Utrecht University

MASTER THESIS

ICA-4037324

Analysis of a Material Point Method for Snow

Author:
Thomas BREEKVELDT

Supervisor:
Arjan EGGES
Amir VAXMAN

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in

Game and Media Technology

April 25, 2017

Abstract

A physically-plausible simulation of snow is a challenging task. The method we analyze in this thesis is a Material Point Method (MPM), that has been adjusted to accommodate both the fluid-like and the granular properties of snow. We analyze the simulation in four aspects. One, the noise generated from problems in numerical stability. Two, we analyze the domain in which simulation parameters perform. Three, we reason about the scale of the simulation. And fourth, we present samples collected from runtime analysis. With this, we obtain an understanding of the stability and the performance of the method.

Acknowledgements

I am immensely grateful for the guidance, compassion and patience shown by Arjan Egges to reshape my project as my supervisor at that time. Especially when my family got struck with bad news when I was finaly making progress after a long time. Without him I wouldn't have had the heart to finish this.

I would also like to thank Amir Vaxman for his guidance, patience and understanding during the finalization of this thesis and for taking over the guidance of my thesis from Arjan Egges. Without him I wouldnt have reached this point.

I am also thankfull for the patience and support from my parents throughout my academic years. Thank you all.

Thomas Breekveldt

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Snow may best be regarded as a cellular form of ice, in which the individual ice crystals of snow are bonded together (Petrovic, 2003). These cells contain a mixture of air and water. Two snow lumps with the same density might have different microstructures which results in slightly different reactions under the same conditions.

Snow, as a mixed material, has a non-Newtonian aspect where the material properties change non-linear in relation to the density (Shapiro et al., 1997). These aspects together make it hard to define specific material properties for snow.

The material point method for snow (Stomakhin et al., 2013a) is a novel aproach bringing the Material Point Method (MPM) into this field of simulation while incorporating some of the non-Newtonian aspects of snow.

1.1 Goal

In this work, we present an analysis of the snow simulation method presented by Stomakhin et al. The method simplifies the material snow, which is a mixture of ice, water, and air, ignoring some physical properties of material behaviour and replacing those with observation-based behaviour (Stomakhin et al., 2013a). The goal of our analysis is to scrutinize theoretical weaknesses and establish a domain in which the simulation behaves properly. The analysis we conduct is outlined in the next section.

1.2 Experiments

1.2.1 Numerical Stability

We analyze a weak point in the simulation where a rounding error in a summation to zero can cause a large-enough effect, so that its effects wrap into the next frame and cause a feedback loop.

We show where in the simulation frame this occurs, and how it affects the rest of the simulation.

1.2.2 Parameter Limits

Users require rules on how to tune the parameters of a simulation in order to perform it correctly. Such parameters include different densities, different materials, among others. The original paper presents a list of variables and material properties as a baseline to start a simulation from which tweaks can be made.

In total, the authors define six values in a comparison set of simulations. And only barely explore the simulation domain with those values. We analyze this

baseline and see if these values seem appropriate in a physical sense (measured data), and we define a domain for these parameters on what seems reasonable within this simulation.

1.2.3 Scaling

Sometimes simulation parameters are coupled in the sense that changing one creates an imballance until another parameter is tweaked accordingly. This could be desired, or undesired behaviour. Currently there exists a coupling when trying to increase the simulation resolution in the form of either the grid size or the particle count. We analyze the parameter relations and elaborate on the implications arising from these relations.

1.2.4 Runtime Analysis

We compare empirical runtime data with the runtime complexity.

1.3 Structure

The remainder of this work is structured as follows. Next, in Chapter 2 we elaborate on snow as a material and present related research on some, or all, aspects of snow. In Chapter 3 we outline the method in detail. Chapter 4 we will outline our analysis criteria and perform our analysis. And in Chapter 5 we will present our conclusion.

Chapter 2

Related Works

2.1 Snow

One of the main interactions with snow include the packing, rolling, and sliding of snow. These interactions can be described as material transport of the snow. Where fracturing/flow occurs as the material transport happens. This material transport can alter the topology and density of the snow. In turn, the density of snow is an important property as it is correlated with the rest of its material properties (Judson and Doesken, 2000).

There are many properties of snow such as self-sticking, suspension, saltation, among others. The mass highly varies depending on the type of snow. Fresh snow starts of light but is highly compressible. Wet snow can even exhibit a flow due to the high concentration of water. With all these aspects combined into a single material it is hard to classify and identify material properties. Shapiro et al. (1997) presents an overview of the data available on snow and proposes the creation of a classification system that is more complex than a classification based on density such as given by (Paterson, 1994).

With all these aspects tied into a single material, it is safe to say: snow is a difficult material to simulate. But snow is not unique in this regard. It belongs in the field of rheology with similar materials.

2.2 Rheology

A literal translation of the greek word Rheology is "study of flow". This field of study concerns itself with materials that fail to be categorized as Newtonian fluids or soft-body solids, such as changing material properties under different circumstances.

Among the materials belonging to rheology are snow, lava, molten chocolate, ice cream, and other types of "flowing" food. These dynamics, often transitioning between the states of solid and fluid throughout the material, remain challenging and open problems. As neither a solver for fluids nor a solver for solid/elastic behaviour fits this problem. Often a two-part solver is used instead. One part of the solver to simulate the solids, and another part to solve the fluids. And a layer on top to transition between both solvers. This duality introduces additional complexity and significant computational cost. Stomakhin et al. introduce a method to simulate both the fluid and solid aspects in one solver (Stomakhin et al., 2014). This augmentation of the Material Point Method (MPM) enables them to simulate the incompressibility of fluids, which MPM would otherwise be weak in. They also add a heat equation to solve the local material temperatures, and the consequent change in material properties.

Another material not readily explained as either a fluid or a solid is foam. There are many types of foam, such as soap foam, shaving foam, among others. Yue et al. (2015) introduce a method to simulate foam using the MPM . They use the continuum approximation, as dense foams tend to behave as seamless continuums. One of their novel contributions is the addressing of artifacts that can appear while simulating foam. As foam folds onto itself, pockets of air created are no longer air, void of material, but become part of the material.

2.3 Fluids

One of the aspects that has to be solved in rheology is the aspect of material transport closely related to the fluid aspect of the fluid-solid duality. When material transport happens there's often accompanying topology changes. One method often employed to accomplish this is to simulate the volume as a level-set. Level-sets are a tried and proven method for separating two regions (such as water and air). By using the level-set it is possible to simulate key attributes of fluids such as incompressibility and the maintaining of volume. Taking it one step further than normal level-sets, Losasso et al.(2006) introduce a method using multiple level-sets to simulate multiple liquids, each with different properties. With one of their novel contributions being a method to resolve the surface interactions between each level-set so interaction between each happens.

Another way to simulate fluids is the use of continuum approximation to divide the volume into smaller volumes, particles, to simulate the transport and topological changes. Using a background Eulerian grid, mechanical forces and collision responses are computed. Using these forces inside the grid it is then possible to push those forces from the grid to each individual particle. Jiang et al.(2015) introduce a combination of Particle-in-Cell (PIC) and Fluid Implicit Particle (FLIP) methods, where the PIC method to compute the particle forces from the grid is stable but dissipative, and countering that the FLIP method is designed to remove dissipation at the cost of being unstable. Combining these two allows for a stable method while maintaining the angular momentum of the particles.

2.4 Granular Solids

Granular solids behave roughly as a fluid, where each grain moves freely. An example is how sand flows down a slope until it finds a resting state using friction. This friction, especially the friction between grains themselves, is what makes granular solids different from fluids. Simulating such a granular solid as sand can be done in two ways. One way is to simulate every grain of sand individually as a solid. As the simulation, and thus the number of particles, grows larger, the simulation becomes expensive. Another method for simulation is using continuum approximation. Borrowed from fluid simulation, many small (but larger than the individual grains of sand) volumes are simulated as particles. However, these volumes lack the frictional component to properly simulate a granular solid. Zhu et al.(2005) introduced a method that slightly modifies existing fluid simulations to account for these frictions. This paved a way for more simulations to employ continuum approximations to simulate materials in the field of rheology.

Chapter 3

A material point method for snow simulation

In this chapter, we walk the reader through the method from Stomakhin et al. (2013). We will first give a more detailed outline of the Material Point Method before we completely cover the method.

3.1 Material Point Method

The Material Point Method (MPM) provides an approximation of a material as a continuum, in order to avoid modelling every individual snow flake. Using this approximation, material properties can be approximated as continuous for certain length and time scales, allowing us to simulate at a managable resolution, rather than each individual atom or molecule. The approximation requires an enforcement of the conservation of mass, conservation of linear- and angular-momentum, and the conservation of energy. This can be written in general terms as followed:

$$\frac{D\rho}{Dt} = 0, \qquad \rho\frac{D\boldsymbol{v}}{Dt} = \nabla \cdot \boldsymbol{\sigma} + \rho\boldsymbol{g}, \qquad \boldsymbol{\sigma} = \frac{1}{J}\frac{\delta\Psi}{\delta\boldsymbol{F}_E}\boldsymbol{F}_E^T \tag{3.1}$$

where ρ is density, t is time, $\boldsymbol{v}$ is velocity, $\boldsymbol{\sigma}$ is the Cauchy stress, $\boldsymbol{g}$ is the gravity, Ψ is the elasto-plastic potential energy density, $\boldsymbol{F}_E$ is the elastic part of the deformation gradient $\boldsymbol{F}$ and $J = det(\boldsymbol{F})$.

