would yield a multitude of different stress measurements as well. Fortunately this is not the case, as the coupling of strain to stress is kept separated in a constitutive relationship specific for each material.

The discussion of stress revolves around traction, \( t \), at a point inside the material and along a specific direction \( n \). The force \( b \) applied on some area \( a \) perpendicular to \( n \) defines the traction as

\[
t(n) = \lim_{a \to 0} \frac{b}{a}.
\] (2.20)

Note that as \( a \to 0 \) so also does \( ||b|| \to 0 \) and the limit is valid. It is also possible to define the traction vector as

\[
t(n) = \sigma n
\] (2.21)

where \( \sigma \) is a second order tensor called the Cauchy stress tensor.

Before any discussion on dynamics, it is often useful to consider the static case. Imagine a deformed body with volume \( v \) and surface \( \partial v \) represented in the spatial view. Assume body forces \( f \) are acting on the body per unit volume, and respective traction forces \( t \) per unit area along the surface. If the body in question is in equilibrium, all forces must cancel, giving

\[
\int_{\partial v} t \, da + \int_{v} f \, dv = 0
\] (2.22)

Expanding the first integral by equation 2.21 makes it possible to express the above relation in term of the Cauchy stress as

\[
\int_{\partial v} \sigma n \, da + \int_{v} f \, dv = 0.
\] (2.23)

As the body is necessarily bounded it is possible to convert the traction part of this relation using the Gauss divergence theorem. For a second order tensor \( S \) that relations is

\[
\int_{\partial v} S n \, da = \int_{v} \nabla : S \, dv
\] (2.24)

where the double contraction \( \nabla : S = \nabla S : I \) was used. Using equation 2.24 in equation 2.23:

\[
\int_{v} (\nabla : S + f) \, dv = 0
\] (2.25)

The argument that this equation must hold could be made for any other enclosed volume \( v' \) in the body, which implies that the integrand must be zero itself, that is

\[
\nabla : S + f = 0.
\] (2.26)
This encapsulates the condition required for spatial equilibrium in the observed body. For the dynamic case this equation will become unbalanced by some residual force \( r \) per unit volume, and so it serves to also express the equilibrium equation as

\[
\nabla : S + f = r.
\] (2.27)

In the above derivation all quantities (area normals and force vectors) were expressed in the Eulerian frame. By using the deformation gradient it is possible to go between the spatial and material view. The transformations can be seen in table 2.1.

<table>
<thead>
<tr>
<th>Stress tensor</th>
<th>Symbol</th>
<th>Area normal</th>
<th>Force</th>
<th>Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cauchy</td>
<td>( \sigma )</td>
<td>spatial</td>
<td>spatial</td>
<td>( P = J\sigma F^{-T} )</td>
</tr>
<tr>
<td>First Piola-Kirchoff</td>
<td>( P )</td>
<td>spatial</td>
<td>spatial</td>
<td></td>
</tr>
<tr>
<td>Second Piola-Kirchoff</td>
<td>( S )</td>
<td>material</td>
<td>material</td>
<td>( S = JF^{-1}\sigma F^{-1} )</td>
</tr>
</tbody>
</table>

Table 2.1: An overview on what measurements the different stress tensors represent. Taken from http://web.stanford.edu/class/cs205b/lectures/lecture7.pdf

These stress measurements are all equivalent, they only apply to the different material configurations. Compare this to the different strain measurements, which in general yield different results depending on the measurement used.

## 2.7 Notation

This section aims to provide a unified image on what quantities are used in this report and what they represent. If nothing else is stated, scalars will be represented by non-bold lower case letters (e.g. \( m_p, \rho, w_{ip} \) ); vectors with bold lower case letters (e.g. \( v_k^i, f_{cl}^* \)); and tensors and matrices by bold upper case letters (e.g. \( R, F \) ). Exceptions to these rules exist, where for example the Cauchy stress tensor is a second order tensor but is commonly denoted as \( \sigma \).

It is important to distinguish between Eulerian and Lagrangian properties. Eulerian quantities live on grid nodes, and will be referred to with a subscript \( i \). Particle quantities, or Lagrangian quantities, will be subscripted with \( p \). These subscripts can be appended to many quantities, and as example \( x_p \) refers to the position of particle \( p \) while \( x_i \) denotes the position of node \( i \). They are not exclusive and can be combined, whereas \( w_{ip} \) refers to the contribution of particle \( p \) to grid node \( i \). In chapter 5 a new type of particle will be introduced. To distinguish them from other particles a subscript \( g \) will be used instead.

A superscript \( k \) (e.g. \( v^k_i \)) is used to represent a measurement done early at that time step \( k \), before forces have been applied to the grid. The corresponding velocity after the force update is denoted \( v^{k+}_i \).

Below is an attempt to group all quantities used throughout this paper. Subscripts and superscripts will be added following the rules outlined earlier.
Table 2.2: Notation of common quantities.
3 The Material Point Method

The Material Point Method is a relatively novel simulation method that was introduced by [Sulsky et al. 1995]. It has been successful in simulating a broad range of physical phenomena, and has recently received increasing traction in the animation community.

MPM is a hybrid method, and as such combines an Eulerian mesh with Lagrangian particles. First, a continuous material is discretized into material points. The particles store all information that will be carried on through the simulation such as, position, velocity, deformation, and other potential properties related to the constitutive model.

An Eulerian grid is used in the background to perform calculations, more specifically to solve the equations of motion. Particles are rasterized to the grid by a weighting function—a kernel—and transfer its attributes to the grid nodes. The solution obtained on the grid is then transferred back to the particles using a PIC/FLIP update scheme, and the particles are advected. Afterwards, the grid is discarded and a new simulation step is initialized. An overview of the algorithm can be seen in figure 3.1.

The rest of this chapter will go into greater detail regarding the steps described above, with the final section summarizing the full MPM algorithm outlined as implementable steps.

3.1 Governing equations

The fundamental equations that will determine the motion of the material is the standard conservation equations for mass and momentum,

\[ \frac{d\rho}{dt} + \rho \nabla \cdot \mathbf{v} = 0, \]

\[ \rho \mathbf{a} = \nabla \cdot \mathbf{\sigma} + \rho \mathbf{b}. \]  

Mass will be kept constant throughout the simulation by keeping particle mass \( m_p \) fixed, and the first equation requires no further attention. The momentum equation 3.2 states that the acceleration of a particle can be determined given the internal forces, represented by the Cauchy stress tensor \( \mathbf{\sigma} \), and the external forces \( \mathbf{b} \). For large
3. The Material Point Method

Figure 3.1: Overview of the algorithm. The steps are classified as Lagrangian and Eulerian to signify what entity is being manipulated, particles or grid nodes. Explanation of the steps: (1) initialized particles are used as input; (2) a background grid is created; (3) mass and velocity are rasterized to the grid, and internal forces are calculated using the constitutive model (red values are inside a boundary and are not valid); (4) new velocities are calculated using external and internal forces; (5) boundary collisions are resolved; (6) velocity is transferred back to the particles; (7) particle position and deformation is updated.

deformations, these governing equations must be supplemented with a constitutive equation that relates strain to stress [Sulsky et al. 1995]. The following section aims to discuss this problem.

3.2 Constitutive model

The constitutive model encompasses a way of determining the stress based on the deformation of a material. To accomplish this, it may use any strain measurement and computations necessary to yield the final stress. All our materials will be purely elastic, which means that the constitutive relationship is only dependent on the current state of deformation.

3.2.1 Hyperelasticity

Hyperelasticity is a type of elastic behaviour that refers to constitutive models where stress can be determined by the gradient from some underlying scalar potential \( \Psi \). This is a suitable model for materials that has no intrinsic directional dependence, like concrete or rubber. The opposite would be a material whose behaviour differs depending on what direction one is observing. Examples include fiber materials like
wood or muscle tissue.

