Simulating Snow with the Material Point Method

Esther de Jong

August, 2015

Contents

List of Figures

Abstract

This thesis discusses the research and development steps for the simulation of snow with the Material Point Method (MPM). In order to gain an understanding in the technique and mathematics the MPM was researched in detail. All ten steps of the MPM were implemented using C++. First the method was implemented without the collision and implicit update. Next the body-collision for the particles and grid nodes were applied and last the semi-implicit update step was applied. The goal of implementing and understanding all the steps of the MPM was obtained. Test were done with snowballs: letting them drop, throwing them against a wall and throwing a small snowball against a static bigger snowball. The two snowballs interacted and the big snowball was pushed back by the small snowball while the small snowball also created a hole in the big snowball. The collision worked correctly for the grid nodes and particles. Finding the balance between the number of particles, their weight and the size of the grid nodes was challenging and while this was improved the simulations improved too and became more realistic. The results were not completely satisfactory yet and when the snowball was dropped shear and velocity decrease was observed after a few frames. This problem could not be solved within the available time. Although the snow characteristics where not completely reached the Lagrangian particle and Euler grid implementation that is the basis of the MPM method was succesfull.

Keywords: Snow, Material Point Method

Chapter 1

Introduction

This thesis will discuss the development and implementation of the material point method (MPM) for snow simulations. The goal of this project was to replicate the results by (Stomakhin *et al.* 2013). In addition to getting a realistic snow simulation the objective was to see whether improvements and/or extensions can be made. In the first section literature will be reviewed. The literature review will be split into two sections. The first section being the literature on snow simulations. The second section discusses further uses of the MPM method. The second section discusses the development of the MPM for Snow simulations. The simulation was developed in C++ and the particle positions are printed to pointcloud files. These files can be converted to bgeo and loaded into Houdini where they are rendered with a volumetric shader. In this section the improvements and expansions of this project will also be discussed. The last section is the conclusion, where the goals will be reviewed.

Chapter 2

Related Work

In this section the findings of the literature review into snow simulations and the different applications of the material point method (MPM) will be discussed.

2.1 Snow simulation

In the paper by (Muraoka and Chiba 2000) the focus is on creating the shape of a snow cover after or during snowfall using virtual snow particles. They can also simulate the melting of the snow cover. The moisture content of the snow is used to determine how well the snow will adhere to an object. Snow fall and a wind field are also simulated, but they do not take into account any dynamic behaviour of the fallen snow.

To create a snow cover (Wang *et al.* 2006) simulate a height field of the ground, when enough snowflakes deposit a the same site, the height of this point will be increased. A wind field is also created to interact with the falling snow flakes. When the wind is strong enough snowflakes that have already fallen can be moved across the ground which can create a more realistic snow scene, but no snow dynamics like compression is applied.

Instead of falling particles (Fearing 2000) describes a method for simulating fallen snow by shooting particles from the ground. This method

also covers snow flutter in areas that are blocked from above but not from the side. The snow pack is transformed into a set of smoothly joining 3D surfaces. This could be used for the initial set-up for a snow scene, but no dynamics for the packed snow are discussed.

The method proposed by (Foldes and Benes 2007) also simulates snow accumulation and partly melted snow, but it's main focus is for large distant views like mountains. So it can be used for background scenes, but not for snow dynamics.

(Moeslund *et al.* 2005) Also combine a snowing simulation with a simulation to model accumulated snow. They use the falling snow particles to estimate where snow will accumulate especially around objects. The snow on the ground is represented by a triangulated mesh and the snow flakes with particles.

The methods above all describe the accumulation of snow sometimes in combination with falling snow. These methods can provide a good set-up for a dynamic snow simulation, but do not lend themselves for any dynamic simulations like throwing snowballs, people walking through the scene or snow rolling down a hill and interacting with the snow underneath.

In the paper by (Sumner *et al.* 1998) they simulate ground planes with either sand, mud or snow that can be deformed by the impact of rigid bodies. This is done with a height field for the material on top of the ground plane and different properties for the different materials. This paper simulates a specific part of snow dynamics and could be used for fast real time simulations, but for more detailed simulations (Stomakhin *et al.* 2013) would provide more realistic results.

Only (Stomakhin *et al.* 2013) seems to deal with the phase change and slight compressibility of snow, since this paper is the basis for this thesis it will be discussed in more detail in chapter 3.

Snow can be seen as a granular material, like sand, but no sand paper other than (Sumner *et al.* 1998) has specificity simulated snow. Since altought they can both be seen as granular materials their properties and

behaviour are not quite the same. Snow sticks together and is slightly compressible, while sand only sticks together if it's mixed with water.

2.2 MPM

Other application of the MPM are researched to see where the snow simulation could be extended. In their paper (Stomakhin *et al.* 2014) expand upon (Stomakhin *et al.* 2013). They extend their possible materials and focus on phase-change, like melting or hardening of for example of butter or lava. To expand the snow simulation this paper would be a basis for simulating melting snow. Further expansion is done in (Ram *et al.* 2015) where they use the MPM to simulate viscoelastic fluids, foams, and sponges. The use of the MPM for granular materials was already proposed by (Bardenhagen *et al.* 2000), but the application to snow was not mentioned in this paper. Another application is fluid-membrane interaction (York *et al.* 2000). The MPM is applied to model sea ice dynamics or pack ice (Sulsky *et al.* 2007) like large deformations that can be observed in the Arctic, as well as localized deformation, and sharp representation of the ice edge. The MPM is also used for explicit cracks within the model material in 2D (Nairn 2003). In his paper (Więckowski 2004) applies the MPM to large strain engineering problems like granular flow in a silo and plastic forming.

Chapter 3

The Material Point Method

In order to fully understand the mathematical process of the material point method (MPM) concisely presented in (Stomakhin *et al.* 2013) we worked through all mathematical spets in order to get a more detailed description which is presented in this chapter.

The MPM is divided into a Lagrangian and Eulerian part, which are particle and grid operations respectively. The snow is represented by the particles and the grid is used for several calculations after which the results are applied to the particles and the grid is reset for the next step. In figure 3.1 you can see the operations for the MPM method.