Depending on the specific implementation these can be realised in different ways. These differences are from the mathematical description of the physical traits a given material has, including the equations that describe stimuli (e.g. deformations) in relation to the material response (e.g. force, stress, energy) they trigger. This is called the constitutive model.

We will further discuss the constitutive model in 3.1.4 after explaining the building blocks of the MPM.

3.1.1 Particles

The main building blocks of the MPM are particles. These particles are used to track mass, momentum, and deformation gradient. Each of these particles is a part of the continuum approximation and thus follows the conservation laws in Equation 3.1. Material properties are also part of this data structure, as they are relevant parameters in the computations. A complete list of particle properties can be found in Table 3.1.

Particles are the carriers of information in the MPM. This information goes through a Lagrangian treatment, that allows for an abstraction simplifying forces

Parameter	**Notation**
Mass (kg)	m
Velocity (m/s)	$\boldsymbol{v}$
Volume (m^3)	V
Elastic deformation gradient	$\boldsymbol{F}_E$
Plastic deformation gradient	$\boldsymbol{F}_P$
Material Properties	**Notation**
Critical compression	θ_c
Critical stretch	θ_s
Hardening coefficient	ξ
Initial density (km/m^3)	ρ_0
Initial Young's modulus (Pa)	E_0
Poisson's ratio	υ

TABLE 3.1: All Particle Properties

Parameter	**Notation**
mass	m
velocity	$\boldsymbol{v}$
velocity change	$\boldsymbol{v}^*$
force	$\boldsymbol{f}$

TABLE 3.2: All Grid Node Properties

into notions of motion among other things. Specifically the discretization of $\frac{D\rho}{Dt}$ and $\rho\frac{D\boldsymbol{v}}{Dt}$ are simplified. However, the computations of derivatives requires a concept of mesh connectivity between the particles. This is achieved by pushing the relevant particle information into a regular (Eulerian) uniform grid.

3.1.2 Grid

The grid is a uniform lattice of points that forms a Eulerian grid. At each point (Node) we store some infromation of nearby particles. The information the grid nodes contain is restricted to the parameters listed in Table 3.2. This is all the information required for the grid computations with one field ($\boldsymbol{v}^*$, the changed speed after updates) used for intermediate storage of values.

The means with which the grid obtains the information from the particles is through a weighting function on nearby particles. This weighting function is a dyadic product of one-dimensional cubic B-splines. The closer the particle is to a grid node, the higher fraction of the particle's space is attributed to that node. With a cut-off point at two cell width's worth of distance away. Note: the grid is uniform, thus a node is equal in size in all three dimensions. The equation to compute the weight of a particle's contribution to a node is given by

$$N_{\boldsymbol{i}}^h(\boldsymbol{x}_p) = N(\frac{1}{h}(x_p - ih))N(\frac{1}{h}(y_p - jh))N(\frac{1}{h}(z_p - kh)) \tag{3.2}$$

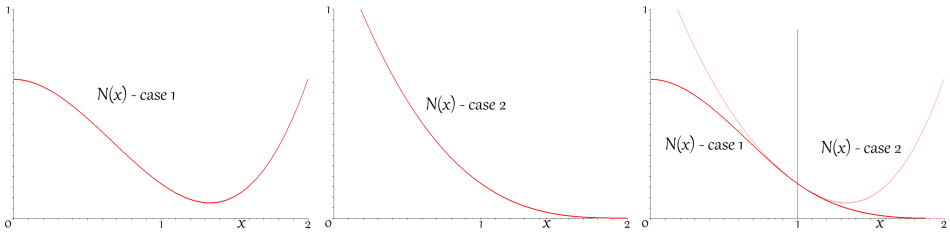

FIGURE 3.1: The weight of a particle as a function of the distance to a grid cell in unit length. With unit length being cell size.

where $\boldsymbol{i} = (i, j, k)$ is the grid index, $\boldsymbol{x}_p = (x_p, y_p, z_p)$ is the particle's position, h is the distance between each node of the grid, and

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

The resulting function is shown in Figure 3.1. For a more compact notation we will use:

$$w_{\boldsymbol{i}p} = N_{\boldsymbol{i}}^h(\boldsymbol{x}_p) \tag{3.4}$$

To interpolate over the grid for our computations, we need the gradient of this weight distribution. The weight function is one dimensional in the relative distance of the particle, done seperately for each axis. To compute this gradient, we take for each axis the original weight function and derive with respect to the distance x, resulting in

$$\nabla N_{\boldsymbol{i}}^h(\boldsymbol{x}_p) = \begin{pmatrix} \nabla N(\frac{1}{h}(x_p - ih))N(\frac{1}{h}(y_p - jh))N(\frac{1}{h}(z_p - kh)) \\ N(\frac{1}{h}(x_p - ih))\nabla N(\frac{1}{h}(y_p - jh))N(\frac{1}{h}(z_p - kh)) \\ N(\frac{1}{h}(x_p - ih))N(\frac{1}{h}(y_p - jh))\nabla N(\frac{1}{h}(z_p - kh)) \end{pmatrix} \tag{3.5}$$

$$\nabla N(x) = \begin{cases} \frac{1}{2}(3|x| - 4) & 0 <= |x| < 1 \\ -\frac{x(|x|-2)^2}{2|x|} & 1 <= |x| < 2 \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

and simplifying the notation to

$$\nabla w_{\boldsymbol{i}p} = \nabla N_{\boldsymbol{i}}^h(x_p) \tag{3.7}$$

This interpolation is done to discretize the $\nabla \cdot \sigma$ terms in the standard FEM manner using the weak form. The interpolation functions of the constitutive model will compute the forces acting on the nodes and update their respective velocities. These updated velocities are then transfered back to the particles using the interpolation weights w_{ip}.

An interesting aspect of this weight function is when sampling is done across the function. Because the sample points are spread exactly unit-length distance (1.0), then the sum of all the points sampled on that curve $N(x)$ is always 1. Because of the distributive properties of multiplication, this summation still holds when the weights are calculated in three dimensions, so that $\sum_{\boldsymbol{i}} w_{\boldsymbol{i}p} = 1$. A similar concept aplies to the weight gradient $\nabla N(x)$ such that $\sum_{\boldsymbol{i}} \nabla w_{\boldsymbol{i}p} = 0$. A small note: with this sampling spread, there are generally four sample points that will

have a non-zero value due to the default case, with a degenerate case where only three points have a non-zero value.

3.1.3 Colliders

At each timestep there are two points in the simulation where there is a check for collision against collision bodies. First, right after the grid forces have been applied to the nodes. Second, on the particles velocities after the velocities have been interpolated from the grid to the particle. Because of the averaging over the nearby gridcells, the particle's velocity is not guaranteed to be collision free. In both cases, they are treated as points for the collision system and the collision is treated as inelastic. The implementation, along with further details, is discussed in Section 3.2.5.

The authors of the original paper propose the use of level-sets (Stomakhin et al., 2013a). Level-sets are datastructures where a uniform grid of points contains the distance to the nearest surface of the object in question. A point collision with a leve-set becomes an interpolation of those distance values to determine a collision.

Generally any collision model that handles large quantities of point-to-x checks fairly well. Preferably where the intersection check is of complexity $O(n)$. For example: for simple objects a combination of half-planes and/or spheres can be sufficient.

3.1.4 Constitutive Model

Now that the components of the MPM have been defined, we describe the core of the MPM, the constitutive model. The mathematical description of the physical traits of a given material is referred to as its constitutive model, and includes the equations that relate stimuli (e.g. deformations) to the material response (e.g. force, stress, energy) they trigger.

The deformation of a physical object can be described as a mapping from its undeformed configuration $\boldsymbol{X}$ to its deformed configuration $\boldsymbol{x}$ by $\boldsymbol{x} = \Phi(\boldsymbol{X})$. This deformation function maps every physical material point X inside the object to its new deformed location x. We define the deformation gradient as $\boldsymbol{F} = \frac{\delta\Phi}{\delta\boldsymbol{X}}$.

A result of elastic deformation is the accumulation of potential energy in the deformed body, referred to as strain energy. The relation between deformation and strain energy is better defined on a local scale. This is done by introducing an energy density function $\Psi[\boldsymbol{\Phi}; \boldsymbol{X}]$ which measures the strain energy per unit undeformed volume on an infinitesimal domain dV around the material point $\boldsymbol{X}$. The energy density function should be expressible as $\Psi[\boldsymbol{\Phi}; \boldsymbol{X}] = \Psi(\boldsymbol{F}(\boldsymbol{X}))$, i.e. a function of the local deformation gradient. Ultimately, the precise mathematical expression for $\Psi(\boldsymbol{F})$ will be the defining property of the material being simulated.