Continuing, if this scalar potential is determined from the deformation $F$, the initial statement would result in

$$\mathbf{P} = \nabla \Psi(F) = \frac{\partial \Psi(F)}{\partial F}.$$  \hfill (3.3)

The notation introduced is the piece-wise derivative of $\Psi$ with respect to $F$, and the entries of $\mathbf{P}$ can also be written $P_{ij} = \frac{\partial \Psi}{\partial F_{ij}}$. $\mathbf{P}$ here is in fact the First Piola-Kirchoff stress. By using the transformation in table 2.1 we get the Cauchy stress as

$$\sigma = \frac{1}{J} \mathbf{P} \mathbf{F}^T = \frac{1}{J} \frac{\partial \Psi}{\partial F} \mathbf{F}^T.$$  \hfill (3.4)

Equation 3.4 captures the strain-stress relationship, and supplements the conservation equations 3.1 and 3.2 with the additional information needed to model large deformations. By defining this potential we will thus have all the tools needed to fully describe the motion of our material points.

### 3.2.2 Fixed corotated model

We will define the potential $\Psi$ using the fixed corotated model as developed by [Stomakhin et al. 2012]. It was created from the more common neo-Hookean model as a more robust alternative. The energy density can be written as

$$\Psi = \mu |\mathbf{F} - \mathbf{R}|_F^2 + \frac{\lambda}{2} (J - 1)^2.$$  \hfill (3.5)

The first term includes $\mathbf{R}$ which comes from the polar decomposition of $\mathbf{F}$ (see appendix B), and its purpose is to remove the contribution of rigid body rotations to the potential energy. The second term considers the volume change of the material, where $J = \det \mathbf{F}$. In equation 3.5 the Frobenius norm $|\cdot|_F$ was used. It is defined by

$$|\mathbf{A}|_F^2 = \sqrt{\sum_i \sum_j |a_{ij}|^2}$$  \hfill (3.6)

where $a_{ij}$ are the entries of the matrix $\mathbf{A}$. Forces will be calculated using equation 3.4 where the gradient of $\Psi$ is required [Jiang 2015]:

$$\frac{\partial \Psi}{\partial \mathbf{F}} = 2\mu(\mathbf{F} - \mathbf{R}) + \lambda(J - 1)\mathbf{F}^{-T}.$$  \hfill (3.7)

The Lamé parameters $\mu$ and $\lambda$ are material specific properties that can be varied to get different material characteristics. They are maybe more commonly expressed in Young’s modulus $E$ and Poisson’s ratio $\nu$, who relate to $\mu$ and $\lambda$ by

$$\mu = \frac{E}{2(1+\nu)}$$
3. The Material Point Method

\[ \lambda = \frac{\nu E}{(1 + \nu)(1 - 2\nu)} \]  

(3.8)

with \( E \in [0, \infty) \), \( \nu \in (-1, \frac{1}{2}) \). Young's modulus affect the stiffness of a material measured in Pascal, where a higher value indicates a stiffer material. Poisson's ratio relates the lateral measurement with change in length when elastically stretching a material, and as such is unitless. Typical values include \( E = 200 \text{ GPa}, \nu = 0.3 \) for steel and \( E = 1 \text{ MPa}, \nu = 0.45 \) for a rubbery-like material.

3.3 Material points

In the Lagrangian view the material is represented by material points. This section describes the particle operations for the MPM algorithm.

3.3.1 Particle initialization

The first step for the simulation is to discretize the initial, continuous, representation of the body into particles. Multiple schemes can be used to find the initial position of the material points. The one used for this work is as follows:

1. Impose a grid over the shape with uniform grid spacing \( l \).
2. Define a grid distribution \( d \) that indicates how many particles per grid cell and dimension should be used. Values of \( d \in [1, 3] \) usually gives good results.
3. Go over all grid cells where the shape overlaps and distribute particles accordingly.

An illustration of the above scheme can be seen in figure 3.2. The initial particle mass \( m^0_p \) and initial volume \( V^0_p \) could be stated as is, but it is often more convenient to relate it to some material density \( \rho_0 \). By declaring \( \rho_0 \) and \( l, d \), the initial quantities can be found as

\[ V^0_p = \frac{l^\alpha}{d^\alpha}, \]

\[ m^0_p = \rho_0 V^0_p. \]  

(3.9)

where \( \alpha \) is 2 or 3 depending on the number of dimensions used. We will moreover take the mass of a particle to be constant over time, which means that

\[ m^k_p = m_p. \]  

(3.10)

All particle attributes that need to be initialized is summarized in table 3.1. The purpose of the Lamé parameters is described in section 3.2. Note that the attributes stored are specific to this MPM implementation.
3. The Material Point Method

Figure 3.2: Discretization. An illustration of one possible discretization of a shape in 2D. Extending the argument to 3D is straightforward.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mass</td>
<td>$m_p$</td>
</tr>
<tr>
<td>Initial volume</td>
<td>$v^0_p$</td>
</tr>
<tr>
<td>Position</td>
<td>$x_p$</td>
</tr>
<tr>
<td>Velocity</td>
<td>$v_p$</td>
</tr>
<tr>
<td>Deformation gradient</td>
<td>$F_p$</td>
</tr>
<tr>
<td>Lamé parameters</td>
<td>$\lambda_p, \mu_p$</td>
</tr>
</tbody>
</table>

Table 3.1: A list of all attributes stored on each particle.

3.3.2 Particle update

The particle properties that will change during the simulation is the time dependent parameters $x^k_p$, $v^k_p$ and $F^k_p$. As outlined in section 3.4.5, the updated velocity $v^{k+1}_p$ will be extrapolated from the grid. As such, left to calculate is $x^{k+1}_p$ and $F^{k+1}_p$. It is possible to let the constitutive model change over time as well, however we do not consider that possibility here.

Updating the position is straightforward. We use an explicit update scheme and calculate the new position by means of linear integration:

$$x^{k+1}_p = x^k_p + \Delta t v^{k+1}_p.$$  \hfill (3.11)

The expression to update $F$ is found by a Taylor expansion around $t$, and is given in [Stomakhin et al. 2013] as

$$F^{k+1}_p = (I + \Delta t \nabla v^{k+1}_p) F^k_p$$ \hfill (3.12)

where $I$ is the identity matrix. The gradient $\nabla v^{k+1}_p$ can be found using the outer product

$$\nabla v^{k+1}_p = v^{k+1}_p \otimes \nabla w^k_{ip} = v^{k+1}_p (\nabla w^k_{ip})^T.$$ \hfill (3.13)

Bear in mind that the position of the particle and its deformation is coupled. This makes sense—$F$ is a function that maps our current position to the initial, undeformed, position. As a result, particle operations that directly influence a position
of a particle should be avoided. Instead, if a specific particle position \( \vec{x}_p \) is desired, adjust it by modifying \( \vec{v}_p^{k+1} \) according to

\[
\vec{v}_p^{k+1} = \frac{\vec{x}_p - \vec{x}_p}{\Delta t} \quad \text{and} \quad \vec{v}_p^{k+1} = \vec{v}_p^{k+1}
\]

(3.14)

before the particle update takes place. This scheme keeps the consistency of \( \vec{x}_p^{k+1} \) and \( \vec{F}_p^{k+1} \) intact while allowing for dynamic behaviour, which could be desirable if, for example, certain boundary conditions are desired.

### 3.4 The grid

The Lagrangian particles defined previously provide a natural representation of the material under observation. These points, however, are uncoupled as no form of connectivity between particles exists. This construction is suitable for describing objects undergoing large deformations. However, in turn, the calculation of material derivatives becomes difficult and expensive. The background Eulerian grid is a solution to this problem, as it provides a fixed mesh of the object where calculations can be done rapidly.

MPM utilizes four properties on the grid: mass \( m_i^k \), velocity \( \vec{v}_i^k \), next velocity \( \vec{v}_i^{k+1} \), and forces \( \vec{f}_i^k \). Of these, mass and velocity are rasterized directly to the grid from the particles, while the forces are calculated from the constitutive model and used to calculate the next velocities.