The first step is to convert the mass and velocity of the particles to the grid nodes. No grid node values from the previous timestep are used, thus the grid is reset at the beginning of every timestep. To convert the mass of the particles to the grid node the following equation is used

$$m_i^n = \sum_p m_p \omega_{ip}^n \tag{3.1}$$

where m_i^n denotes the mass of the grid node, m_p denotes the mass of the particle, and ω_{ip}^n denotes the weighting function. ω_{ip}^n is calculated using an one-dimentional cubic B-spline for all axis. The values for the

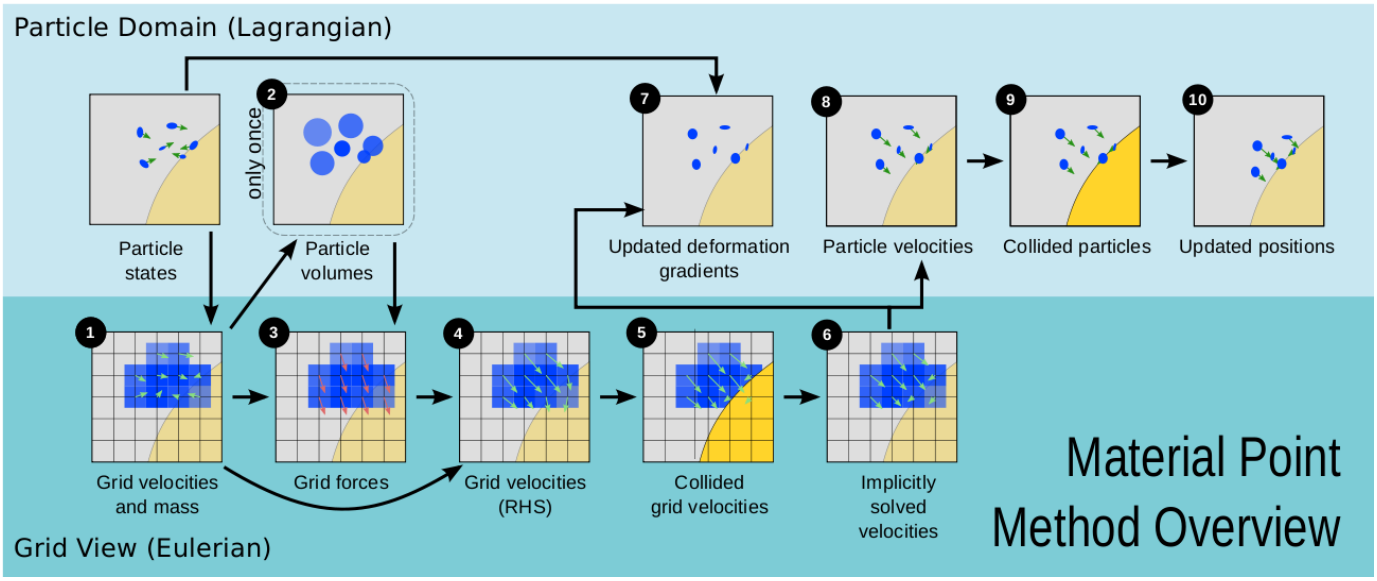


Figure 3.1: *Material Point Method (Stomakhin* et al. *2013)*

different axis are multiplied using the following equation:

$$N_i^h(x_p) = N(\frac{1}{h}(x_p - ih))N(\frac{1}{h}(y_p - ih))N(\frac{1}{h}(z_p - ih)) \tag{3.2}$$

where $N_i^h(x_p) = \omega_{ip}^n$ for a more compact notation, i=(i,j,k) denotes the grid index, $x_p = (x_p, y_p, z_p)$ denotes the evaluation position, h is the grid spacing and

$$N(x) = \begin{cases} \frac{1}{2}|x|^3 - x^2 + \frac{2}{3}, & 0 \leq |x| < 1 \\ -\frac{1}{6}|x|^3 + x^2 - 2|x| + \frac{4}{3}, & 1 \leq |x| < 2 \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

where the input x denotes the distance between the particle and the grid node. x is normalized by h so that the weighting function will extent over $2h$ in all directions. The gradient of the weighting function can be calculated as

$$\begin{aligned} N_{ix}^h &= \delta N(x)N(y)N(z) \\ N_{iy}^h &= N(x)\delta N(y)N(z) \\ N_{iz}^h &= N(x)N(y)\delta N(z) \end{aligned} \tag{3.4}$$

where

$$\delta N(x) = \begin{cases} \frac{3}{2}x^2 - 2|x|, & 0 \leq |x| < 1 \\ -\frac{1}{2}x^2 + 2|x| - 2, & 1 \leq |x| < 2 \\ 0, & \text{otherwise} \end{cases} \tag{3.5}$$

and $\nabla\omega_{ip} = \nabla N_i^h(x_p) = N_{ix}^h N_{iy}^h N_{iz}^h$. This will be used in later steps.

Using the result of formula 3.1 the grid node velocity is calculated by the following formula

$$v_i^n = \sum_p v_p^n m_p \omega_{ip}^n / m_i^n \tag{3.6}$$

where v_p^n denotes the particle velocity.