The stress tensor is a descriptor used to describe both interior forces (eg. stress), and the reaction forces on the material's surface. This stress tensor can be used to yield formulas for both force and tension, and is readily computed from the strain energy density definition and the stiffness modi of the object. There are a variety of stress descriptors that can be used for this purpose.

A material might create strain to promote volume conservation, while another material might create more strain when exposed to shear. As a consequence, it is common for the design process for constitutive models to define certain intermediate quantities which are derived from $\boldsymbol{F}$, yet capture the specific traits of the

deformation that the energy or stress values depend on more concisely than the deformation gradient itself.

We next revert the discussion from infinitesimal points to an approximation of the continuum. For each particle p a strain measure is intended to be a quantitative descriptor for the severity of a given deformation, i.e. a way to gauge how far this configuration is from a rest configuration. We define the sest state as $\boldsymbol{F}_p = \boldsymbol{I}$. In multiplicative plasticity theory it is customary to separate $\boldsymbol{F}_p$ into an elastic part $\boldsymbol{F}_{Ep}$ and a plastic part $\boldsymbol{F}_{Pp}$ such that $\boldsymbol{F}_p = \boldsymbol{F}_{Ep}\boldsymbol{F}_{Pp}$. For the next few paragraphs, we omit the notation of p as each variable is in relation to the same particle.

We define the constitutive model for snow used in the simulation in terms of the elasto-plastic energy density function

$$\Psi(\boldsymbol{F}_E, \boldsymbol{F}_P) = \mu(\boldsymbol{F}_P)||\boldsymbol{F}_E - \boldsymbol{R}_E||_F^2 + \frac{\lambda(\boldsymbol{F}_P)}{2}(J_E - 1)^2 \tag{3.8}$$

with the elastic part given by the fixed corotated energy density from (Stomakhin et al., 2013b) and the Lamé parameters being functions of the plastic deformation gradients

$$\mu(\boldsymbol{F}_P) = \mu_0 e^{\xi(1-J_P)} \quad \text{and} \quad \lambda(\boldsymbol{F}_P) = \lambda_0 e^{\xi(1-J_P)} \tag{3.9}$$

where $J_E = det(\boldsymbol{F}_E)$, $J_P = det(\boldsymbol{F}_P)$, $\boldsymbol{F}_E = \boldsymbol{R}_E\boldsymbol{S}_E$ by the polar decomposition, λ_0, μ_0 are the initial Lamé coefficients and ξ is a dimensionless plastic hardening parameter.

Additionally we define the portion of deformation that is elastic and plastic using the singular values of the deformation gradient. We define a critical compression θ_c and stretch θ_s as the thresholds to start plastic deformation (or fracture). Namely, the singular values of $\boldsymbol{F}_E$ are restricted to the interval $[1-\theta_c, 1+\theta_s]$.

This material is elastic in the regime of small deformations as dictated by the $\boldsymbol{F}_E$ dependence in Equation 3.8. When the deformation exceeds a critical threshold (either stretch or compress) it starts deforming plastically. This also affects the material properties in accordance with Equation 3.9, making it stronger under compression (packing) and weaker under stretch (fracture), allowing realstic snow phenomena to be simulated.

3.2 A Material Point Method for Snow Simulation

From here on, we discuss the MPM for snow simulation. We give a rough outline of each step here and refer the reader to Stomahkin et al. (2013) for an in-depth look at each step.

Rasterize particle data to the grid. Information contained by each particle is attributed to the grid cells it influences.

Compute particle volumes and densities. A zero'th-frame only computation where the denstiy and volume of each particle is computed.

Compute grid forces. The strain is computed from the deformation from the previous frame. After which the stress force is computed. Any external forces such as gravity are also accounted for here.

Update velocities on grid. Using the newly computed forces on each node the respective velocity is updated.

Grid-based body collisions. First collision pass. Between each grid node and each collision body and resolve any velocities if needed.

Solve the linear system. An optional step for semi-implicit integration. Explicit integration maintains the current velocity value.

Update deformation gradient. The deformation of a particle is computed based on the gradient of nearby velocities.

Update particle velocities. The node velocities are attributed back to each particle.

Particle-based body collisions. Second collision pass. Now done between each particle and each collision body. Velocities resolved if needed.

Update particle positions. Updating of the particle positions after all velocity modifications are done.

3.2.1 Rasterize particle data to the grid

At the start of the frame, the grid is empty and the information of mass and velocity still resides with the particles. The first step is to transfer information from the particles to the grid. The only information required at this point is the mass and the velocity of the particles.

The mass of grid node at position $\boldsymbol{i}$ at time n is noted as $m_{\boldsymbol{i}}^n$ and is the summation of all the mass contributed by the particles according to the weight function. This is given by:

$$m_{\boldsymbol{i}}^n = \sum_p \left(m_p w_{\boldsymbol{ip}}^n\right) \tag{3.10}$$

where m_p is the mass of particle p and $w_{\boldsymbol{ip}}^n$ is according to Equation 3.4.

Similar to the mass, the velocity of a grid node is the summation of the velocity contributed by the particles. However, to conserve momentum a normalization by the mass is required. Computing the node velocity $\boldsymbol{v}_{\boldsymbol{i}}^n$ at position $\boldsymbol{i}$ with the normalized weighting is done as

$$\boldsymbol{v}_{\boldsymbol{i}}^n = \sum_p \left(\boldsymbol{v}_p^n \frac{m_p w_{\boldsymbol{ip}}^n}{m_{\boldsymbol{i}}^n}\right) \tag{3.11}$$

where $\boldsymbol{v}_p^n$ is the velocity of particle p.

3.2.2 Compute particle volumes and densities

The only data present in the initial configuration of the simulation is a particle's mass and velocity. From these values, also using the grid resolution, we compute the rest state density of each particle. This step is done only once to compute V_p^0, the volume of particle p at timestep 0. The particle volume in later timesteps is computed as a modification of the volume at timestep 0 by $V_p^n = J_p^n V_p^0$ where J_p^n is the determinant of the deformation gradient $\boldsymbol{F}$ at timestep n.

To compute V_p^0, the first step is to compute the node densities at position $\boldsymbol{i}$, $\rho_{\boldsymbol{i}}^0$ using:

$$\rho_{\boldsymbol{i}}^0 = \frac{m_{\boldsymbol{i}}^0}{h^3} \tag{3.12}$$

where m_i^0 is the mass of the node at frame 0, and h is the distance between each node.

Then we compute a particle's density ρ_p^0 by using the same weighting function, but now transfering information from node to particle, as shown in

$$\rho_p^0 = \sum_{\boldsymbol{i}} \left(\frac{m_{\boldsymbol{i}}^0 w_{\boldsymbol{ip}}^0}{h^3} \right) \tag{3.13}$$

And finally estimate a particle's volume at as V_p^0 using

$$V_p^0 = \frac{m_p}{\rho_p^0} \tag{3.14}$$

We store the result to compare future changes in the density with, as this difference affects our material properties.

3.2.3 Compute grid forces

With the mass and velocity information transfered to the grid we can approximate the stress-based forces using the continuum approximation. While there isnt an actual deformation in the grid causing the stress, we can think of the grid node velocity vectors as deformation waiting to happen. If $\hat{\boldsymbol{x}}_{\boldsymbol{i}}$ is the position of grid node $\boldsymbol{i}$, then we define the deformed location by

$$\hat{\boldsymbol{x}}_{\boldsymbol{i}} = \boldsymbol{x}_{\boldsymbol{i}} + \Delta t \boldsymbol{v}_{\boldsymbol{i}} \tag{3.15}$$

given the current velocity $\boldsymbol{v}_{\boldsymbol{i}}$ of the node. If we refer to the vector of all grid nodes $\hat{\boldsymbol{x}}_{\boldsymbol{i}}$ as $\hat{\boldsymbol{x}}$, then the MPM approximation of the total elastic potential can be written as

$$\Phi(\hat{\boldsymbol{x}}) = \sum_p \left(V_p^0 \Psi(\hat{\boldsymbol{F}}_{Ep}(\hat{\boldsymbol{x}}), \boldsymbol{F}_{Pp}^n) \right) \tag{3.16}$$

where V_p^0 is the volume of particle p at frame 0 we computed in Equation 3.14, $\boldsymbol{F}_{Pp}^n$ is the plastic part of the deformation gradient $\boldsymbol{F}$ of particle p at time t^n and $\hat{\boldsymbol{F}}_{Ep}$ is the elastic part which is related to $\hat{\boldsymbol{x}}$ as

$$\hat{\boldsymbol{F}}_{Ep}(\hat{\boldsymbol{x}}) = \left(\boldsymbol{I} + \sum_{\boldsymbol{i}} \left((\hat{\boldsymbol{x}}_{\boldsymbol{i}} - \boldsymbol{x}_{\boldsymbol{i}}) (\nabla w_{\boldsymbol{ip}}^n)^T \right) \right) \boldsymbol{F}_{Ep}^n \tag{3.17}$$