#### 3.4.1 Construction

To define the background grid we need a function that projects material properties from a particle to the grid. This function is commonly known as a *kernel*. Our intuition is that a particle \( p_1 \) close to a node \( i \) should contribute more to the node than some other particle \( p_2 \) located further away. If we denote the basis function between a particle and node as \( w_{i \rightarrow p} \) (as in weight), the mathematical expression of the previous statement would be that

\[
w_{i \rightarrow p_1} \geq w_{i \rightarrow p_2}
\]

(3.15)

if \( ||\vec{x}_{p_1} - \vec{x}_i|| < ||\vec{x}_{p_2} - \vec{x}_i|| \). The choice of kernel will ultimately affect how the material points will be represented in the spatial view. Typically, one dimensional splines extended to multiple dimensions by a dyadic product is used for its simplicity to analyse and ease of use in computations [Klar et al. 2016]. We will employ the same kernel as described in [Stomakhin et al. 2013], who define it using one-dimensional cubic B-splines,

\[
N(x) = \begin{cases} 
\frac{1}{6}x^3 - x^2 + \frac{2}{3} & \text{if } 0 \leq |x| < 1 \\
-\frac{1}{6}|x|^3 + x^2 - 2|x| + \frac{4}{3} & \text{if } 1 \leq |x| < 2 \\
0 & \text{otherwise}
\end{cases}
\]

(3.16)
The kernel is now defined as the dyadic product of \( N(x) \) in three dimensions

\[
N_i^h(x_p) = N\left(\frac{x_p}{l} - i\right)N\left(\frac{y_p}{l} - j\right)N\left(\frac{z_p}{l} - k\right)
\]  

(3.17)

where \( x_p = (x_p, y_p, z_p) \) is the world coordinates of particle \( p \), \( i = (i, j, k) \) is the grid index defined in index space, and \( l \) is as previous the grid spacing. The gradient will be needed to calculate grid forces, and is calculated in the usual manner by

\[
\nabla N_i^h = \left(\frac{\partial N_i^h}{\partial x}, \frac{\partial N_i^h}{\partial y}, \frac{\partial N_i^h}{\partial z}\right)
\]

and

\[
\frac{\partial N_i^h}{\partial x} = \frac{\partial N\left(\frac{x_p}{l} - i\right)N\left(\frac{y_p}{l} - j\right)N\left(\frac{z_p}{l} - k\right)}{l}
\]

(3.18)

\[
\frac{\partial N_i^h}{\partial x} = \frac{\partial N\left(\frac{x_p}{l} - i\right)N\left(\frac{y_p}{l} - j\right)N\left(\frac{z_p}{l} - k\right)}{l}
\]

(3.19)

The other parts of the gradient can be deducted by similarity from the expression above. By using \( \text{sign}(x) = 1 \) if \( x \geq 0 \), otherwise \( \text{sign}(x) = -1 \), the derivative of \( N(x) \) can be written

\[
N'(x) = \begin{cases} 
\text{sign}(x)\frac{3}{2}x^2 - 2x & \text{if } 0 \leq |x| < 1 \\
-\text{sign}(x)\frac{1}{2}x^2 - 2x - 2\text{sign}(x) & \text{if } 1 \leq |x| < 2 \\
0 & \text{otherwise}
\end{cases}
\]

(3.20)

Now, the weight is simply

\[
w_{ip} = N_i^h(x_p)
\]

(3.21)

\[
\nabla w_{ip} = \nabla N_i^h(x_p).
\]

(3.22)

Weights are associated with a particle at position \( x_p \), and it makes sense to extend the notation to the weight. As such, the weight for a specific particle and time step will be referred to as \( w_{ip} \).

### 3.4.2 Transfer to grid

The definitions made above now allows us to transfer particle attributes to the grid. Mass can be rasterized by

\[
m_i^k = \sum_p m_p w_{ip}^k
\]

(3.23)

It is not possible to rasterize velocity directly to the grid, as this would not result in conservation of momentum [Stomakhin et al. 2013]. Instead, we transfer velocities by rasterizing particle momentum and dividing it by the nodal mass in equation 3.23,

\[
p_i^k = \sum_p m_p v_{ip} w_{ip}^k
\]

(3.24)
\[
\dot{q}_i^k = \frac{p_i^k}{m_i^k}
\]

(3.25)

where care has to be taken so as not to divide by zero. In general, a Lagrangian quantity \(q_p\) can be converted to the Eulerian grid as

\[
q_i = \sum_p q_p w_{ip}.
\]

(3.26)

### 3.4.3 Force calculations

It is possible to derive the internal forces on the grid either from the weak form of the momentum equation 3.2 or as the gradient of total potential energy. As we have a well defined energy density we will go for the latter option.

The total energy \(W\) of the system can be expressed as a function of the initial configuration \(\Omega_0\) using an integral,

\[
W = \int_{\Omega_0} \Psi(F(X))dX.
\]

(3.27)

Discretization of equation 3.27 changes integration to summation. The integration was over the initial occupied volume, and as such the initial particle volume \(V_0\) needs to be included in the discretized expression [Stomakhin et al. 2013]:

\[
\bar{W} = \sum_p V_0 \Psi_p(F_p)
\]

(3.28)

Forces can be calculated from a scalar field \(W\) by \(f = -\nabla W\). We want to use the discretized version of the energy potential \(\bar{W}\), and we want to find the spatial forces corresponding to this field. By taking the derivative with respect to the spatial discretization and using the chain rule we get

\[
f_i = -\nabla \bar{W} = -\frac{\partial \bar{W}}{\partial x_i} = -\sum_p V_0 \frac{\partial \Psi_p(F_p)}{\partial F} (F_p)^T \nabla w_{ip}
\]

(3.29)

Rewriting equation 3.29 for a specific time step yields

\[
f_i^k = -\sum_p V_0 \frac{\partial \Psi_p(F_p^k)}{\partial F} (F_p^k)^T \nabla w_{ip}^k + f_e(x_i)
\]

(3.30)

where an external force term \(f_e\) has been added. This typically is gravity, but could include any external force field.
3. The Material Point Method

3.4.4 Grid update

The grid update aims to find values for the calculated velocity $v_i^{k+}$. Using grid forces we calculate it by linear integration as

$$v_i^{k+} = v_i^k + \frac{\Delta t}{m_i} f_i^k.$$  \hspace{1cm} (3.31)

The grid velocity $v_i^{k+}$ is not necessarily final and may be altered after this step. This will be the case when resolving collisions.

3.4.5 Transfer to particles

As mass is kept constant on the particles to preserve total mass, the only property we need to transfer back from the grid to the particles is velocity. There are different ways to go about this, and we will use the combined PIC/FLIP scheme as in [Stomakhin et al. 2013].

The PIC (Particle-In-Cell) update scheme transfers grid velocities to particles in a manner that closely resembles the particle-to-grid transfer applied earlier. A particle obtains its velocity by gathering nearby, updated, nodal velocities $v_i^{k+}$ weighted by the kernel as

$$v_{[\text{PIC}]p}^{k+1} = \sum_i v_i^{k+} w_{ip}^k.$$  \hspace{1cm} (3.32)

In contrary to the particle-to-grid transfer the PIC transfer does not conserve angular momentum [Jiang et al. 2015]. This is no surprise as particles by default has no notion of rotation, and the result is that PIC introduces damping on the overall rotational motion of an object. PIC effectively works as a velocity filter on the particles, and, as a result of the dissipation, is generally stable.

To alleviate the damping of PIC, the FLIP (FLuid Implicit Particle) update scheme is introduced. The idea is to do an incremental update using the previous velocity of the particle instead of extrapolating the new velocity directly from the grid,

$$v_{[\text{FLIP}]p}^{k+1} = v_p^k + \sum_i (v_i^{k+} - v_i^k) w_{ip}^k.$$  \hspace{1cm} (3.33)

As can be seen in equation 3.33, the new velocity is calculated dependent on the velocity change on the grid, whereas equation 3.32 discards all notion of the previous particle velocity. This greatly reduces the loss of angular momentum as seen in PIC. Unsurprisingly, this also makes FLIP less stable as instabilities have a greater tendency to live on through the simulation, whilst they are damped out by the PIC update [Jiang et al. 2015]. These opposing behaviours work well together, and a combination tends to produce compelling results. We do this by a linear interpolation of the different updates schemes:

$$v_p^{k+1} = av_{[\text{FLIP}]p}^{k+1} + (1-a)v_{[\text{PIC}]p}^{k+1}.$$  \hspace{1cm} (3.34)
The constant $a$ is a value between $[0, 1]$, where 0 gives pure PIC and 1 pure FLIP. All simulations in this report has used $a = 0.95$.