Only for the first timestep the initial particle densities and volumes are calculated. The density of a cell can be estimated with m_i^0/h^3 this can be weighted back to the particle using the following equation

$$\rho_p^0 = \sum_i m_i^0 \omega_{ip}^0 / h^3 \tag{3.7}$$

The particle initial particle volume can then be calculated as

$$V_p^0 = m_p/\rho_p^0 \tag{3.8}$$

For the next and third step the deformation gradient F is used. F is split in an elastic F_E and plastic F_P part so that $F = F_E F_P$. The elastic part is the reversible type of deformation. It deforms the object, but when it's removed the object can still return to it's original form. The plastic part is the irreversible type of deformation. This deformation is applied after the elastic deformation has reached a certain threshold. Grid forces are calculated using equation

$$f_i(\hat{x}) = -\sum_p V_p^n \sigma_p \nabla\omega_{ip}^n \tag{3.9}$$

where $V_p^n = J_P^n V_p^0$, $J_P = detF_P$, $\nabla\omega_{ip}^n$ denotes the weighting gradient as

in equation 3.4, and

$$\sigma_p = \frac{1}{J_p^n}\frac{\partial \Psi}{\partial F_E}(\hat{F}_{Ep}(\hat{x}), F_{Pp}^n)(F_{Ep}^n)^T \tag{3.10}$$

which can be derived to

$$\sigma_p = \frac{2\mu(F_{Pp})}{J_p^n}(F_{Ep} - R_{Ep})F_{Ep}^T + \frac{\lambda(F_{Pp)}}{J_p^n}(J_{Ep} - 1)J_{Ep}I \tag{3.11}$$

where $J_E = detF_E$, $F_E = R_E S_E$ by the polar decomposition, and

$$\mu(F_P) = \mu_0 e^{\xi(1-J_P)} \text{ and } \lambda(F_P) = \lambda_0 e^{\xi(1-J_P)} \tag{3.12}$$

where ξ denotes the hardening coefficient, λ_0, and μ_0 are the initial Lamé coefficients. The first Lamé coefficient is calculated as

$$\lambda_0 = \frac{E_0 v}{(1+v)(1-2v)} \tag{3.13}$$

where v denotes the poisson's ratio which is set to 0.2, and E_0 the Initial Young's modulus which is set to 1.4 x 10^5. The second Lamé coefficient is calculated as

$$\mu_0 = \frac{E_0}{2(1+v)} \tag{3.14}$$

$\hat{F}_{Ep}(\hat{x})$ can be calculated as

$$\hat{F}_{Ep}(\hat{x}) = (I + \sum(\hat{x}_i - x_i)(\nabla\omega_{ip}^n)^T)F_{Ep}^n \tag{3.15}$$

but in the paper they set $\hat{x}_i = x_i$ for equation 3.9, which results in $\hat{F}_{Ep}(\hat{x}) = F_{Ep}^n$.

The force is then used to update the velocities on the grid nodes for step four with the following equation

$$v_i^* = v_i^n + \triangle t \frac{1}{m_i} f_i^n \tag{3.16}$$

To include the gravity in this equation $f = f + gm_i$ is applied after the force is calculated.

For the fifth step the temporary grid node velocity is updated with

the grid-based body collision. First the grid velocity v is converted to the reference frame of the collision object by

$$v_{rel} = v - v_{co} \tag{3.17}$$

where v_{rel} denotes the velocity of the grid or particle transformed into the reference frame of the collision object, and v_{co} denotes the velocity of the collision object. If the object does not have a velocity $v_{rel} = v$. If the bodies are separating no collision is applied, this is true when

$$v_n = v_{rel}\dot{n} \geq 0 \tag{3.18}$$

If equation 3.18 is false the collision is calculated as

$$v_t = v_{rel} - nv_n \tag{3.19}$$

where v_t denotes the tangential portion of the relative velocity. If the snow has to stick to the object $v_{rel}^{'} = 0$. Otherwise dynamic friction is applied and

$$v_{rel}^{'} = v_t + \mu v_n v_t / ||v_t|| \tag{3.20}$$

where μ denotes the friction coefficent, and $||v_t||$ is the length of v_t. The velocity is then transformed back with

$$v^{'} = v_{rel}^{'} + v_{co} \tag{3.21}$$

With this updated velocity the implicit integration is performed as the sixth step. Initially this was just an explicit step making v^* the new node velocity. The method used for the implicit step is the conjugate residual method, which is outlined in algorithm 1.

This loop continues until the residual r falls below a certain threshold or the maximum amount of loops is reached. At the beginning of the implicit step $\hat{F}_{Ep}$, $\hat{R}_{Ep}$, and $\hat{S}_{Ep}$ are calculated by

$$\hat{F}_{Ep} = (I + \triangle t \nabla v_i^{n+1}) F_{Ep} \tag{3.22}$$

$v_0 = v^*$
$r_0 = v_0$
calculate Ar
$r_0 = v_0 - Ar_0$
$p_0 = r_0$
compute Ar
$Ap_0 = Ar_0$
for $i \leq$ *max number of loops or r small enough* **do**
  $\alpha_i = \frac{r_i^T Ar_i}{Ap_i^T Ap_i}$
  $v_{i+1} = v_i + \alpha_i p_i$
  $r_{i+1} = r_i - \alpha_i Ap_i$
  $\beta_i = \frac{r_{i+1}^T Ar_{i+1}}{r_i^T Ar_i}$
  $p_{i+1} = r_{i+1} + \beta_i p_i$
  $Ap_{i+1} = Ar_{i+1} + \beta_i Ap_i$
  $k = k + 1$
end

Algorithm 1: Conjugate Residual Method

where the velocity gradient ∇v_p^{n+1} is calculated by

$$\nabla v_p^{n+1} = \sum_i v_i^{n+1} (\nabla \omega_{ip}^n)^T \tag{3.23}$$

and $\hat{F}_{Ep} = \hat{R}_{Ep} \hat{S}_{Ep}$ by the polar decomposition.