The MPM spatial discretization of the stress-based forces can then be given by

$$-\boldsymbol{f}_{\boldsymbol{i}}(\hat{\boldsymbol{x}}) = \frac{\delta \Phi}{\delta \hat{\boldsymbol{x}}_{\boldsymbol{i}}}(\hat{\boldsymbol{x}}) = \sum_p V_p^0 \frac{\delta \Psi}{\delta \boldsymbol{F}_E}(\hat{\boldsymbol{F}}_{Ep}(\hat{\boldsymbol{x}}), \boldsymbol{F}_{Pp}^n)(\boldsymbol{F}_{Ep}^n)^T \nabla w_{\boldsymbol{ip}}^n \tag{3.18}$$

That is, $\boldsymbol{f}_{\boldsymbol{i}}(\hat{\boldsymbol{x}})$ is the force on grid node $\boldsymbol{i}$ resulting from elastic stresses. When written in terms of the Cauchy stress we obtain

$$\boldsymbol{f}_{\boldsymbol{i}}(\hat{\boldsymbol{x}}) = -\sum_p V_p^n \boldsymbol{\sigma}_p \nabla w_{\boldsymbol{ip}}^n \tag{3.19}$$

$$\boldsymbol{\sigma}_p = \frac{1}{J_p^n} \frac{\delta \Psi}{\delta \boldsymbol{F}_E}(\hat{\boldsymbol{F}}_{Ep}(\hat{\boldsymbol{x}}), \boldsymbol{F}_{Pp}^n)(\boldsymbol{F}_{Ep}^n)^T \tag{3.20}$$

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

From the tech report (Stomakhin et al., 2013b) we obtain the following expression

$$\frac{\delta \Psi}{\delta \boldsymbol{F}_E} = 2\mu(\boldsymbol{F}_E - \boldsymbol{R}_E) + \lambda(J_E - 1)J_E\boldsymbol{F}_E^{-T} \tag{3.21}$$

where $\boldsymbol{R}_E$ is the rotational part of the Polar Decomposition of matrix $\boldsymbol{F}$. Using this we can rewrite the Cauchy stress to be

$$\boldsymbol{\sigma} = \frac{1}{J}\frac{\delta \Psi}{\delta \boldsymbol{F}_E}\boldsymbol{F}_E^T = \frac{2\mu}{J}(\boldsymbol{F}_E - \boldsymbol{R}_E)\boldsymbol{F}_E^T + \frac{\lambda}{J}(J_E - 1)J_E\boldsymbol{I} \tag{3.22}$$

and incorporate that into the stress-based force calculation, resulting in

$$\boldsymbol{f_i}(\hat{\boldsymbol{x}}) = -\sum_p V_p^0\big(2\mu(\boldsymbol{F}_E - \boldsymbol{R}_E)\boldsymbol{F}_E^T + \lambda(J_E - 1)J_E\boldsymbol{I}\big)\nabla w_{ip}^n \tag{3.23}$$

This final equation is what we use to compute the grid forces from the current deformation.

3.2.4 Update velocities on grid

After we compute the stress-based forces acting on each grid node, each node now has a mass, velocity and a force acting on it. We apply the force to the node to compute the node's new velocity. At this point any additional forces, such as gravity, should be applied to the grid nodes too. To update the node velocities we use the equation

$$\boldsymbol{v_i^*} = \boldsymbol{v_i^n} + \Delta t\frac{\hat{\boldsymbol{f}}_{\boldsymbol{i}}^n}{m_{\boldsymbol{i}}} \tag{3.24}$$

where $\hat{\boldsymbol{f}}_{\boldsymbol{i}}^n$ is the total acting force on a node after adding gravity, and other forces, acting on that node n at time t^n.

3.2.5 Grid-based body collisions

Collision is done twice each timestep. In both cases collision is processed in the same manner and all collisions are inelastic. This first time it is done for grid nodes. The second time, later in the simulation, is done for the particles. In both cases we treat them as mathematical points to check for collision against the collider.

For each node it is checked whether it collides with any of the colliders and any collision will immediately be resolved. Regardless of the collider system in place, in case of a collision we obtain the collision normal $\boldsymbol{n}$ from the collider and its object velocity $\boldsymbol{v}_{co}$. First we look at the collision in the reference frame of the collider. The node velocity $\boldsymbol{v}$ is transformed to the relative velocity by $\boldsymbol{v}_{rel} = \boldsymbol{v} - \boldsymbol{v}_{co}$. If the bodies are separating ($v_n = \boldsymbol{v}_{rel} \cdot \boldsymbol{n} >= 0$), then no collision is applied. Let the tangential portion of the relative velocity be $\boldsymbol{v}_t = \boldsymbol{v}_{rel} - \boldsymbol{n}v_n$. If a sticking impulse is required ($||\boldsymbol{v}_t|| <= -\mu v_n$), then we simply let $\boldsymbol{v}'_{rel} = 0$, where the prime indicates that the collision has been applied. Otherwise, we apply dynamic friction, and $\boldsymbol{v}'_{rel} = \boldsymbol{v}_t + \frac{\mu v_n \boldsymbol{v}_t}{||\boldsymbol{v}_t||}$, where μ is the coefficient of friction. Finally, we transform the collided relative velocity back into world coordinates with $\boldsymbol{v}' = \boldsymbol{v}'_{rel} + \boldsymbol{v}_{co}$.

For a sticky collision, where snow sticks to a vertical wall or a ceiling, we set the $\boldsymbol{v}'_{rel} = 0$. This is done unconditionally for collision agains surfaces where this

behaviour is wanted, because the previously outlined Coulomb friction does not handle this case properly.

3.2.6 Solve the linear system

For semi-implicit integration we solve the linear system

$$\sum_{\boldsymbol{j}} \left(\boldsymbol{I}\delta_{\boldsymbol{ij}} + \beta \Delta t^2 m_{\boldsymbol{i}}^{-1} \frac{\delta^2 \Phi^n}{\delta \hat{\boldsymbol{x}}_{\boldsymbol{i}} \delta \hat{\boldsymbol{x}}_{\boldsymbol{j}}} \right) \boldsymbol{v}_{\boldsymbol{j}}^{n+1} = \boldsymbol{v}_{\boldsymbol{i}}^{*} \tag{3.25}$$

where β chooses between explicit ($\beta = 0$), trapezoidal ($\beta = \frac{1}{2}$), and backward Euler ($\beta = 1$). However, for simplicity sake we use explicit integration ($\beta = 0$) simplifying the equation to $\boldsymbol{v}_{\boldsymbol{i}}^{n+1} = \boldsymbol{v}_{\boldsymbol{i}}^{*}$. If instead an implicit method is desired, the linear system above has to be solved for $\beta = 1$.

3.2.7 Update deformation gradient

At this point with the updated node velocities known, we update the deformation gradient of the particles. We compute the new deformation directly from the velocity gradient inside the grid. This velocity gradient is computed as

$$\nabla \boldsymbol{v}_p^{n+1} = \sum_i \boldsymbol{v}_{\boldsymbol{i}}^{n+1} (\nabla w_{\boldsymbol{ip}}^n)^T \tag{3.26}$$

Using this velocity gradient we define intermediate values of the elastic and plastic deformation as

$$\hat{\boldsymbol{F}}_{Ep}^{n+1} = (\boldsymbol{I} + \Delta t \nabla \boldsymbol{v}_p^{n+1}) \boldsymbol{F}_{Ep}^n \tag{3.27}$$

$$\hat{\boldsymbol{F}}_{Pp}^{n+1} = \boldsymbol{F}_{Pp}^n \tag{3.28}$$

so that all initial changes are attributed to the elastic part of the deformation gradient. And the total deformation is

$$\boldsymbol{F}_p^{n+1} = (\boldsymbol{I} + \Delta t \nabla \boldsymbol{v}_p^{n+1}) \boldsymbol{F}_{Ep}^n \boldsymbol{F}_{Pp}^n = \hat{\boldsymbol{F}}_{Ep}^{n+1} \hat{\boldsymbol{F}}_{Pp}^{n+1} \tag{3.29}$$

The next step is to clamp the elastic deformation to the limits defined by the particle's properties critical stretch θ_s and critical compression θ_c. For this we compute the singular value decomposition such that $\hat{\boldsymbol{F}}_{Ep}^{n+1} = \boldsymbol{U}_p \hat{\boldsymbol{\Sigma}}_p \boldsymbol{V}_p^T$ and then clamp the eigenvalues to the permitted range $\boldsymbol{\Sigma}_p = \text{clamp}\big(\hat{\boldsymbol{\Sigma}}_p, [1 - \theta_c, 1 + \theta_s]\big)$. Finally, we shift the remainder of the deformation, if any, to the plastic component using

$$\boldsymbol{F}_{Ep}^{n+1} = \boldsymbol{U}_p \boldsymbol{\Sigma}_p \boldsymbol{V}_p^T \quad \text{and} \quad \boldsymbol{F}_{Pp}^{n+1} = \boldsymbol{V}_p \boldsymbol{\Sigma}_p^{-1} \boldsymbol{U}_p^T \boldsymbol{F}_p^{n+1} \tag{3.30}$$