### 3.5 Collisions

Many meaningful simulations include some form of interaction with boundary objects. We consider all boundary objects as rigid and undeformable, and all boundary collisions will be inelastic. The algorithm we use is heavily influenced by [Stomakhin et al. 2013] and [Klar et al. 2016], but due to our explicit time step update it will simplify slightly. The collision scheme used is often referred to as *sticky*.

All boundary geometry are represented as signed distance functions $\phi$ which makes collision checking fast and straightforward. We use the convention that $\phi(x) < 0$ if $x$ is inside the body. Collisions could be enforced on a particle based level. However, this tends to produce poor results as $x_p^{k+1}$ and $F_p^{k+1}$ would not be updated properly, resulting in them being out of sync [Klar et al. 2016]. Instead, impact is determined on the grid, after the calculation of the new grid velocities $v_i^{k+1}$. As the collision is inelastic, the resolution is simple: if the grid node is inside the body (i.e. $\phi(x_i) < 0$), set the next velocity of that node to the velocity of the boundary $v_c$,

$$v_i^{k+1} = v_c^k(x_i).$$  \hfill (3.35)

See figure 3.1 for an illustration.

At first glance it is not clear what particle behaviour the above scheme will result in; by the formulation one might get the impression that particles will stick as soon as one of its grid nodes comes in contact with a boundary object. This is not the case. Due to the nature of the rasterization, a single particle will occupy multiple grid nodes (where the number depends on the size of the kernel) and a particle is stuck only if all grid nodes are inside the boundary. Instead, the simple scheme presented is surprisingly useful for a great variety of cases. Non-sticky collisions can be further studied in [Klar et al. 2016].

### 3.6 Algorithm overview

The full MPM algorithm can be summarized into the following steps.

1. Discretize the shape into particles using the scheme in section 3.3.1. Initialize $x_p$, $v_p$, $F_p$, $\lambda_p$, and $\mu_p$ as desired.
2. Create a uniform grid that can store $m_i^k$, $v_i^k$, $v_i^{k+}$, and $f_i^k$.
3. Transfer mass, velocity, and force to the grid using equations 3.23, 3.24, and 3.30:

$$m_i^k = \sum_p m_p w_{ip}^k.$$
3. The Material Point Method

\[ p_i^k = \frac{1}{m_i^k} \sum_p m_p v_{ip} w_{ip}^k \]

\[ f_i^k = -\sum_p V_p^0 \frac{\partial \Psi_p(F_p^k)}{\partial F_p} (F_p^k)^T \nabla w_{ip}^k + f_c(x_i) \]


\[ v_i^{k+} = v_i^k + \Delta t f_i^k \]

5. Resolve boundary collisions by modifying the velocity vector \( v_i^{k+} \). If \( \phi(x_i) < 0 \), use equation 3.35 to set \( v_i^{k+} = v_c^k(x_i) \).

6. Transfer velocity back to the particles using equation 3.34:

\[ v_p^{k+1} = a v_{\text{FLIP}}^{k+1} + (1 - a) v_{\text{PIC}}^{k+1} \]

7. Update particle position and deformation with equations 3.11, 3.12, and 3.13:

\[ x_p^{k+1} = x_p^k + \Delta t v_p^{k+1} \]

\[ F_p^{k+1} = (I + \Delta t v_p^{k+1} (\nabla w_{ip}^k)^T F_p^k) \]

8. Discard the grid, and start a new time step from step 2.
3. The Material Point Method
4 Multi-body solver

In the last chapter we introduced a framework for coupling material points with some boundary object represented as a signed distance function. Already this gives us great freedom in designing our simulation setup to the problem at hand. It is, for example, possible to devise an experiment with multiple separate bodies in the same solver. That is, given two bodies $b_1$, $b_2$ that are spatially separated, we can discretize each body into sets of points $p_{b_1}$ and $p_{b_2}$. By adding both sets of points to the simulation they will be able to interact independently with boundaries, and even to a certain degree with each other. This is not, however, without errors.

Due to the nature of the MPM algorithm, all points—even if separated spatially as different bodies—will get rasterized to the same grid where the equations of motion are solved. This is not a problem as long as the bodies in question are far away from each other, but it becomes an issue when the two bodies are close.

Imagine a case where two bodies collide. At the interface of the collision there will exist particles from both bodies that are rasterized to the same grid node. These particles will be treated as if they were from the same body, as the grid has no way of distinguishing where the contribution came from. As a result, particles at the interface will feel non-physical attraction forces from the other body creating visible artifacts. Most notably

- merging of bodies (see fig 4.1) and,
- non-physical (and non-controllable) friction.

This chapter will introduce an algorithm for resolving body-to-body collisions inside the solver. It allows for arbitrarily many colliding bodies.

The two-body collision algorithm first discussed is heavily influenced by the work of [Huang et al. 2011], who also propose an extension that would allow for more than two colliding objects at a grid node. Their methods works by finding a single contact normal for each body at a grid node, and they resolve the collision along that normal for each body independently. We were unable to find a way to construct such a vector that also would conserve momentum. The method we will outline in this chapter preserves momentum exactly.
4. Multi-body solver

Figure 4.1: Falling teapots. Dropping teapots on to each other can lead to them merging (the right-most picture).

4.1 Two-body systems

The problem statement previously introduces the core of the problem: loss of information at the collision interface due to the mutual grid. The natural solution is to introduce additional grids.

Each body $b$ will be rasterized to its own set of grids, storing nodal information $m_k^b, \mathbf{v}_k^b, \mathbf{v}_k^b + \mathbf{f}_k^b$. The grids must be aligned to allow for comparisons between grid values of different bodies, i.e. $m_k^b$ and $m_k^d$ must refer to the masses of body $b$ and $d$ respectively at the same world space coordinate at time $k$. To avoid confusion, we will from here on refer to a grid node as a node corresponding to a grid position $\mathbf{x}_i$. That means that $m_{bi}^k$ and $m_{di}^k$ contributes to the same grid node, but to different grids.

By default the grids are uncoupled. Our goal here is to define exactly what is meant with two grids colliding, and to find governing equations that can resolve these collisions in a physically realistic manner.

4.1.1 Contact conditions

Similar to the boundary collision scheme, it makes sense to consider a node as colliding if it is inside a colliding body. With two colliding bodies $b$ and $d$ this condition translates to checking whether grid nodes overlap, as can be done by

$$m_{bi} > 0 \land m_{di} > 0.$$ (4.1)
4. Multi-body solver

This is not enough to determine a collision or not, as the different grids could be moving away from each other (this expression is somewhat inaccurate—grids are static and do not move, however we consider the virtual motion \( \delta x_i = x_i + \Delta t v_i \). This gives us a notion of ‘grid motion’, and motivates the previous wording). Instead we want to constraint the velocity of a grid node so that the two grids won’t penetrate further. This is expressed as the impenetrable condition

\[
v_{bi} \cdot n_{(b,d)i} + v_{di} \cdot n_{(d,b)i} = 0 \tag{4.2}
\]

where \( n_{(b,d)i} \) are an outwards pointing normal from \( b \) to \( d \) and vice versa. To ensure conservation of momentum we require the normals to be collinear:

\[
n_{(b,d)i} = -n_{(d,b)i} \tag{4.3}
\]

Combining equation 4.3 and equation 4.2 yields

\[
0 = (v_{bi} - v_{di}) \cdot n_{(b,d)i} = (v_{bi} - \frac{m_{bi} v_{di} + m_{di} v_{di}}{m_{bi} + m_{di}}) \cdot n_{(b,d)i} = (v_{bi} - v_{i}^{cm}) \cdot n_{(b,d)i}
\]

where we used the center of mass velocity

\[
v_{i}^{cm} = \frac{\sum b m_{bi} v_{bi}}{\sum b m_{bi}}, \tag{4.4}
\]

and the fact that \( v_{bi} \cdot n_{(b,d)i} = v_{di} \cdot n_{(b,d)i} \) (derivable from equation 4.2). As the impenetrability condition should only be enforced on colliding bodies, only velocity in the direction of the normal should be restricted. Thus, two grids are considered to be colliding at a grid node if

\[
(v_{bi} - v_{i}^{cm}) \cdot n_{(b,d)i} > 0 \iff (v_{di} - v_{i}^{cm}) \cdot n_{(d,b)i} > 0. \tag{4.5}
\]