To compute Ar for the implicit update, first δF_{Ep} is computed by

$$\delta F_{Ep} = \sum_p \triangle t r \nabla (\omega_{ip}^n)^T F_{Ep}^n \tag{3.24}$$

and the derivative of the grid node force is calculated as

$$\delta f = -\sum_p V_p^0 A_p (F_{Ep}^n)^T \nabla \omega_{ip}^n \tag{3.25}$$

where A_p is calculated by

$$A_p = 2\mu(\delta F_{Ep} - \delta R_{Ep}) + \lambda J_{Ep} F_{Ep}^{-T} (J_{Ep} F_{Ep}^{-T} : \delta F) + \lambda (J_{Ep} - 1) \delta (J_{Ep} F_{Ep}^{-T}) \tag{3.26}$$

where $J_{Ep} F_{Ep}^{-T}$ =cofactor (F_{Ep}), and $J_{Ep} F_{Ep}^{-T} : \delta F_{Ep}$ is calculated by

$$J_{Ep} F_{Ep}^{-T} : \delta F_{Ep} = J_{Ep} F_{Ep}^{-T} \cdot \delta F_{Ep} \tag{3.27}$$

$\delta(J_{Ep}F_{Ep}^{-T})$ can be calculated as

$$\delta(J_{Ep}F_{Ep}^{-T}) = \frac{\partial}{\partial F}(J_{Ep}F_{Ep}^{-T}) : \delta F_{Ep} \tag{3.28}$$

To solve δR for A_p, $\delta F = \delta R S + R\delta S$ can be derived to

$$R^T\delta F - \delta F^T R = (R^T\delta R)S + S(R^T\delta R) \tag{3.29}$$

because $R^T\delta R$ is skew symmetric there are only three independent values and δR_{Ep} can be solved as

$$\begin{pmatrix} S_{00}+S_{11} & S_{21} & -S_{02} \\ S_{12} & S_{00}+S_{22} & S_{01} \\ -S_{02} & S_{10} & S_{11}+S_{22} \end{pmatrix}^{-1} \begin{pmatrix} (R^T\delta F - \delta F^T R)_{01} \\ (R^T\delta F - \delta F^T R)_{02} \\ (R^T\delta F - \delta F^T R)_{12} \end{pmatrix} = \begin{pmatrix} \delta R_{01} \\ \delta R_{02} \\ \delta R_{12} \end{pmatrix} \tag{3.30}$$

Which is used to calculate δR as

$$\begin{matrix} 0 & \delta R_{01} & \delta R_{02} \\ -\delta R_{01} & 0 & \delta R_{12} \\ -\delta R_{02} & -\delta R_{012} & 0 \end{matrix} \tag{3.31}$$

so $\delta R_{Ep} = R_{Ep}(R_{Ep}^T\delta R_{Ep})$.

Ar can then be calculated as

$$Ar = r - \beta\triangle t m_i^{-1}\delta f \tag{3.32}$$

where β is 0 for explicit, $\frac{1}{2}$ for trapezoidal, and 1 for backward Euler.

The grid node velocity is then used to update the particles. First of all by updating the deformation gradient and it's elastic and plastic part. The deformation gradient for each particle is updated by

$$F_p^{n+1} = (I + \triangle t\nabla v_p^{n+1})F_p^n \tag{3.33}$$

To see what part of the deformation gradient is Elastic and what part is Plastic the elastic deformation gradient is temporary defined by

$\hat{F}_{Ep}^{n+1} = (I + \triangle t \nabla v_p^{n+1}) F_{Ep}^n$. Initially attributing all the changes to the elastic part of the deformation gradient. Which is then used to compute the singular value decomposition $\hat{F}_{Ep}^{n+1} = U_p \hat{\Sigma_p} V_p^T$ the singular values are then clamped $\Sigma_p = \text{clamp}\ (\Sigma_p, [1 - \hat{\theta}_c, 1 + \theta_s])$. With this clamped value the new plastic and elastic components of the deformation gradient are computed by

$$F_{Ep}^{n+1} = U_p \Sigma_p V_p^T \text{and} F_{Pp}^{n+1} = V_p \Sigma_p^{-1} U_p^T F_p^{n+1} \tag{3.34}$$

$F_p^{n+1} = F_{Ep}^{n+1} F_{Pp}^{n+1}$ should always be true.

The particle velocity is updated using part flip and part pic update using the following equation

$$v_p^{n+1} = (1 - \alpha) v_{PICp}^{n+1} + \alpha v_{FLIPp}^{n+1} \tag{3.35}$$

where $v_{PICp}^{n+1} = \sum_i v_i^{n+1} \omega_{ip}^n$ and $v_{FLIPp}^{n+1} = v_p^n + \sum_i (v_i^{n+1} - v_i^n) \omega_{ip}^n$ with α set to 0.95. A second collision step is performed using v_p^{n+1}.

For the last step the particle positions are updated using the following equation

$$x_p^{n+1} = x_p^n + \triangle t v_p^{n+1} \tag{3.36}$$

In short:

1. Transfer the particle mass and velocity to the grid nodes
2. Compute the initial particle volumes and densities. Only for the first timestep.
3. Compute the forces on the grid nodes.
4. Update the grid velocities with the force.
5. The grid based body collision.
6. The implicit or explicit update.
7. Update the particle elastic and plastic deformation gradient.
8. Update the particle velocities with FLIP and PIC.
9. Particle body collision.
10. Update the particle position.

All steps except the second one are repeated during the following time steps.

Chapter 4

Applications & results

The goal of this project was to recreate the paper by (Stomakhin *et al.* 2013) where they simulate snow dynamics using the material point method (MPM). C++ was chosen as the programming environment, because it was the main programming language of this course and had all the abilities necessary for this project. The main challenge this paper posed was it's mathematical complexity. Understanding and being able to transfer the mathematics into code was the most import part of the process. The visual part was not a main development focus of the project therefore the data was exported and the scene was rendered in Houdini.

4.1 Development

At the start of the simulation the particles are placed within a certain area, and the mass, velocity, and volume of each particle is initialized. To distribute the particles evenly, but randomly in a sphere the flowing equation is used

$$x = r\lambda^{1/3}\sqrt{1-u^2}\cos(\phi) y = r\lambda^{1/3}\sqrt{1-u^2}\sin(\phi) z = r\lambda^{1/3}u \qquad (4.1)$$

where r denotes the radius of the sphere, λ denotes a random value between 0 and 1, u denotes a random value between -1 and 1, and ϕ denotes a random value between 0 and 2π. As can be seen in image

3.1 the MPM consists of 10 individual steps. The primary goal was to implement all steps of the MPM. Initially the collision steps were not implemented for simplicity and the time integration was made explicit instead of semi-implicit for the same reason.

