Master's Thesis in Graduate School of Library, Information and Media Studies

Position Based Snow Simulation with Phase Change

March 2021

201921662

CHEN YI

Position Based Snow Simulation with Phase Change
位置ベース法に基づく相転移を伴う雪のシミュレーション

Student No.: 201921662
氏名：CHEN YI
Name: CHEN YI

Snow is the most common winter scene in animation. The simulation of snow is widely used in various places. Most snow are naturally accompanying with abundant water resources. When the environmental temperature is near to the melting point, the existence of water will make the snow have drastic phase changes. This leads to different elastoplastic characteristics of snow and will further affect the formulation of snow shape. Therefore, it is necessary to simulate both solid and fluid water in order to increase the realism of snow simulation.

In our research, we introduce a method based on Position Based Dynamics by using two different discretization methods, Discrete Element Method (DEM) and Smoothed Particle Hydrodynamics (SPH) to simulate the interaction of surrounding water and ice crystals in snow. By introducing stretch constraints to DEM particles which performs as the interlinks of snow particles, we successfully simulate the deformation effect. In addition, we address the problem of over-connected interlinks by controlling the number of connections with considering the homogeneous freezing effects of ice crystals in snow. Also, to simulate the interaction of snow and its melts, our methods transfers the heat by using a conduction model for heterogeneous materials and the phase of a particle will be determined based on its current temperature. We blends the contribution of solid and fluid solver to imitate the latent heat effect and successfully stabilize the simulation.

In addition, we develop a fully GPU-based algorithm which addresses the complex implementation of phase-change problems. Since the transition of a particle from one phase to another creates empty spaces in memory which yields low performance on GPU computation, a parallelized exclusive scan is performed so that a compacted array can be obtained after the phase change. Furthermore, to prevent the read/write contradiction caused by the out-of-order execution on GPU, we utilize the atomic operations to comply the lock-free implementation of recording the information of interlinks.

As a result, our method is able to perform various characteristics of snow including deformation, phase change, and the rigid-fluid interactions. Despite of the limitations caused by our choice of constraint and the uniform size of particles, our position-based solver is a stable and controllable solution for simulating the complex behaviors of snow-water interactions.

In the future, we plan to simplify the need of parameter tuning by using XPBD to allow user to reference to real-world measurements. Also, we expect to extend our simulator to make it possible to interact with other objects so that it can be used to simulate various scenarios.

Principal Academic Advisor: Masahiko MIKAWA
Secondary Academic Advisor: Makoto FUJISAWA

Position Based Snow Simulation with Phase Change

CHEN YI

Graduate School of Library, Information and Media Studies

University of Tsukuba

March 2021

Contents

List of Figures

Chapter 1

Introduction

Snow scene is one of the most general public's imagination for the frosty winter landscapes. In high latitude sites, snow covers can overwhelm roads, vehicles and even trees when the mighty blizzard arrives and leave tons of accumulated snow for weeks until the temperature bouncing back to the melting point. However, the temperature difference between day and night causes the snow constantly hovering in between solid and fluid phases. This intense bilateral phase changing process gradually solidifies the originally loose snowflakes and changing them into stiff ice fragments, making the behavior of snow closer to the muddy ground instead of sandy desert. In this paper, we introduce an algorithm to simulate the interaction of different phases of snow including ice, water, and snow by considering the internal phase changes.

In recent years, many snow simulation methods have been proven to be useful to reduce complex modeling tasks in Computer Graphics (CG) production. Generally, snow in most of researches, is approximated with large amount of particles where each particle carries various physics quantities such as mass or density. These properties will then be used to simulate the behavior of snow with a computational method specialized to solve specific effects in limited conditions which constantly neglects the influence of internal water. Therefore, these methods typically results in unrealistic artifacts such as sand-like snow. In recent researches, Material Point Method (MPM) has been widely used to resolve these problems via considering the constitutive material property of snow by treating it as a continuum elastic-plastic body [2]. This approach, unfortunately, cannot demonstrate effects such as reflections and refraction precisely due the the ignorance of the internal water. This causes the artists still have to make further adjustments in order to achieve their expectations. Moreover, these force-based methods are notorious for the dependence on small time steps and the necessity of solving non-linear problems. This limits most of the applications unable to interact with user in the manner of real-time or just-in-time during the designing process.