4.1.2 Normal calculation

As the initial shape has been discretized into unconnected points, no notion of ‘outwards facing’ remains on points. Instead we need to determine it from the current configuration. The mass grid encodes information regarding locality, and it makes sense to derive the normal as a measurement of how the mass distribution changes over the grid. Denote the normal of a body \( n_{bi} \). Then

\[
n_{bi} = -\frac{\nabla m_{bi}}{|\nabla m_{bi}|}, \tag{4.6}
\]

where the minus sign was introduced to create an outwards facing normal. In general, letting \( n_{(b,d)i} = n_{bi} \) and \( n_{(d,b)i} = n_{di} \) for two colliding bodies will not satisfy equation 4.3. There are multiple methods proposed in [Huang et al. 2011] to ensure collinearity. All simulations in this report has used the method of averaging the normals with

\[
n_{(b,d)i} = -n_{(d,b)i} = \frac{n_{bi} - n_{di}}{|n_{bi} - n_{di}|}. \tag{4.7}
\]
4. Multi-body solver

4.1.3 Resolving collision

This section will discuss body $b$ colliding with $d$. All arguments made regarding one body can be made interchangeable with the other due to symmetry, and may in some cases be omitted for brevity.

Contact is determined if equation 4.5 is satisfied for any of the bodies using the next nodal velocities $v_{bi}^k$ or $v_{di}^k$. The superscript will be omitted in the following derivation. Note that the center of mass velocity also needs to be calculated using the next nodal velocities. If the bodies are in contact, the collision is resolved by applying the impulse $f_{bi}^*$ and $f_{di}^*$ to each respective body. By virtue of Newton’s third law we demand

$$f_{bi}^* + f_{di}^* = 0. \tag{4.8}$$

The impulse is employed in the direction of the normal as

$$f_{bi}^* = f_{(b,d)i}^* \tag{4.9}$$

and it is applied as a superposition to the grid force,

$$\vec{f}_{bi} = \vec{f}_{bi} + f_{bi}^*. \tag{4.10}$$

The purpose of the impulse force is to calculate a correction velocity $v_{bi}^*$ so that the grid node velocity satisfies equation 4.5. The velocity relate to the force as

$$v_{bi}^* = \frac{\Delta t}{m_{bi}} f_{bi}^*, \tag{4.11}$$

and we update the velocity by

$$\vec{v}_{bi} = \vec{v}_{bi} + v_{bi}^*. \tag{4.12}$$

Now, using the impenetrable condition 4.2 with the corrected velocities $\vec{v}_{bi}$ and $\vec{v}_{di}$ it is possible to calculate the impulse required to deflect the bodies:

$$0 = \vec{v}_{bi} \cdot \vec{n}_{(b,d)i} + \vec{v}_{di} \cdot \vec{n}_{(d,b)i} =$$

$$= (v_{bi} + v_{bi}^*) \cdot n_{(b,d)i} + (v_{di} + v_{di}^*) \cdot n_{(d,b)i} = \{4.3\} =$$

$$= ((v_{bi} - v_{di}) + (v_{bi}^* - v_{di}^*)) \cdot n_{(b,d)i} = \{4.11\}$$

$$= ((v_{bi} - v_{di}) + \frac{\Delta t}{m_{bi}} f_{bi}^* - \frac{\Delta t}{m_{di}} f_{di}^*) \cdot n_{(b,d)i} = \{4.8\} =$$

$$= ((v_{bi} - v_{di}) + \Delta t \frac{m_{bi} + m_{di}}{m_{bi}m_{di}} f_{bi}^*) \cdot n_{(b,d)i}$$

$$\Rightarrow$$

$$f_{bi}^{norm} = f_{bi}^* \cdot n_{(b,d)i} = \frac{m_{bi}m_{di}}{(m_{bi} + m_{di})\Delta t} (v_{di} - v_{bi}) \cdot n_{(b,d)i} \tag{4.13}$$

If no friction is desired the collision algorithm is completed. However, the extension to include friction follows a path very similar to the one above.

For friction, let the tangential part at the contact node be $s_{(b,d)_i}$ and $s_{(d,b)_i} = -s_{(d,b)_i}$. The corresponding impenetrability condition is now instead the no-slip contact condition

$$\mathbf{v}_{bi} \cdot s_{(b,d)_i} + \mathbf{v}_{di} \cdot s_{(d,b)_i} = 0.$$  \hfill (4.14)

Using the same steps as used to reach equation 4.13, equation 4.14 becomes

$$f_{bi}^{tan} \equiv f_{bi}^s \cdot s_{(b,d)_i} = \frac{m_{bi} m_{di}}{(m_{bi} + m_{di}) \Delta t} (\mathbf{v}_{di} - \mathbf{v}_{bi}) \cdot s_{(b,d)_i}.$$  \hfill (4.15)

The tangential force $f_{bi}^{tan}$ is the force required to remove all relative tangential motion, but we do not want this value to exceed what is stated by the Coulomb friction model $f_{friction} = \mu_f f_{norm}$. The resulting expression for $f_{bi}^s$ becomes

$$f_{bi}^s = f_{bi}^{norm} n_{(b,d)_i} + \min(\mu_f f_{bi}^{norm}, f_{bi}^{tan}) s_{(b,d)_i}$$  \hfill (4.16)

from which the velocity is calculated by equation 4.11. The most efficient way to calculate $f_{di}^s$ is to use equation 4.8, which also concludes the contact algorithm.

### 4.2 N-body systems

The biggest hurdle when increasing the number of bodies in the same simulation is how to resolve collisions when more than two bodies collide at the same grid node. Depending on the size of the kernel when rasterizing particles to the grid this can be more or less common; either way it is common enough to motivate a generalized treatment.

When dealing with additional colliding bodies, it is not necessarily clear how to calculate the contact normals nor how to distribute forces along those normals to achieve a realistic result. The first idea would be to try to generalize the initial assumptions used, namely equations 4.2 and 4.3, to account for additional bodies. This proves difficult, as it adds additional unknown parameters but creates no new equations to solve. Instead, it is easier to look at all pair-wise collisions at a grid node and solve for these individually. This section will describe the changes needed to accommodate this principle.

#### 4.2.1 Contact condition

As our idea is to use pair-wise comparisons, the initial assumptions of impenetrability and collinearity still holds. However, due to the additional bodies some calculations are not valid. Most notably,

$$(v_{bi} - v_{di}) \cdot n_{(b,d)_i} \neq (v_{bi} - v_{cm}) \cdot n_{(b,d)_i}$$
4. Multi-body solver

and as a result

\[(v_b - v_i^{cm}) \cdot n_{(b,d)i} \neq (v_d - v_i^{cm}) \cdot n_{(d,b)i}\]

which means that the base-line condition equation 4.5 is no longer valid. Instead, we state as a definition that \(b\) is in contact with \(d\) if

\[(v_b - v_i^{cm}) \cdot n_{(b,d)i} > 0. \quad (4.17)\]

The general idea behind equation 4.17 is as follows: the velocity of \(b\) should be changed if it does not conform to the velocity of the node, defined as the center of mass velocity of all grids, with respect to \(d\). Equation 4.17 is a generalized statement of equation 4.5 despite their similarities, as the latter assumes only two bodies. However, due to the equivalence in equation 4.5 no longer being true, there might exist cases where

\[(v_b - v_i^{cm}) \cdot n_{(b,d)i} > 0, \quad (v_d - v_i^{cm}) \cdot n_{(d,b)i} < 0\]

and it is unclear whether any impulse forces should be applied or not. To ensure that equation 4.18 is being correctly identified for each pair, it seems sensible to relax the contact definition to say that two bodies \(b,d\) are in contact if \(b\) is in contact with \(d\) or \(d\) is in contact with \(b\). Stated as a condition, two bodies are in contact if

\[\max((v_b - v_i^{cm}) \cdot n_{(b,d)i}, (v_d - v_i^{cm}) \cdot n_{(d,b)i}) > 0. \quad (4.18)\]