3.2.8 Update particle velocities

After the grid velocities have been computed, we can atribute the relevant velocity components back to the particles. For this we use a hybrid system of Particle In Cell (PIC) and Fluid-Implicit Particle (FLIP). The PIC method is dissipative, and while the FLIP method is designed to remove this, it is, at times, unstable. Instead of relying on a single interpolation we combine the two as given by C. Jiang et al. (2015)

$$\boldsymbol{v}_p^{n+1} = (1 - \alpha) \boldsymbol{v}_{PICp}^{n+1} + \alpha \boldsymbol{v}_{FLIPp}^{n+1} \tag{3.31}$$

where

$$\boldsymbol{v}_{PICp}^{n+1} = \sum_{\boldsymbol{i}} \boldsymbol{v}_{\boldsymbol{i}}^{n+1} w_{\boldsymbol{i}p}^{n} \quad \text{and} \quad \boldsymbol{v}_{FLIPp}^{n+1} = \boldsymbol{v}_{p}^{n} + \sum_{\boldsymbol{i}} (\boldsymbol{v}_{\boldsymbol{i}}^{n+1} - \boldsymbol{v}_{\boldsymbol{i}}^{n}) w_{\boldsymbol{i}p}^{n} \tag{3.32}$$

3.2.9 Particle-based body collisions

At this point, we perform the second collision check. Due to velocities being attributed back to a particle over a region, it might now have a velocity vector pointing into a collision object. To correct for this we have this second collision check. The collision check is computationally the same as described in section 3.2.5, aside from the velocity $\boldsymbol{v}$ now being the particle's velocity instead of the node's velocity.

3.2.10 Update particle positions

Finally, particle positions are updated using $\boldsymbol{x}_p^{n+1} = \boldsymbol{x}_p^n + \Delta t \boldsymbol{v}_p^{n+1}$. At this point all data contained by the grid has become void and can be cleared, as the method has no dependency on previous frames.

Chapter 4

Analysis of the Method

In this chapter we analyze the method made by Stomahkin et al. to identify the (un)intended working limitations of the simulation.

4.1 Numerical Stability

Trying to represent an infinite series of real numbers in a finite number of bits requires an approximation, generally 32 bits. Most calculations with real numbers result in values that cant exactly be represented by that amount of bits. Therefore results are often rounded to fit into this finite representation. This especially becomes aparant when values of vastly different magnitude are summed and there is not enough precision to correctly subract the two numbers.

We analyze the stability of the simulation when a summation fails to sum perfectly to the expected value (zero in the case of no strain). Specifically looking at the effects from an error rippling through the simulation. This analytical analysis will be done simulating a single frame with only one particle.

4.1.1 Analysis

The potential failure here is a very specific outcome of a summation. However, the values which are summed over can, in some cases, vary greatly in magnitude. Specifically in our case we will sum over one particle which is moving with a velocity of $1m/s$, and is currently free from stress.

Normally, as a particle moves through the grid, it interacts with grid cells in a 4x4x4 area around it as the weight function only has four non-zero values. In the three-dimensional case, this results in a 4x4x4 area. The discrete intervals over the weighting function in Equation 3.3 sum to 1 exactly. However, there exists a degenerate case when a particle is perfectly centered in a grid cell, as mentioned in Section 3.1.2. Figure 4.2 shows this case of Equation 3.3, where the positions are chosen such that only three positions are within the weight function's range. The particle will then only affect a 3x3x3 area because cells beyond the nearest neighbour, two neighbours away are exactly at 2 gridcell widths away. The weight function cuts off at $"< 2"$ and those further values are not selected as seen in Figure 4.2. Suppose that even if the cutoff point was $"= 2"$, their values would be exactly zero and thus the cell would not contribute to the simulation regardless.

The case we want to analyze is not this degenerate case, but an extremely small offset from this position. When we shift the particle an extremely small distance to either side one of the grid cells that was exactly on the edge of the domain of the weight function is now within this domain. This grid cell will have a very small weight value as the weight function rapidly approaches zero near the

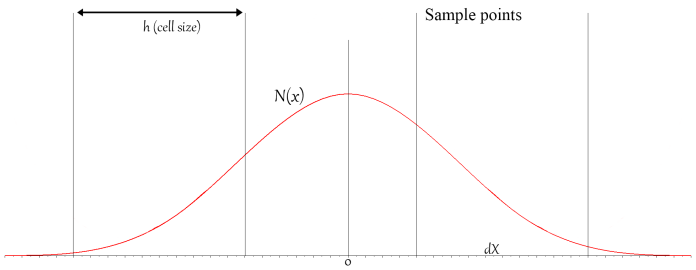

FIGURE 4.1: The weight of the particle-distance function to a cell, sampled with four points, separated by the grid cell size h

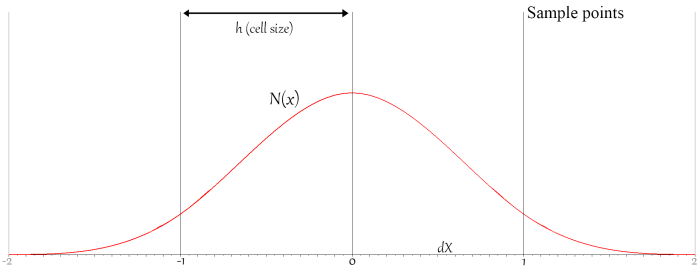

FIGURE 4.2: The weight of the particle-distance function to a cell, sampled with three points, separated by the grid cell size h.

input value of 2. In turn, because of this low weight value, those cells also have a significantly lower mass and velocity contributions from Equation 3.10 and 3.11.

This comes back when we try to compute the velocity gradient in Equation 3.26 which helps us compute the strain. Similarly how the intervals of the weight sum to 1 the weight gradient sums to 0. Because we're only looking at a single particle all nodes share the same velocity. And because of this the expected value of the velocity gradient is the same as the weight gradient, namely 0.

Here's where the empty sum problem arises. In the multiplication of the velocity and the weight gradient (both being extremely small for those grid cells barely within the range) the difference in magnitude compared to normal values start to exceed floating point precission. The resulting summation will therefore not be a zero matrix but instead will be a non-zero matrix containing some rounding errors.

This non-zero value, or rather error, will be interpreted by the method as the pressence of strain. The method will then incorporate this into a tiny deformation gradient in Equation 3.29. These small error values might be deemed of insignificant magnitude. The resulting force computed from this in Equation 3.23 will also be relatively small. However, these forces are generated without respect to the cells' mass. The application of the force does use the mass to update the velocity $v = \frac{f}{m}$. Meaning that in Equation 3.24, with our extremely small mass values, the velocity will become an unrealistically large value.

This large velocity error will compound back into the velocity gradient calculation using Equation 3.26. And at this point we've come full circle, creating a feedback loop from the initial error.

Parameter	Notation	Value
Critical compression	θ_c	2.5×10^{-2}
Critical stretch	θ_s	7.5×10^{-3}
Hardening coefficient	ξ	10
Initial density (kg/m^3)	ρ_0	4.0×10^2
Initial Young's modulus (Pa)	E_0	1.4×10^5
Poisson's ratio	ν	0.2

TABLE 4.1: The only parameter values given in the original paper.

4.1.2 Conclusion

The feedback loop created here adds energy to the system, rather than dissipate some of the existing energy. This is never a good thing for the stability of a simulation. Even if the force generated happens to counter the current velocity vector of a particle, the presence of this still increases the strain.

This issue arises when the mass in grid cells is extremely small, such that any erroneous force generated has a noticable effect. Our example with only a single particle exacerbates the issue. When there are multiple particles these cases average out among the particles and because there is more mass present any erroneous forces have less impact. But, there will always exist grid cells at the exterior of large particle volumes with small mass values. This is where this behaviour might still appear.

A solution might be, for the specific case of absence of stress, to clamp the velocity gradient to zero for values close to zero. Or otherwise clamp the deformation gradient to the identity matrix for tiny deviations from the identity matrix. The difficulty in this is determining the appropriate values to start the clamping at, as sometimes the difference between just a small value or an erronous value is hard to tell.

4.2 Parameter Limits

A number of parameters are listed within the paper, and a few simulation examples exists for different values of these parameters. However, the actual domain of these parameters is largely undocumented. There exist some deviations listed in their experiment section. Nevertheless, the extend to which parameter variations still create reasonable behaviour is not entailed.

With this analysis we try to define the limits to which the parameters remain functional. The starting point values, as suggested by Stomakhin et al. (2013), are given in Table 4.1. We will use these values as a starting point for our analysis.