For the first step the grid node mass and velocity had to be calculated. This is done by looping over the particles as described by equations 3.1 and 3.6. To keep the amount of loops for the summation to a minimum the outer loop consists of all particles. Since all particle data is needed but not all grid nodes are active. The grid nodes relative to the particles positions are calculated and the current formula is executed. The grid node position relative to the particle position can be found easily, but the other way around finding the particles within the weighting range of the grid node is a challenge.

```
for i ≤ number particles do
    for i ≤ grid nodes within x range do
        calculate weight and weight gradient for x
        for i ≤ grid nodes within y range do
            calculate weight and weight gradient for y
            for i ≤ grid nodes within z range do
                calculate weight and weight gradient for z
                calculate the final weightfunction
                the summation equation
            end
        end
    end
end
```

Algorithm 2: Summations

The second step is only calculated during the first time step. This step calculates the particle density and with the density the volume with equations 3.7 and 3.8. The density function also contains a summation this time over the nodes, but the same algorithm as for step one is used here. Since it loops over all the active grid nodes.

The third step that calculates the grid forces was one of the more difficult steps and required some more research and the mathematics from the technical report. The grid force is calculated by equation 3.9.

The paper provided equation 3.10 which could not directly be used for the code. The derivative was provided by the technical document and resulted in 3.11. For the decomposition the Eigen library was used. This library does not have a function for polar decomposition, but has one to calculate the singular value decomposition. Which is given as $F_E = U\sigma W^*$. R_E and S_E can be calculated by $R_E = WV^T$ and $S_E = VSV^T$. How to calculate λ_0 and μ_0 was not explained in the paper and some research had to be done to find the correct formula's. Which are described by equation 3.13 and 3.14 in chapter 3.

All values for the fourth step which is calculated by equation 3.16, were already calculated in previous steps so the temporary velocity could be calculated with ease. In the paper (Stomakhin *et al.* 2013) they do not mention the gravity in this equation. Without the gravity force integrated the snowball stays at the same position. The equation

$$f = f_i^{int} + f_i^{ext} \tag{4.2}$$

as described by (Steffen *et al.* 2008). Where f^{int} is the internal force calculated by equation 3.9, and f^{ext} the external force calculated by $f^{ext} = m_i g$.

Step five was not implemented immediately after step four, but later on because it seemed less important at the moment. But without this step it is difficult to show the capabilities of the MPM. The snow was falling, but without the collision the effect of the elastic and plastic forces could not be observed. For the collision first only a ground plane was chosen, because it was the easiest to implement and the effect of a snowball falling apart could be shown. During the final stages the ground plane was expanded to a box so that the grid would not get to big. The formula for the grid and particle collision is the same and the first step is to check is to see if the particle or grid node is inside or on the object's surface. A sticky impulse can be applied to let the snow stick to the surface. Also the object can be given a velocity which is not implemented yet. For both the grid node and particle the input to the collision function is their new velocity.

Step six was the implicit integration. In case the explicit integration is used the temporary velocity became the new velocity. In the code the new velocity was not assigned to be the velocity after this step but after step eight because the new and old velocity where both still needed in steps seven and eight. The implicit integration was implemented last. This was done because the program could work without it, but would improve from the implicit integration. Thus if there was time left at the end it could still be implemented. Next to the computation of the grid force the implicit update was the most challenging. In algorithm 1 the conjugate residual method is described. v_0 is set to be the grid node velocity calculated by equation 3.16. The velocity was deemed converged when the residual value was 0.0001 or smaller. This value was determined by testing values between the range of 0.01 and 0.00001. Where 0.0001 converged between 10 and 30 loops like they mention in the paper (Stomakhin *et al.* 2013). $\hat{R}$ and $\hat{S}$ can be calculated in the same way as $F = RS$ in step three. Ap provided a bigger challenge. The technical report explained it in more detail, but not all. The mathematics was worked out in chapter 3. The matrix calculations for the cofactor and $\delta(J_{Ep}F_{Ep}^{-T})$ had to be coded, because they could not be calculated by the NGL or Eigen library.

The update of the deformation gradient,seventh step, required some more research in the decomposition and the Eigen library. The Eigen library returns the diagonal values for the singular matrix. These values have to be clamped according to the critical compression and stretch. When the elastic value is clamped the plastic value has to be updated with the inverse of the clamped value. The elastic $F_{Ep}^{n+1} = U_p\Sigma_p V_p^T$ times plastic $F_{Pp}^{n+1} = V_p\Sigma_p^{-1}U_p^T F_p^{n+1}$ matrix should be the deformation gradient. This value cannot be compared in the code because the operations by the Eigen library, $U_p \sum_p V_p^T V_p \sum_p^{-1} U_p^T$ returns a matrix close to the identity matrix, but not exactly the identity matrix.

The eight step was to update the particle velocities this was done with a Particle-in-cell (PIC) and a Fluid-implicit-particles (FLIP) part as equation 3.35. This step was quick to implement since the summation parts could be reused from previous code and the equation wasn't that

difficult.

The particle collision uses the same equations as the grid collision. Only now the input is the new particle velocity. Thus once the grid collision was integrated the particle collision could be implemented fairly easily. The particle collision was one of the two functions that was implemented in the particle class itself.

The last step of the MPM was to update the particle positions using the particle velocity. This step was done in the particle class.

Initially the size of the grid spacing was too big. Which caused some unpredictable effects that looked more like a flocking simulation. According to (Stomakhin *et al.* 2013) there should be 4-10 particles per grid cell for the initially packed snow so this was the aim.

Next the code had to be checked for mistakes and missed minuses. The final step was to create a stable simulation. This was done trough trail and error.

Beside the fill algorithm for the sphere an obj file of points can also be imported and simulated as snow.