4.2.2 Resolving collisions

As previously, contact is determined using the next nodal velocity \(v_{ki}^{k+}\) but with equation 4.18 instead of 4.5. When contact is determined, the goal is yet again to correct grid velocities by some impulse force so that they conform to the contact condition. The total impulse for a body \(b\) is now the sum of all pair-wise impulses

\[f_{bi}^* = \sum_{d \neq b} f_{(b,d)i}^*. \quad (4.19)\]

Now, \(f_{bi}^* \neq f_{di}^*\) in general. Newton’s third law instead states that

\[0 = \sum_b f_{bi}^* = \sum_b \sum_{d \neq b} f_{(b,d)i}^* \quad (4.20)\]

For a frictionless simulation the problem turns into an optimization problem of minimizing \(\sum_b |f_{bi}^*|\) given equation 4.18 and \(f_{(b,d)i}^* \cdot n_{(b,d)i} \leq 0\). Solving this problem is both difficult and costly in practice. To simplify the problem we use the assumption that the normals \(n_{(b,d)i}\) are orthogonal. This relaxes the problem to allow for pair-wise comparisons and impulse adjustments independently of each other. Thus, the algorithm for two-body collisions can be repeatedly applied for all pairs at a grid node until all pairs have been resolved.
4. Multi-body solver

4.3 Results

In the work of [Huang et al. 2011] they make a good argument for how the collision algorithm for two bodies perform. We will instead solely focus on the extended algorithm for arbitrarily many bodies.

4.3.1 Colliding spheres

Consider a moving sphere colliding with three stationary spheres positioned each with its center on a vertex of a unilateral triangle as in figure 4.2. The triangle lies in the horizontal plane, here defined as \( y = 0 \). The colliding sphere has been positioned in the middle of the triangle 3 meters above, and is advancing with a speed of 3 m/s. No gravity is applied. Each sphere has a radius of 1 m and has been discretized into 3,911 points each using a uniform grid spacing of 20 cm. The friction has been set to \( \mu = 0.2 \). After 10 seconds the \( y \) values of the initially non-moving spheres and their velocity can be seen in table 4.1.

Figure 4.2: Colliding spheres. A moving sphere colliding with three stationary spheres. Even though all spheres’ particles are separated their grids connect, and in the second picture (top right) there are grid nodes where three grids are colliding.
4. Multi-body solver

<table>
<thead>
<tr>
<th>Sphere</th>
<th>y value [m]</th>
<th>v [m/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>−9.04</td>
<td>1.46</td>
</tr>
<tr>
<td>Red</td>
<td>−8.93</td>
<td>1.45</td>
</tr>
<tr>
<td>Brown</td>
<td>−8.76</td>
<td>1.44</td>
</tr>
</tbody>
</table>

Table 4.1: Colliding sphere data after 10 seconds of simulation.

4.3.2 Four way

Four cubes with side 1 m are positioned symmetrically around the origin as in figure 4.3. The grid spacing is 10 cm, with each cube being discretized into 9 261 points. They all travel with 3 m/s towards the center, no gravity is used. In the second frame only collisions between at most two grids happen at a grid node, however in the third and fourth frame there exist grid nodes where more than two grids are at contact.

4.3.3 Kubb

Two sets of slightly jittered bricks, kubbs\(^1\), are stacked on top of each other by a small distance. See figure 4.4. They are dropped from an average distance of 0.4 and 0.9 meters respectively. The dimensions of one brick is 15x30x15 cm\(^3\), and its discretized to 735 points with a grid spacing of 4 cm. There are a total of 98 separate bodies and a total of 72 030 particles in the simulation. The bricks starts from a stand still, and gravity is applied.

The ground has been converted to a level set. It has a small mound to create for a more interesting simulation (barely visible in the top left image). After 10 seconds the bricks have come to a rest.

4.3.4 Sand brush

One simulation is not restricted to only one type of material. Consider figure 4.5; here we have coupled an elastic brush together with plastic sand. These will interact dynamically in the simulation using our collision algorithm without any extra coupling.

The motion of the brush is determined by the shaft which has been modelled as a boundary. As the boundary impose the condition \( v_k^b = v_k^b \) for all touched grid nodes the elastic material will loosely follow the boundary. The brush is discretized into 3 603 points with a Young’s modulus of 1 MPa and Poisson’s ratio of 0.1 to assimilate a rubbery-like material.

The sand has been modelled using the Drucker-Prager plasticity model as described by [Klar et al. 2016]. It consists of 230 913 particles, and they have been colored

\(^1\)Kubb is a lawn game frequently played in Sweden. The game involves knocking over a set of wooden blocks, similar in shape to the bricks shown in the image.
depending on their initial $y$-displacement to create a sand-like feeling.

4.4 Discussion

We start this section by concluding that we have successfully developed and implemented a contact algorithm that can handle, in theory, any number of colliding bodies.

We are somewhat restricted in what tools we can use to evaluate the correctness of the collision algorithm. Ideally we would like to use energy diagrams and compare the state before collision with the state after. In theory this is possible; at any time we have the exact energy of the system by the sum of kinetic and potential energy (disregarding thermal effects). However, this would not yield satisfactory conclusions for multiple reasons.

Firstly (1), we noted in chapter 3 that the PIC update scheme we use is inherently dissipative, especially for angular momentum. By adding FLIP we alleviate the problem somewhat, however it does not disappear entirely [Jiang et al. 2015]. Additionally (2), in cases where bodies collide with boundary objects we have, by design, introduced energy loss into the system. Lastly (3), we have introduced friction in the system. This would result itself as a thermal energy loss, which, as noted, we do not consider. The correctness of our collision resolution scheme, which is heavily influenced by [Huang et al. 2011], is discussed in length in their paper. Instead, we will resort to arguments of symmetry in our discussion.

Colliding spheres: One of the most prominent artifacts of the contact algorithm can be seen in figure 4.2: seemingly non-colliding bodies are affecting each other. This is the result from our impenetrable condition, equation 4.2. In reality it does not really make sense: grid nodes should obviously be allowed to penetrate each other, but by doing so effectively forces contact to be resolved on a particle level. This proves difficult, as previous simple operations as determining contact and calculating normals instead become unfeasible.

Furthermore, there is a notable discrepancy in vertical displacement due to different velocities, see table 4.1. This error is introduced with the assumption that all contact normals at a grid node are orthogonal, which in practice is extremely rare. It is amplified on our case as we systematically resolve grid node collisions in the same order. That is, given a node with three colliding bodies $b_1, b_2, b_3$, we will resolve these pair wise as $[b_1, b_2], [b_1, b_3], [b_2, b_3]$. We do this as it is a simple way to guarantee reproducibility. Instead one might introduce a randomized approach that changes the resolution order from time step to time step. This should result in a significantly smaller discrepancy.

Four way: A natural result of the particle smearing is that attributes become filtered and less prominent in the Eulerian domain. Especially sharp surface features like corners becomes smooth, which affects normal calculations. A scenario like in figure 4.3 is problematic as it contains many edge-to-edge collision cases.
The setup is symmetric while the simulation is not. This is partly due to the collision resolution scheme as discussed earlier, however inaccuracies is introduced before there are any grid nodes with more than two collision grids. The background grid is thought of as a static continuous grid, but it too is sampled into discrete points. As such, moving the background grid in relation to the simulation setup would potentially yield different outcomes.

**Memory:** One valid concern with the proposed method is regarding its memory footprint. By using one separate grid for every body \( b_1, ..., b_n \), there will effectively need to be \( n \) sets of values at every grid node. What is more, if these bodies are separated spatially and do not collide, only one grid is valid at each point in space. All invalid grids, however, still needs to store valid values at these grid nodes in order to allow for comparisons (invalid values refer to values which the body does not occupy, i.e. \( m_{ba}^k = 0 \)). By this reasoning the memory usage should increase far more than \( n \)-times when increasing the number of bodies. This has efficiently been solved by using a sparse grid representation like OpenVDB. This reduces the memory usage immensely by not having to store invalid values to the same extent. The result is a linear increase in memory, as naturally expected. The goal of the *Kubb* simulation is to show that this method scales well when increasing the number of bodies.

**Material coupling:** The collision algorithm is indifferent to the constitutive model used and is a standalone addition to the MPM framework. As such it can be used in conjunction with any number of different materials, and it gives for an easy coupling of materials to create dynamic scenarios as can be seen by the *Sand Brush* example.