4.2.1 Critical Compression and Critical Stretch

These parameters define the maximum force to be generated by the simulation. Using the default values given in Table 4.1 we modify the deformation gradient to produce the largest elastic strain possible. For the full breakdown of the computations done we refer the reader to Appendix A.

$$\boldsymbol{F}_E = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & 0.975 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 0 & 0.975 & 0\\ 0 & 0 & 1\end{bmatrix} \tag{4.1}$$

When running this value through the equations we obtain the strain.

$$\begin{aligned}\sigma &= 2\mu(\boldsymbol{F}_E - \boldsymbol{R}_E)\boldsymbol{F}_E^T + \lambda(J_E - 1)J_E\boldsymbol{I}\\ &= \begin{bmatrix}-947.91667 & 0 & 0\\ 0 & -3791.6667 & 0\\ 0 & 0 & -947.91667\end{bmatrix}\end{aligned} \tag{4.2}$$

With this strain affecting particle p, we compute the combined forces acting on the grid cells this particle affects. Solving for all 27 grid cells, of which only 9 hold a relevant force, we obtain:

$$f = -1907.83V_p \tag{4.3}$$

From this point we also reduced the problem to the single dimension of the y-axis, and ignore any components of forces along the other axis, as those forces are not fighting gravity.

We do a similar thing for the gravity acceleration to obtain a relation between this acceleration and the particle's mass with $f = m \cdot a$. The acceleration here is the gravity concstant $a = -9.81$. While keeping the mass as a variable we scale it down by the fraction of mass that is already colliding with the floor ($\frac{1}{6}^{th}$ of the mass) for which the simulation explicitly sets those forces to zero.

$$\boldsymbol{f}_g = m_p \cdot (1 - \frac{1}{6}) \cdot -9.81 \tag{4.4}$$

Simplifying to

$$\boldsymbol{f}_g = -8.175 \cdot m_p \tag{4.5}$$

We can then compare these forces to obtain a relation between mass and volume, in other words density. This relation is an indication at which point the stress force is able to negate the force of gravity.

We obtain a density of $0.004285kg/m^3$ as the tipping point. A higher density means that gravity wins, a lower density means this maximum stress wins.

Conclusion

This value is far from the listed 400 kg/m^3 as initial density. This could mean this method is not assumed to simulate the rest state, but it is expected for the simulation to cut off quickly after the major deformation has been simulated. e.g., the initial deformation after an impact, but not the settling of the particles on the ground.

When referencing the video footage from the paper it neither confirms nor denies the hypothesis that the simulation has to be cut off. Because most animation fragments are so short, there is no indication of this computed behaviour. The exception to this footage might be the aftermath of the collision between the snow castle and the cannonball where some slight sinking / collapsing of snow takes

place near the end of the video fragment (Jang, 2014). However, this is only one poor example and thus inconclusive.

The continued work of Stomakhin et al. (2014) mentions the following: "MPM methods do not handle arbitrarily incompressible materials." and improve upon MPM by adding a Chorin-style projection technique to resolve the incompressibility problem. From this perspective, it might be impossible for MPM to maintain snow's correct volume upon rest state.

In the case where this hypothesis is true, it is not necessarily a bad thing, but expectations and setup of simulation has to be tailored to suit this simulation. It would mean only high deformation simulations make sense.

4.2.2 Hardening Coefficient

This parameter comes in to play when enough stress is applied on a particle so that part of the stress is pushed into plastic deformation. This plastic deformation then causes a modification of the Lamé coefficients. The hardening coefficient affects the rate at which the Lamé coefficients change. A plastic deformation of compression results in higher Lamé coefficients. This increases stress forces generated from a particle's elastic strain. The reverse is also true where a stretching deformation lowers the Lamé coefficients resulting in lower stress forces.

The equation that gives us the modified Lamé coffiecient is given as

$$\lambda(\boldsymbol{F}_P) = \lambda_0 e^{\xi(1-J_P)} \tag{4.6}$$

where λ_0 is the initial Lamé coefficient, ξ is the dimensionless plastic hardening parameter and J_P is the determinant of the plastic deformation gradient. Basically $e^{\xi(1-J_P)}$ acts as a scalar of the Lamé coefficient, so we will focus on that part of the formula. In Figure 4.3 we have plot for 4 different values of ξ along with $\xi = 10$ as per given in the original paper.

We observe that if ξ increases there is a significant increase in the value of the Lamé paremeters when J_P is less than one. With larger Lamé coefficients there will also be a larger strain measure as σ will have larger result value in Equation 3.23. Alternatively, as ξ approaches zero, the entire effect of the modification of the Lamé parameters dies off keeping the Lamé parameters near constant. If $\xi = 0$, and the Lamé parameters no longer change upon plastic deformation, part of the non-Newtonian behaviour of snow is removed.

Conclusion

The lower bound of the hardening coefficient seems unimpactful, as it shuts down the modification of the material properties, so we propose a lower bound of $\xi > 0$. Looking at the upper bound, this parameter scales the outcome exponentially due to the positioning of this parameter in the function. We therefore propose an uperlimit around $\xi = 15$.

The default value of $\xi = 10$ here seems a reasonable value with the impact it has.

4.2.3 Initial Density

The initial value given for the density is 400 kg/m^3. Using definitions of snow this falls into the category of firn which is a highly dense form of snow consisting of snow packed together from multiple seasons. (Paterson, 1994)

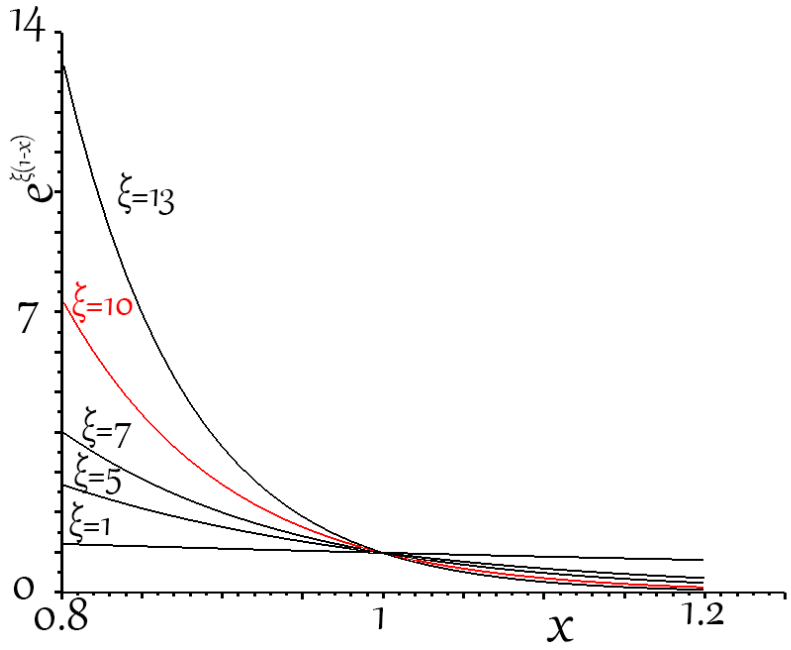

FIGURE 4.3: A plot of different values for the hardening coefficient ξ in the function $y = e^{\xi(1-x)}$. Where x is $det(\boldsymbol{F}_P)$. The baseline using the given value for ξ is shown in red.

Freshly fallen snow has a density in the range of 50-100 kg/m^3 (Judson and Doesken, 2000). And the classification of snow in the actual 400 kg/m^3 is a very dense snow called Firn (Paterson, 1994).

Finally, the threshold of the density snow can self-support itself in this simulation environment was established at 0.004285 kg/m^3 in Section 4.2.1.

Conclusion

The creation of a snowball usually compresses snow when the ball-shape is formed. This might be why a higher density than fresh snow has been chosen. Though, the choice for this density has not been argued for in the paper.

Another option might be that the authors chose for visual compelling values, rather than exact realistic values.

From the implementation perspective of the simulation, the mention of the initial density variable is questionable, because the simulation arrives at a density value for each particle after the first frame. Thus listing this as a simulation parameter when it is the result of computations is illogical.

However, the density does describe a relation between multiple parameters. Defining the density means that when the grid resolution changes, the particle masses have to be changed accordingly to keep the density the same. However, it is near impossible to set the density itself before the simulation starts. Not without reverse engineering the frame zero computations and set the particle masses algorithmitically, or just trial and error from the user until proper frame zero values appear. In other words, defining this value on its own does not translate well into actual properties in the simulation. We will elaborate on this in Section 4.3.