4.2 Class diagram

In figure 4.1 you can see the class diagram. For the Particle and Grid class only a few of the more important attributes were added to the class diagram to keep the diagram organized. The class MPM calls all the separate steps of the MPM in the Grid class. The Grid class contains the functions of the MPM that require grid information. It also contains a structure for the grid nodes. The particle class contains all the particle variables and the functions for the particle object collision and to update the particle position. All other particle variables are calculated in the Grid class and then pushes to the Particle class. The class ExportParticleData exports the particle positions to a pointcloud (.ptc) file. It only needs to be called once every certain number of loops, depending on the time step. Since the timestep is always smaller than the time between

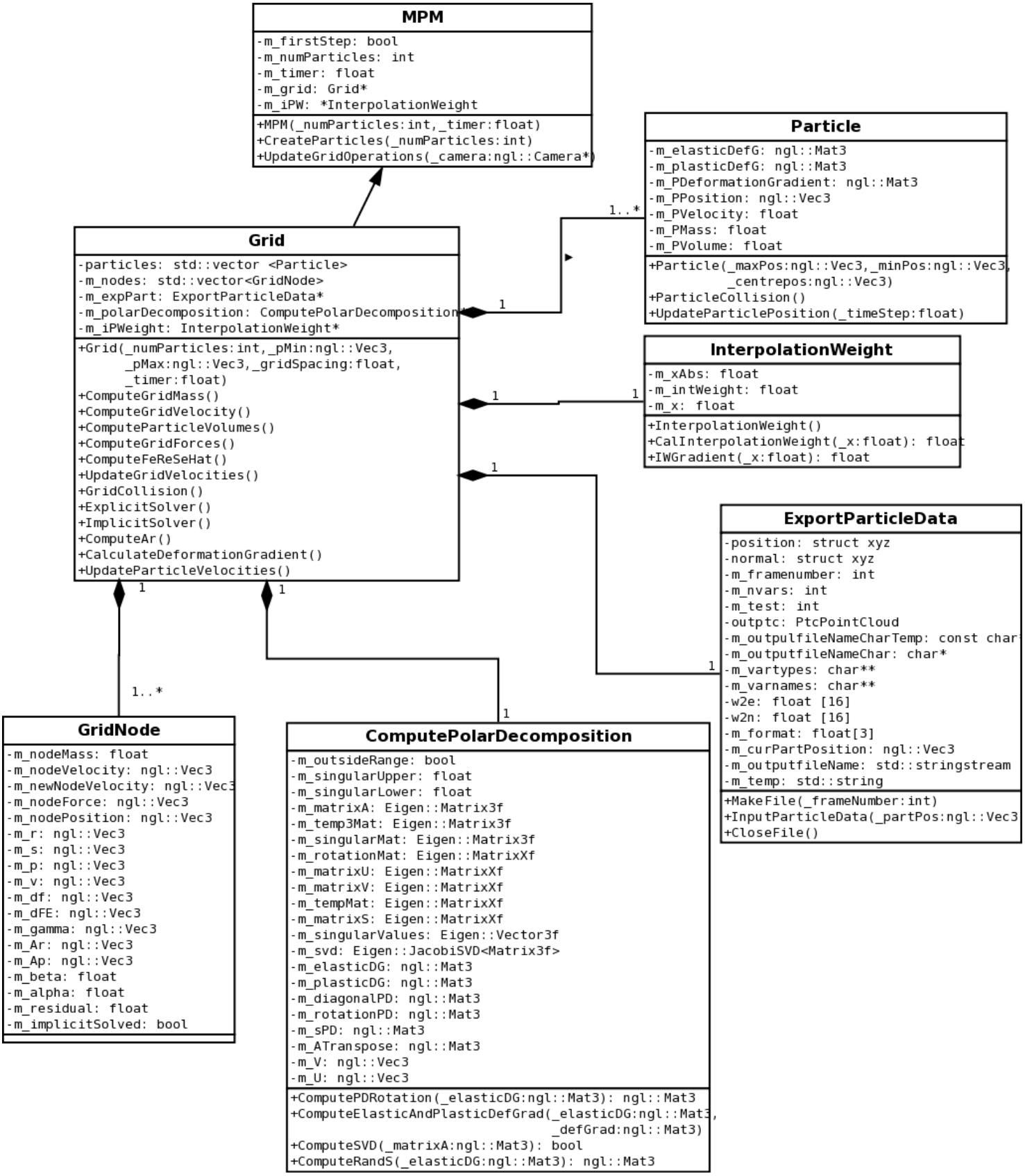


Figure 4.1: *Class Diagram*

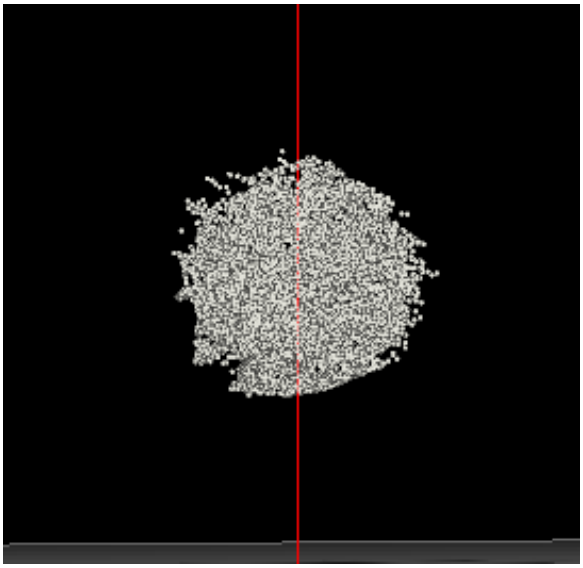
frames. The ComputePolarDecomposition class computes the singular value decomposition with the Eigen library and also uses these values to compute the polar decomposition. The InterpolationWeight computes the interpolation weight and weight gradient for the inputted value. It only computes one axis at a time. It's used by the Grid class to calculate the weight or weight derivative for the inputed value. Because of the parallel programming with openMP the variables in this class had to be localized.