Simulating more than one material type in the same simulation is not something specific to our collision resolution scheme—it is possible in all MPM solvers. However, by introducing separate grids we can ensure that no blending of the materials will occur. As such, running the simulation without our algorithm would result in sand particles sticking to the brush.
Figure 4.3: *Four way.* Four cubes colliding, starting from the initial configuration in the top figure.
4. Multi-body solver

Figure 4.4: *Kubb*. 98 *kubbs* dropped on the ground.

Figure 4.5: *Sand brush*. A elastic brush moving through sand. The closeup reveals how the brush is being kept in place by positioning parts of the brush inside the boundary. The force from the sand affects the particles as they try to ‘escape’ the boundary.
5 Cracks with the Material Point Method

Investigating cracks is interesting for many different purposes. It is extremely useful in the framework of numerical simulation to model material failure, as the richness of data outperforms experimental methods that often are limited to bulk or surface measurements [Bardenhagen et al. 2015]. In computer games or storytelling it provides a certain realism that adds to the user experience.

MPM is a versatile method, and by exerting increasing amounts of stress to a body it will eventually break without special treatment. There are two problems with this:

1. After the material has cracked, the different pieces still belong to the same body and will be rasterized to the same grid. The result will be visible artifacts identical to those outlined in chapter 4. Most significantly—the bodies may merge together again if kept in contact.

2. Art directability of the crack is hard and cumbersome. Weaknesses can be introduced by modifying the material properties at places, but these are often unreliable and create for unrealistic behaviour.

The main problem of simulating cracks is by their discontinuous nature. Computational algorithms in general often rely in some sense on a continuous representation of the underlying material, which may be more or less suited for representing these discrete elements. Our goal will be to address the problem of dynamic material fracturing in the framework of MPM.

The introduction of cracks brings with it a new set of difficulties. For our purposes, these include

- Explicit crack representation. We need a way to describe a crack within the MPM framework, and more importantly we need to be able to track this crack during the simulation.

- Dynamic crack propagation. As we are not interested in static examination of a fractured specimen, we want the crack to be able to propagate freely during the simulation.

As particles are being rasterized to a continuous Eulerian grid, MPM has no inherent way of representing cracks. The rest of this chapter will describe our extension to
the MPM algorithm to allow material fractures.

5.1 Previous work

There have been previous work that generalize MPM to represent cracks by introducing ways to describe discontinuities in the material.

The CRAMP (CRAcks with Material Points) algorithm is a recent addition to MPM ([Nairn 2003]) that has been used to successfully simulate a range of different materials. These include wood, cementitious composites, and explosives [Bardenhagen et al. 2015]. The method introduces cracks in the Eulerian grid by allowing grid nodes to have multiple velocity fields. Particles from opposite sides of a crack will be rasterized to different grids, which is determined by a line-crossing algorithm from the particle to the grid.

In CRAMP a crack is represented as a Lagrangian mesh, which in 2D constitutes of connected line segments and in 3D a polygon mesh, and end points are tracked as massless particles scattered together with the original material. As a Lagrangian property the crack mesh automatically gets integrated into the MPM algorithm and will spatially transform with the body. Having the particles massless implies that no other grid calculations will be affected and they can easily be integrated into the existing framework.

Another approach is to enrich the constitutive model with additional localized deformation [Guiamatsia et al. 2006], [Nguyen 2013]. As in CRAMP, cracks are associated with material points. However, the displacement discontinuity imposed by a crack is instead modeled by using a strong discontinuity approach where the displacement jump is mapped as an enhanced strain field.

5.2 Extended algorithm

Our goal is low computational overhead and easy directability of crack propagation. As such, the CRAMP algorithm is considered too costly, especially in three dimensions. Instead we will focus on a slightly different approach, and follow the steps outlined below:

1. Pre-facture the specimen by splitting the material into different pieces,

2. Glue the pieces together using massless material points and scatter these in the interface between pieces,

3. Define a particle based criteria for when pieces should break apart.

This section will describe the above steps in greater detail.
5. Cracks with the Material Point Method

5.2.1 Pre-fracture

The purpose of the pre-fracturing stage is to provide the possibility to directly manipulate when and where a certain material breaks. By the efforts produced in the last chapter we have created a tool to manage this process: we introduced additional grids for the sole purpose of simulating discrete objects in the same simulation.

If we split the initial body into different pieces we have created a discontinuity at the surfaces, and thus we have the necessary requirement needed to represent a crack. See figure 5.1. This crack is not explicitly defined as in CRAMP. Instead, a crack is implicitly defined as the intersection between particle groups.

![Pre-fracture](image.png)

Figure 5.1: Pre-fracture. The topmost picture shows the initial specimen. It is divided into two parts, as represented by the different colors. The intersection between the groups represent a material crack. The last picture shows glue particles (colored black) scattered between the parts. These will make sure the parts are treated properly as one body if the material has not been broken apart.

5.2.2 Merging grids

By virtue of the particles being split into separate groups, they will each be rasterized to its own grid. The different pieces, as created by the pre-fracturing process, will thus be seen as separate objects by our multi-body solver, and hence it will apply the collision scheme to resolve contact. This is desired in cases where the material has been broken apart, but it must be possible to consider the pieces as a single body. We do this by merging the two grids. We take this to mean that the merged
grid should behave as if the particles of each body had been rasterized to the same grid.

The joint mass of two grids $b, d$ can be found by direct addition,

$$m_{\text{JOIN}}^k = m_b^k + m_d^k.$$

(5.1)

Force is handled in the same manner,

$$f_{\text{JOIN}}^k = f_b^k + f_d^k.$$

(5.2)

Merging the velocities is not possible by direct addition, as they are rasterized as the mass-weighted average. Instead we transfer the velocity via the momentum,

$$v_{\text{JOIN}}^k = \frac{m_b^k v_b^k + m_d^k v_d^k}{m_b^k + m_d^k}.$$

(5.3)

That this equation is valid for the updated nodal velocity $v_{\text{JOIN}}^{k+}$ can be seen by

$$\frac{m_b^k v_b^{k+} + m_d^k v_d^{k+}}{m_b^k + m_d^k} = \frac{m_b^k v_b^k + m_d^k v_d^k + \Delta t f_b^k + \Delta t f_d^k}{m_b^k + m_d^k} = \frac{m_b^k v_b^k + m_d^k v_d^k + \Delta t f_{\text{JOIN}}^k}{m_b^k + m_d^k} = v_{\text{JOIN}}^k + \Delta t f_{\text{JOIN}}^k = v_{\text{JOIN}}^{k+},$$

where equation 3.31 was used. The joint grid now represents the combined body of $b$ and $d$.

### 5.2.3 Glue particles

In order to manage a dynamic fracture scenario it is necessary to encapsulate localized information on whether two grids are connected (and should be merged) or separate (and should be treated as such). We do this by introducing a new set of particles $G$ to all particles $P$, so that $G \subset P$. We denote these particles glue particles and will reference them by a running index $g$.

A glue particle

- is massless, and thus will not affect any grid calculations;
- contains a criteria $c_g$ for the material failure mode. The criteria is basically a value threshold, and the Lagrangian value will need to be transferred to the grid.
- has a radius of influence in which it will affect grid nodes (and will glue grids together by merging them using the operations defined previously).

These particles will be scattered in the intersection between two particle groups, and will trigger a merging of grids when satisfied. If the criteria gets violated for a particle $g$, it will be invalidated and removed from the set $P$ (and $G$).
5.2.4 Crack failure mode

We determine if a node is cracked or not using a force based glue criteria with respect to the relative grid force \( \Delta f_g \). For our purposes we will disregard forces due to compression and we remove the inwards force component with respect to the contact normal \( n^k_{(b,d)i} \). This limits us to only consider tensile and shearing crack forces, which seems plausible considering how glue works. The relative grid force is then

\[
\Delta f_g^k = \left( f_{bi}^k - \min(0, f_{bi}^k \cdot n^k_{(b,d)i}) n^k_{(b,d)i} \right) - \left( f_{di}^k - \min(0, f_{di}^k \cdot n^k_{(d,b)i}) n^k_{(d,b)i} \right).
\]

The state of the node is determined by comparing the size of \( \Delta f_g^k \) with the nodal threshold criteria \( c_i^k \), as acquired by rasterizing the glue particles to the grid:

\[
c_i^k = \frac{\sum_g c_g w^k_{gi}}{\sum_g w^k_{gi}}
\]

The introduction of the weight average in equation 5.5 makes the threshold independent on the number of nearby glue particles. As a result it becomes easier to achieve the desired material break point. Lastly, we can determine if two grids should be merged at a grid node if

\[
|\Delta f_g^k| < c_i^k
\]

is satisfied. If not the grids will stay separate and the multi-body solver will resolve the collision.