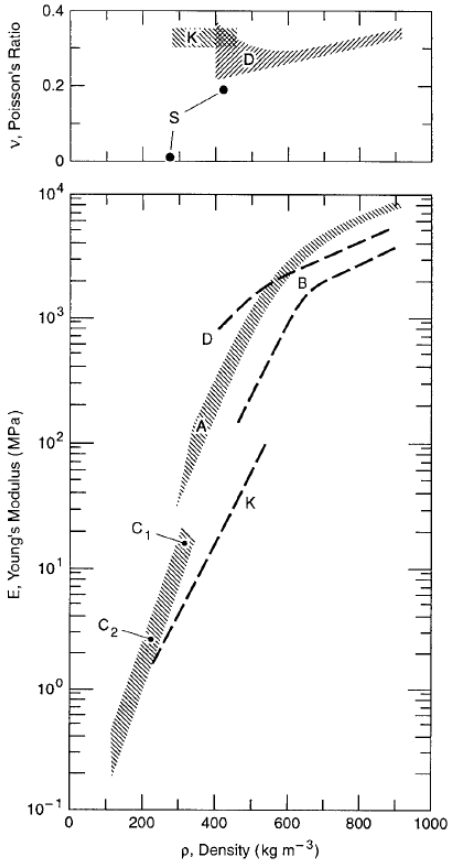

FIGURE 4.4: Young's modulus and Poisson's ratio vs. density for dry, coherent snow. From figure B1 in L. H. Shapiro et al. (1997)

4.2.4 Young's Modulus and Poisson's Ratio

Mechanical studies of snow have shown these material properties depending on the density of the snow. This simulation similarly modifies the material properties as there is deformation. Most likely accompanied by a change in density.

However the simulation still needs a value for the initial configuration to work from. These standard values given in the original paper are listed in Table 4.1. The Poisson ratio $\nu = 0.2$ is similar to the mechanical study of (Shapiro et al., 1997).

The Young's modulus $E_0 = 1.4 \times 10^5$ is quite different from the $E = 4.0 \times 10^3$ (Shapiro et al., 1997).

Conclusion

The Poisson ratio falls exactly into the measured range at the lower end. Observing Figure 4.4, we set a suitable domain mimicking physical values to roughly be $\nu = [0.2, 0.35]$. And the domain for the Young's modulus as $E_0 = [1.4 \times 10^3, 1.4 \times 10^5]$. The difference between the Young's modulus in the simulation and the physical properties is too large to make an accurate domain assumption based on physical properties. We do have the starting value from the original paper and the observation that the Young's modulus drops several orders of magnitude as the density drops in the physical analysis.

4.3 Scaling

In a previous section we highlighted the reliance of the size of the stress force comparing to external forces being dependend on the density of the particles. In this section we point out a myriad of interweaved dependencies that alter the density if one aspect is changed. We perform a numerical analysis to highlight an unwritten coupling of parameters and their interaction with eachother.

4.3.1 Analysis

Making A Particle Heavier

As the mass of a particle increases, but it volume remains unchanged, the density increases. The same strain affecting the particle (ie. the maximum elastic strain) will have less of an effect because turning the strain into a stress force bears no correlation to a particle's mass.

The fact that a heavier particle moves slower is a normal concept in $f = m \cdot a$. The same force on a heavier object is a lower acceleration.

Adding Particles

When more particles are added there are more particles occupying the same space. With the grid volume remaining constant, an increase in particles equates to a decrease in their individual volume.

We correct the particle's mass after more particles have been added. The mass correction is inverse to the change of the volume the particles occupy. So each particle maintains the same density. With this correction, the combined stress force from all particles then remains the same. And, this stress force acting on the combined mass of the particles gives the same resulting velocity as before.

Changing Grid Cell Size

When the grid cell size is changed between simulations, the amount of volume encompassed by a 4x4x4 area of grid cells changes. This in turn changes the total volume a particle can occupy. Thus also changing the total volume of the particle and thus the density of said particle.

4.3.2 Conclusion

As shown in Section 4.2.1, the density of the particle maintains a balance of the size of the resulting particle velocities. This is because the stress force is only directly affected by the particle's volume, and external forces are only affected by the node's mass. Keeping a strict balance between these two is paramount to maintaining the correct magnitude of stress forces, and in turn resulting velocities.

To keep both of these forces in the same order of magnitude, a balance between the volume and mass has to be maintained following the set density. The interweaving of how the density changes with the change of one of these three aspects is what makes a clean setup tricky.

4.4 Runtime Complexity

To establish the runtime complexity an implementation of the simulation was made. The data presented in this chapter was created by running this implementation on a simple desktop computer.

A couple steps outlined in Section 3.2 were combined in the process. The final algorithm steps are as follows:

- **Computing Sigma** For each particle the cauchy stress tensor is computed as per Equation 3.22.
- **Computing Cell Mass, Velocity and Forces** For each particle the mass is distributed over its nearby grid cells using Equation 3.10. The same is done for the velocity at the same time Equation 3.11. We delay the normalization of the velocity until the next step. At the same time we compute the stress force acting on this node due to this particle using its cauchy stress tensor in Equation 3.23.
- **Updating Velocities** For each grid cell we normalize the velocity with the node's mass. Then aply the stress force with Equation 3.24. And finally check for and handle collisions as described in Section 3.2.5.
- **Updating Particles from Grid** For each particle we compute both the particle's velocity at $t + 1$ with Equation 3.31 and the velocity gradient with Equation 3.26. After obtaining that particle's velocity gradient we compute the deformation gradient of the next timestep as in Section 3.2.7. And right before updating the particle's position we check for and handle collisions. This is all done in a single loop.

4.4.1 Analysis

Here we perform an analysis of the runtime complexity. We first define our variables used in this section: P,N, and C. Where P is the particle count, N is the node count and C is the collider count.

Using the steps mentioned above:

- **Computing Sigma** Loops over all particles once. $T(P) = O(P)$.
- **Computing Cell Mass, Velocity and Forces** Loops over all particles once. $T(P) = O(P)$.
- **Updating Velocities** Loops over all grid cells once and performs a collision check for each cell. $T(N, C) = O(N * C)$.
- **Updating Particles from Grid** Loops over all particles once and performs a collision check for each particle. $T(P, C) = O(P * C)$.

The final complexity is $T(P, N, C) = O(P * C + N * C)$.

From our implementation we have gathered the data in Figure 4.5 and 4.6 . The simulation was implemented as a single threaded CPU simulation and run on an i5-3570K processor.

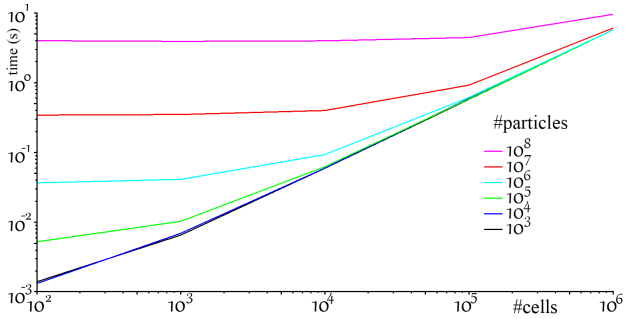

FIGURE 4.5: Our measured runtime data when increasing the cell count with varying starting particle counts.

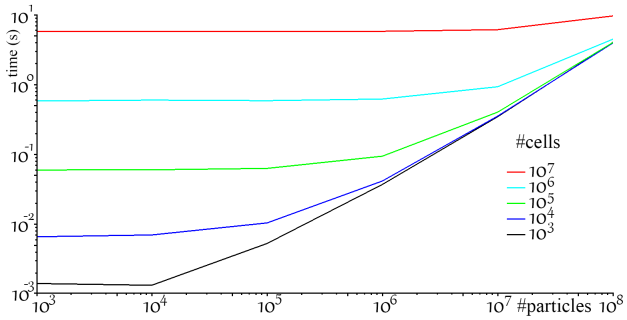

FIGURE 4.6: Our measured runtime data when increasing the particle count with varying starting cell counts

4.4.2 Conclusion

When running the simulations we've kept the collider count C the same through all simulations. We observe that the measured data corresponds to the big-O notation of $O(P * C + N * C)$ simplified to $O(P + N)$ due to C being constant.

The measured data coincides with the theoretical complexity with the linear trend showing on the double log-scale. We conclude that the simulation indeed scales linearly with respect to the particle count and grid cell count.

Chapter 5

Conclusion

5.1 Summary

The objective of this thesis was to analyze the method created by Stomakhin et al. (2013) and create an understanding of simulation possibilities within the simulation.

To analyze the method, we studied the method implementation on four aspects. Firstly, we looked at numerical stability where we found a potential noise feedback loop. However, this noise does not manifest itself when particles are in close enough proximity to eachother. This means that there is enough mass wich miniscule stress force noise cant noticably affect. Secondly, we took a look at the parameters mentioned by the paper as starting values. We found here that most values fall within an expected range of measured data, with the major exception being the critical compression and stretch limits. The maximum size of the stress generated seems meaningless against the mass of the particles. Thirdly, we paid attention to the scaling of the simulation resolution. We observed that the scaling behaves expectedly. Lastly, an implementation of the simulation was used to measure the time taken to simulate a single frame. Using this data we observed that the simulation scales linearly as we expected from the theoretical analysis.