4.3 Problems

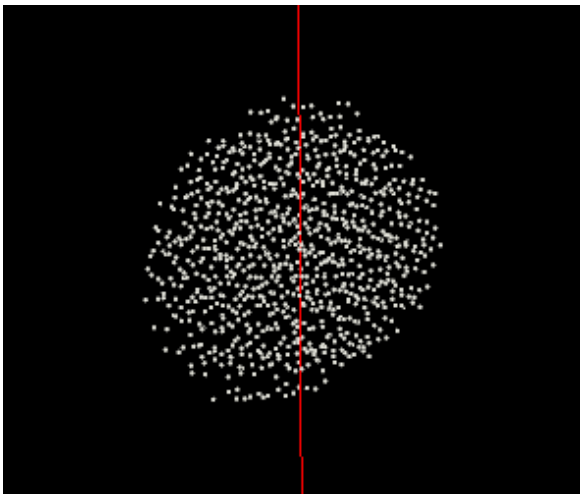
During the development process several problems were encountered. One of the problems was the mathematics, the wrong implementation or a forgotten minus for the force function gave wrong results. To find and prevent these mistakes the code was reviewed and checked several times.

The distance between the grid nodes and for the weighting function was initially set to two meters instead of the length between two grid nodes. The large distance between the grid nodes caused all particles to be influenced by the same grid nodes. The large distance for the weighting function had the same problem, but it was already an improvement because there were a lot more grid nodes and the weighting function caused the particles to be mainly influenced by the grid nodes closest to them. Thus some strange behaviour of the particles was observed. Since there should be between 4-10 particles per grid cell for the initial packed snow.

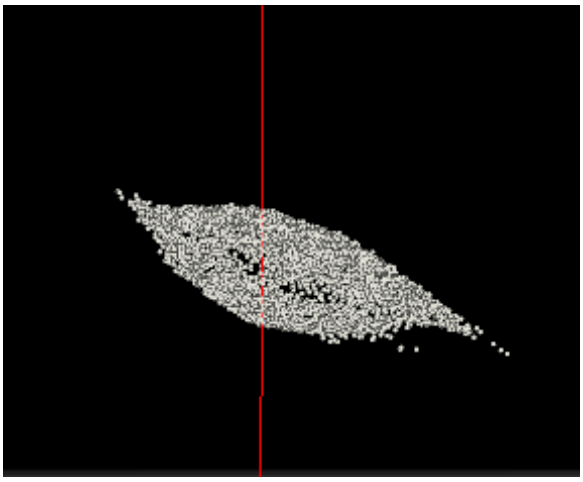
The distribution of the particles in the snowball was first done by a normalized random vector times a random number between zero and the radius. This random function was uniform. So for the sphere the density at the edges was lower than at the centre. This caused the snowball to collapse in on itself. If the density is too high or too low the snowball starts behaving wrong and the simulation crashes. In image 4.2(a) you can see what happens if the particles are packed too densely. The particles are pushed outwards and the surface become unevenly. The



(a) Density too high



(b) Velocity difference



(c) Shear

Figure 4.2: *Problems*

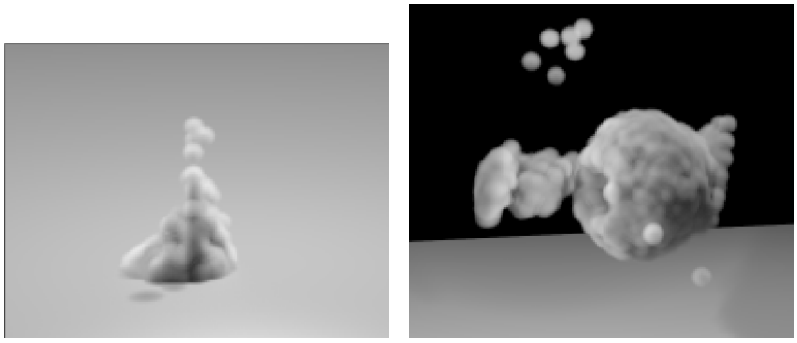
paper stated a high number of particles for a simple snowball around $3x10^5$. Which was a too high number for testing and there was not enough time for the final simulations to be done with this amount.

The last and most difficult problem was to get the simulation stable once everything was implemented according to the paper and finding the right timestep and values for all parameters. This problem was not completely solved before the deadline. A lot of different settings for the parameters were tested and can be seen in the video's. The paper gave a number of parameter values, but not all. For example they mention how many particles they use for the snowball simulations, but not what their weight, size, or the distance between the grid nodes are. The code was too slow to be able to run realtime so all tests were simulated and exported to the pointcloud files. Because this took some time for each simulation the debugging and testing was a slow process. For example for the explicit update a timestep of 10^{-4} proved to be unstable so had to be even higher. When the gravity force was applied to the snowball the snowball was compressed in the y axis and one side seemed to fall faster than the other side. This can be seen in image 4.2(b). To see if the effect would disappear when α for the PIC and FLIP velocity update was set to 1 so the velocity update was only FLIP and the other way around did not yield any results. The only thing was that when falling the velocity would increase, but after a while it would decrease again before hitting the ground. This effect dissapeared when the velocity update was only FLIP. To narrow down the source of the problem σ was set to the identity matrix. This resulted in the snow finally falling to the ground, but it did not keep it's shape and the snow bounced back up see video FLIPandGravity. The Cauchy stress σ consisted of a λ and μ part. When only λ was used for σ the snow collapsed in on itself and looked more like fluid when it hit the ground, see video OnlyMuAndLamda. When only μ was used for σ the shear was observed again, image 4.2(c). Lower stretch or compression values did not seem to have any effect on this. Increasing the weight and adding an initial velocity to the snowball seemed to have the most realistic effect. The snow did not break apart in chunks, but in the paper it was also mentioned that they added a

noise pattern to get the chunky fracture which was not implemented. As a final test a small snowball was thrown towards a big snowball. In the first few frames you can see the big snowball denting in where the small snowball hits it. But when the two snowballs start falling they start to shear together, see the video TwoSnowballSmash.