5.3 Results

The previous steps has been implemented as an extension to the multi-body solver presented in the last chapter. The following section described the achieved results.

5.3.1 Controlled tearing

The first example aims to investigate if our method is successful in directing how a material cracks. Consider a rectangular brick as in figure 5.2. It is discretized into 35 301 material points, and a small cut-out has been created on one side to create a material weakness that will help initiate the crack. The same simulation with identical material parameters was run three different times, with the only difference being how the material has been pre-fractured. Gravity is used in all simulations.

The left-most specimen has not been pre-fractured at all. As can be seen the material still breaks in the middle as expected, and it has smaller fractures close to the top plate due to high forces close to the boundary. The other two simulations has the
crack interface modified by the pre-fracture process: the middle has a diagonal cut and the right-most has a crack in a zick-zack pattern. All glue particles has a glue threshold of $c_p \in [140, 160]$ N.

Figure 5.2: Controlled tearing. Tearing of a sponge-like material. The red and blue specimens has been pre-fractured.

5.3.2 Internal cracks

Consider a torus as in figure 5.3. It has been pre-fractured into two pieces by a horizontal plane splitting it in the middle and consists of two symmetric halves; one on bottom and one on top. Glue particles has been scattered on the outer half of both the two disks that represent the interface where the bottom and top parts meet. As a result, the inner halves of the two disks constitutes two open internal cracks.

The torus approaches the right wall with a velocity of 10 m/s and consists of 37 100 points. The outer radius is 50 cm and the internal is 25 cm. No gravity is applied.

5.4 Discussion

Controlled tearing: The purpose of this development has been to create realistic looking fracturing. Figure 5.2 clearly shows that it is possible to direct the fracture
process using our technique. Focus has been given especially to make the setup phase of pre-fracturing easy. By equation 5.5 not much thought has to be given to the distribution of glue particles in the crack intersection; as long as all distances to nearby glue particles is less than the size of the kernel the setup will function properly.

Additionally, glue particles as a concept is very flexible. We have used them to transfer a particle property to the grid, and to resolve the crack on the grid. This limits us to the fact that we need the Lagrangian property to be comparable to some grid value. In our case we relate it to the grid force. Instead it is possible to have the crack determination done purely in a Lagrangian frame of reference. The difference is that glue particles would be using calculations derived from the constitutive model to derive a local glue measurement. This could then be compared to some internal threshold $c_g$, and if the threshold gets violated the glue particle gets invalidated. As a result, the logic on the grid level reduces to a look-up to see if there are any nearby glue particles.

At the moment only non-branching cracks is supported by the algorithm. In branch intersections more than two different grids will be connected, and glue particles would need additional information to differentiate between the grids.

**Internal cracks:** The example in figure 5.3 highlights one of the major problems with our method. Collision is resolved using the multi-body solver developed in the last chapter, and the problem of contact being determined prematurely was highlighted. By using the same solver we naturally introduce the same problems here. [Nairn 2003] addresses this problem by introducing an additional step in the collision algorithm. Further development on this technique should investigate whether their approach is applicable here.
Figure 5.3: *Internal cracks.* A fractured torus colliding with a wall. The frames are ordered top to bottom, left to right.
6 Conclusions

We have made an effort to make a clear and concise overview of MPM. Considering the formalism that is required to describe the MPM algorithm, it is somewhat surprising how straightforward the basic implementation is, especially when using an explicit time integration scheme as in our case. The algorithm outlined in section 3.6 provides all necessary parts needed for a functioning MPM implementation.

The basic MPM was extended to allow for arbitrarily many independent bodies inside the same solver. Our work was primarily an extension of the work done by [Huang et al. 2011]. They discussed an alternative algorithm for allowing additional bodies inside the solver, however we could not find a satisfying solution that would conserve momentum. The main problem was due to their idea of resolving grid collisions independently at each grid node. Instead, we kept the core of their collision algorithm intact but changed the resolution scheme to resolve collisions in a pair wise fashion on the background grid. This resulted in a method that scaled significantly better when increasing the number of bodies. A couple of numerical artifacts was introduced, but methods to reduce these inaccuracies was also presented.

The work produced in the multi-body solver was leveraged to develop a novel method of introducing cracks within the framework of MPM. The method drew inspiration from different sources, most notably the CRAcks with Material Points (CRAMP) method by [Nairn 2003], but literature describing this precise scheme has not been found prior to this work. Cracks are not explicitly represented by a Lagrangian mesh, instead they are implicitly defined as the intersection between parts of a body that has been pre-fractured. To indicate where a material is cracked and where it is not, glue particles is introduced. These are scattered in the crack intersection and resembles the Lagrangian mesh as used in CRAMP, however they have fundamentally different functions. The Lagrangian mesh in CRAMP is the de facto crack, and the particle rasterization phase needs to be complemented with a line-crossing algorithm that determines what side a particle is on. Our glue particles are unconnected and does not represent a crack, and they are used by the Eulerian grid by a neighbour look-up to determine if a grid node should be treated as cracked or not. By this notion our method will perform significantly faster than CRAMP. A tearing scenario was presented in chapter 5 to show the applicability of this procedure.
6. Conclusions
Bibliography


45

# A Simulation parameters

## A.1 Contact data

<table>
<thead>
<tr>
<th>Sim</th>
<th>points</th>
<th>$E$ [kPa]</th>
<th>$\nu$ [1]</th>
<th>$\Delta t$ [ms]</th>
<th>seconds [s]</th>
<th>frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merging</td>
<td>17 445</td>
<td>30</td>
<td>0.1</td>
<td>4.2</td>
<td>2.8</td>
<td></td>
</tr>
<tr>
<td>Pool</td>
<td>15 644</td>
<td>50</td>
<td>0.1</td>
<td>6.9</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>Four way</td>
<td>37 045</td>
<td>50</td>
<td>0.1</td>
<td>2.6</td>
<td>4.7</td>
<td></td>
</tr>
<tr>
<td>Kubb</td>
<td>72 020</td>
<td>90</td>
<td>0.1</td>
<td>2.1</td>
<td>9.5</td>
<td></td>
</tr>
<tr>
<td>Sand brush</td>
<td>230 931</td>
<td>353.7</td>
<td>0.1</td>
<td>0.26</td>
<td>38.2</td>
<td></td>
</tr>
<tr>
<td>(sand)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sand brush</td>
<td>3 601</td>
<td>1000</td>
<td>0.1</td>
<td>0.26</td>
<td>38.2</td>
<td></td>
</tr>
<tr>
<td>(brush)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A.1: Simulation details.
A. Simulation parameters
B Appendix: Algebra

This section aims to list some useful mathematical operators.

B.1 Polar decomposition

The polar decomposition splits a square matrix $A$ into unitary rotation matrix $R$ multiplied by an stretch matrix $S$ as

$$A = RS.$$  \hfill (B.1)

B.2 SVD

The singular value decomposition (SVD) of a $m$ by $n$ matrix $A$ is a factorization by

$$A = UΣV^*$$  \hfill (B.2)

where $U$ is a unitary $m$ by $m$ matrix, $Σ$ a diagonal $m$ by $n$ matrix comprised of real numbers, and $V$ a unitary $n$ by $n$ matrix. The diagonal values $σ_i$ of $Σ$ is denoted the singular values of $A$, and are usually taken in a descending order such that $σ_i ≥ σ_j$ for $i > j$.

It is possible to calculate the polar decomposition using the singular value decomposition. By taking $R = UV^*$ and $S = VΣV^*$ we get

$$RS = UV^*VΣV^* = UΣV^*$$  \hfill (B.3)

which is the definition of the SVD in B.2.