Our analysis a better understanding of the simulation. And an insight to noise being generated through numerical instability. The simulation works within its intended simulation range. Albeit not being able to simulate the snow's resting state, because MPM does not handle incompressible materials well.

5.2 Future Work

While this thesis was being worked on the original authors already have continued their research on this subject. For that we direct the reader to the continued work of Stomakhin et al. (2014) for a method that already adresses some of the weaknesses identified in this thesis.

However, even the expanded method still has room for improvement. Such as, what initially got me invested into analyzing this method, the lack of external forces such as wind. It could be possible to extract volumetric information from the grid and use the distributive property of the grid when the information gets contributed back towards the particle.

Appendix A

Computing the Relation Between Mass and Volume For Stress

Here we present the computations done when focussing solely on the vertical velocity component of a particle. For simplicity in computations the particle is perfectly centered in a grid cell. And the bottom (y-axis) grid cells are colliding with a plane. This computation focusses solely on on the forces and velocities on the y-axis and will simplify some vector math.

Using the maximum strain possible on the y-axis within the limits of the clamping in $\boldsymbol{\Sigma}$ with value $\theta_c = 2.5 \times 10^{-2}$ we arrive at the deformation gradient as

$$\boldsymbol{F}_E = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.975 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.975 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We use the default Young's modulus and Poisson's ratio for the lamé parameters

$$\mu = \frac{E}{2(1+\nu)} = \frac{140000}{2(1+0.2)} = 58333.333$$

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} = \frac{140000}{(1+0.2)(1-2\times 0.2)} = 38888.889$$

Using Equation 3.23 we obtain

$$\begin{aligned} \sigma &= 2\mu(\boldsymbol{F}_E - \boldsymbol{R}_E)\boldsymbol{F}_E^T + \lambda(J_E - 1)J_E\boldsymbol{I} \\ &= \begin{bmatrix} -947.91667 & 0 & 0 \\ 0 & -3791.6667 & 0 \\ 0 & 0 & -947.91667 \end{bmatrix} \end{aligned}$$

We position the particle perfectly centered in a grid cell to obtain the degenerate weight sampling case. This simplifies the computations to 27 grid cells instead of 64. This degenerate case is shown for the weight function in Figure 4.2.

Further more we will have the particle resting slightly above a perfectly horizontal implicit collider plane. The bottom 9 grid cells will be inside the collider and thus, due to collision, those cell's velocity will inevitably be set to zero from the equations in Section 3.2.5.

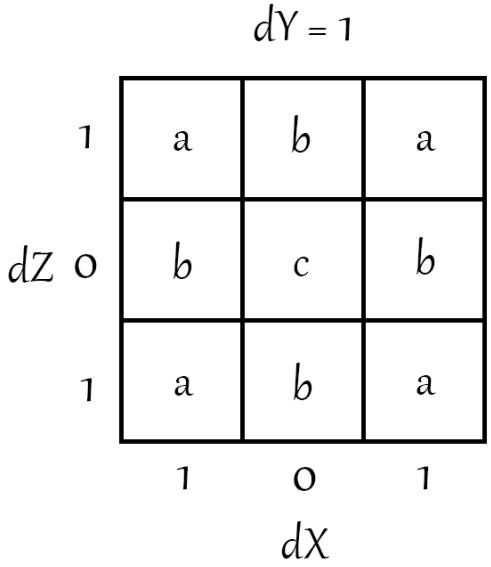

FIGURE A.1: Illustrating how there are only 3 different cases when computing the weight gradient using the absolute values of the distances. For a single cell in the upper layer the computation is $result = N(dX)\nabla N(dY)N(dZ)$. So, $a = N(1)\nabla N(1)N(1)$, $b = N(0)\nabla N(1)N(1)$ and $c = N(0)\nabla N(1)N(0)$

Looking at Equation 3.23 we note that the weight gradient is used. And from our degenerate case illustrated in Figure 4.2 the weight gradient of a perfectly centered particle on an axis will have a slope of 0. The entire middle layer will have a y-axis distance of zero and thus a slope of zero.

Combining the collision and the degenerate case only the upper 9 grid cells contribute to the particle's eventual resulting velocity from the strain. We will now compute the stress forces in those relevant grid cells, and only for the y-component. We will refer to the layer of these relevant cells as the upper layer, as it is the top layer in regards to the y-axis.

From Figure A.1 we obtain the three possible cases for grid cells. Now we compute the ∇w_{ip}^n for each case. Note, these are only the weight gradient values for the y-component.

$$N(1) = \frac{1}{6}$$
$$N(0) = \frac{2}{3}$$
$$\nabla N(1) = -\frac{1}{2}$$

$$a = N(1) \cdot N(1) \cdot \nabla N(1) = \frac{1}{6} \cdot \frac{1}{6} \cdot -\frac{1}{2} = -\frac{1}{72}$$
$$b = N(1) \cdot N(0) \cdot \nabla N(1) = \frac{1}{6} \cdot \frac{2}{3} \cdot -\frac{1}{2} = -\frac{1}{18}$$
$$c = N(0) \cdot N(0) \cdot \nabla N(1) = \frac{2}{3} \cdot \frac{2}{3} \cdot -\frac{1}{2} = -\frac{2}{9}$$

Now we can compute the total upward stress force generated by the strain by summing over the grid cells.

$$
\begin{aligned}
f_a &= -3791.6667 \cdot -\frac{1}{72} = 55.66 \\
f_b &= -3791.6667 \cdot -\frac{1}{18} = 210.65 \\
f_c &= -3791.6667 \cdot -\frac{2}{9} = 842.59 \\
f &= -(4 \cdot f_a + 4 \cdot f_b + f_c) = -1907.83
\end{aligned}
$$

Note we left out the V_p from Equation 3.23. So to put the force in terms of the volume

$$
f = -1907.83 V_p
$$

Now we compare this to gravity force aplied to all the nodes. We adjust for the mass colliding with the plane because its velocity will be set to zero regardless. We do so by subtracting those weight values from the total. Using the weight gradients in a similar fashion as above we can compute the weights of the cells that have to be subtracted and obtain a scalar value between 0 and 1.

$$
\begin{aligned}
N(1) &= \frac{1}{6} \\
N(0) &= \frac{2}{3}
\end{aligned}
$$

$$
\begin{aligned}
a &= N(1) \cdot N(1) \cdot N(1) = \frac{1}{216} \\
b &= N(1) \cdot N(1) \cdot N(0) = \frac{1}{54} \\
c &= N(1) \cdot N(0) \cdot N(0) = \frac{2}{27} \\
w_{sub} &= 4 \cdot a + 4 \cdot b + c = \frac{1}{6}
\end{aligned}
$$

We know the sum of all the weights is 1. So to get the fraction of the mass we do want we subtract it from 1.

$$
\begin{aligned}
\boldsymbol{f}_g &= m_p \cdot (1 - w_{sub}) \cdot -9.81 \\
&= m_p \cdot (1 - \frac{1}{6}) \cdot -9.81 \\
&= m_p \cdot -8.175
\end{aligned}
$$

$$
\begin{aligned}
-8.175 m_p &= -1907.83 \times V_p \\
8.175 m_p &= 1907.83 \times V_p \\
m_p &= 233.37 \times V_p
\end{aligned}
$$

To turn this into a density given $\rho = \frac{m}{V}$ we obtain a density of $0.004285kg/m^3$ for a particle to theoretically support itself under its own weight from stress.

Bibliography

Jang, E. (2014). *GPU-Accelerated Material Point Method Snow Simulation*. Youtube. URL: https://www.youtube.com/watch?v=Mv0t7sKHgpU.

Jiang, C. et al. (2015). "The Affine Particle-In-Cell Method". In: *ACM Transactions on Graphics* 34.4.

Judson, A. and N. Doesken (2000). "Density of Freshly Fallen Snow in the Central Rocky Mountains". In: *Bulletin of the American Meteorological Society* 81.7.

Losasso, F. et al. (2006). "Multiple Interacting Liquids". In: *ACM Transactions on Graphics* 25.3.

Paterson, W.S.B. (1994). *The Physics of Glaciers*. Elsevier.

Petrovic, J. J. (2003). "Review Mechanical properties of ice and snow". In: *Journal of Materials Science* 38.

Shapiro, L. H. et al. (1997). "Snow mechanics : review of the state of knowledge and applications". In: *CRREL Report* 97.3.

Stomakhin, A. et al. (2013a). "A Material Point Method for Snow Simulation". In: *ACM Transactions on Graphics* 32.4.

— (2013b). *Material Point Method for Snow Simulation*. Tech. rep.

Stomakhin, A. et al. (2014). "Augmented MPM for Phase-Change and Varied Materials". In: *ACM Transactions on Graphics* 33.4.

Yue, Y. et al. (2015). "Continuum Foam: A Material Point Method for Shear-Dependent Flows". In: *ACM Transactions on Graphics* 34.5.

Zhu, Y. and R. Bridson (2005). "Animating Sand as a Fluid". In: *ACM Transactions on Graphics* 24.3.