4.4 Extension

A few possible improvements and extensions include:
The first improvement would be to get the simulation working like in the paper. Since all steps seem to be implemented correctly one of the improvements could be to made a user interface could be added where the user can load in any mesh and set the different parameters so a lot of different simulations can be run. When a mesh is imported the particles should then be assigned the right mass, corresponding with the initial density, and start velocity. Extensions could be to be able to let the snow melt, or combine it with a snowing method. Or extent the MPM so that it can do more materials than just snow. In (Stomakhin *et al.* 2014) the MPM is expanded with a heat function and grid, to simulate heat flow and phase transition. Since the ten MPM steps are roughly the same and an extra step is added to solve the heat equation. The code from this project could be used. At the moment all the snow has to be placed by the user. In the related work several techniques where used to create snow landscapes. Those could be used to transform scenes into snow scenes and then transform the snow mesh into particles so the MPM can be applied for an interactive scene. A snowing simulation which is also based on particles could also be added. Although it does not necessary have to interact with the MPM since it takes a long time for snow to build up.



(a) Snowball drop (b) Snowball smash

Figure 4.3: *Simulation Renders*

4.5 Efficiency

The amount of particles and grid nodes make the program slow, but the parallel programming increased the speed by more than a factor of 2. The disadvantage of this algorithm is that the time step has to be quite small to achieve a stable simulation. With more time the code could be made more efficient and the parallel programming could be applied to every loop.

4.6 Results

In the end several snowballs were simulated. The different simulations were a snowball dropping without any start velocity, a snowball dropping with a start velocity, a snowball being thrown against the wall, and a smaller snowball being thrown against a larger snowball. Although none of them had the desired effect that was set at the beginning of the project several of the characteristic properties of the MPM and snow can be observed. Two of the snowball scene were rendered out with a volumetric shader in Houdini, image 4.3(a) and 4.3(b).

Chapter 5

Conclusion

The main goal of this project was to implement and understand the Material Point Method (MPM) for snow as described by Stomakhin, A., et al. (2013). This goal can be split into the understanding of the mathematics, the ability to implement the mathematics, and to get a realistic snow simulation. The goal of understanding all the mathematics was achieved and discussed in depth in chapter 3. Some of the steps, like the grid node force and the semi-implicit integration were a challenge. These steps were also the most challenging to implement. Once the algorithm, see algorithm 2, that could be used for the different summations, was determined it could be applied in almost every step. This resulted in not having to loop through all the particles per grid node which saves a lot of time.

The goal of a realistic snow simulation was not completely reached. While several of the properties can be observed like when one snowball is thrown towards another snowball the static snowball is pushed back by the force and the snow in the middle where the small snowball hits is pushed back even more. In the scene where no gravity is applied you can see the elastic force returning the snowball to its original shape where the plastic force hasn't broken it. Unfortunately strange behaviour like the shear and a velocity decrease after the snowball has fallen a small distance are still observed. A lot of different solutions were applied, see chapter 4, and some made the simulation more realistic none of them

completely worked. This behaviour seemed to result from the elastic deformation gradient. The details can be found in chapter 4 and while the possible source could be determined all the equations seemed to be applied correctly. Therefore a solution could not be found within the time available. The shear and velocity decrease also occured without the implicit update and while this step uses the deformation gradient FE too it was not responsible for the shear and velocity decrease.

Another goal that was mentioned in the introduction was to expand upon this paper and while the possibilities are discussed in the previous chapter the lack of time did not lend itself for an extension upon the MPM for snow.

So in conclusion the goal of this masters project was partially achieved by implementing and understanding all the steps, but the simulation does not have the desired result yet. The time necessary to get a stable and realistic snow simulation was not foreseen at the start of this project. With a few more weeks and a lot more tests with parameters and doing more code reviews to see where the simulation could be going wrong a final snow simulation could almost certainly be reached.

Bibliography

Bardenhagen S., Brackbill J. and Sulsky D., 2000. The material-point method for granular materials. *Computer methods in applied mechanics and engineering*, **187**(3), 529–541.

Fearing P., 2000. Computer modelling of fallen snow. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 37–46.

Foldes D. and Benes B., 2007. Occlusion-based snow accumulation simulation. In *Vriphys*. Citeseer, 35–41.

Moeslund T. B., Madsen C. B., Aagaard M. and Lerche D., 2005. Modeling falling and accumulating snow. In *Second International Conference on Vision, Video and GraphicGraphics*. Citeseer.

Muraoka K. and Chiba N., 2000. Visual simulation of snowfall, snow cover and snowmelt. In *Parallel and Distributed Systems: Workshops, Seventh International Conference on, 2000*. IEEE, 187–194.

Nairn J. A., 2003. Material point method calculations with explicit cracks. *Computer Modeling in Engineering and Sciences*, **4**(6), 649–664.

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 *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM, 157–163.

Steffen M., Kirby R. M. and Berzins M., 2008. Analysis and reduction of

quadrature errors in the material point method (mpm). *International Journal for Numerical Methods in Engineering*, **76**(6), 922–948.

Stomakhin A., Schroeder C., Chai L., Teran J. and Selle A., 2013. A material point method for snow simulation. *ACM Transactions on Graphics (TOG)*, **32**(4), 102.

Stomakhin A., Schroeder C., Jiang C., Chai L., Teran J. and Selle A., 2014. Augmented mpm for phase-change and varied materials. *ACM Transactions on Graphics (TOG)*, **33**(4), 138.

Sulsky D., Schreyer H., Peterson K., Kwok R. and Coon M., 2007. Using the material-point method to model sea ice dynamics. *Journal of Geophysical Research: Oceans (1978–2012)*, **112**(C2).

Sumner R. W., O'Brien J. F. and Hodgins J. K., 1998. Animating sand, mud, and snow.

Wang C., Wang Z., Xia T. and Peng Q., 2006. Real-time snowing simulation. *The Visual Computer*, **22**(5), 315–323.

Więckowski Z. a., 2004. The material point method in large strain engineering problems. *Computer methods in applied mechanics and engineering*, **193**(39), 4417–4438.

York A. R., Sulsky D. and Schreyer H. L., 2000. Fluid–membrane interaction based on the material point method. *International Journal for Numerical Methods in Engineering*, **48**(6), 901–924.