In our research, we use Position Based Dynamics (PBD) to acquire higher stability and controllability via directly manipulating the position of objects [3]. This allows us to handle the behavior of snow by solving multiple constraints applied on simulated particles. In addition, referencing to Macklin's work [4], our method divided snow material into two different particle sets. While the fluid water is discretized by Smooth Particle Hydrodynamics (SPH) (Chapter 4.2), we use Discrete Element Method (DEM) to perform rigid body physics on the collisions of ice fragments. To simulate phase change effects, we developed an algorithm to compute snow-to-ice, ice-to-water, and water-to-ice transitions (Chapter 6).

Furthermore, we implemented our algorithm on CUDA in a lock-free manner so that the computation can grant more advantages of concurrent execution on modern GPUs, which largely improves the performance of our simulator.

In summary, our approach starts from the micro aspect of snow simulation, which focuses on the interaction of different phases of water. With our algorithm, we also give out an alternative to the long-standing problem of multi-phase particle simulation on GPGPU. More specifically, our main contributions are as follows:

- A stable and controllable position-based solver for snow simulation
- An innovative way to simulate snow by taking the interactions of snow-ice, ice-water and water-ice into consideration
- A lock-free algorithm for bilateral particle transition on GPU

Chapter 2

Related Work

2.1 Snow Simulation

In real-time applications, the snow accumulation effects have caught the eye of researchers for a long time. A very first research proposed by Nishita et al. [5] simulates the shape of fallen snow by using the density distribution modeled from metaballs. Fearing's method [6] accumulates snow in an upward manner, shooting snowflakes upward into the air and redistributes the configurations until the snowfall is stable. Both methods are capable of reconstructing realistic snow accumulation effects, yet their simulation cannot capture the transforming feature caused by phase changes. In Maréchal's research [7], they presented an environmental heat transfer model for simulating snow melting transitions at the terrain in the winter scenes. While these researches are mostly used in static snow modeling, Takahashi et al. simulated the sintering effect by considering snow as non-Newtonian fluid [8]. Their method extended the original Navier-Stoke equation with additional sticky force caused by sintering effect, and successfully made snow particles be able to attach with the environmental objects. Until recently, snow simulation in computer graphics often assumed that the phase of the snow did not change because of its complexity. Therefore, most of the methods were unable to accurately simulate the shape and behaviors of high-temperature moist snow (close to melting point). Stomackin et al.[2] overcame this bottleneck with their MPM solver, which combines both Eulerian and Lagrangian methods . Similar to PIC/FLIP solver used in computational solid dynamics [9], they provided a constitutive model for simulating the elastoplastic properties generated by varying phase change inside snow material. Following their footage, Gissler et al.[10] demonstrated an implicit compressible SPH solver for snow simulation. Their method is capable of simulating various effects of snow including snow fall accumulations, phase change, and the interaction with environmental rigid bodies. In contrast to Stomackin and Gissler's method, we did not only use one way to discretize snow particles. We approximate snow material by two different particle sets so that our method can perform the moist effects from fluid and rigid body collision from solid respectively.

2.2 Rigid Fluid Coupling

The technique of rigid-fluid coupling is very important when simulating multi-phases fluid. The interaction of rigid and fluid in early research are typically treated as one-way cou-

pling effects [11][12]. Movement of rigid bodies are seldom be affected by fluid because of the difficulty and computation cost of Eulerian-based methods. Instead, Lagrangian-based methods provide better alternatives on two-way coupling since the advection of fluid can be derived from the movement of particle. In Carlson's research [13], they introduced a semi-Lagrangian solver with Lagrange multipliers to reconstruct realistic interaction for most rigid fluid sceneries. Other Lagrangian-based method, for example, Rungjiratananon et al. [14] takes wetness into consideration simulating the interaction of sand and water by combining DEM and SPH. In our research, we referenced to their concept of wetness. We assume that all snow particles have fixed water around them. This allows us to dynamically connect solid particles in order to perform elastoplastic characteristics of snow.

While some focus on handling two-way interactions, other researches put their efforts on the stability at the interfaces. In heterogeneous particle simulations, the density of a particle can largely affected by the density of its neighbors. The sudden density changes at the interface leads to overestimated pressure force and pushes fluid away from rigid. Solenthaler et al. [15] resolved this problem by modifying the density computation of SPH with adapted density for each particle. Similar to their result, Akinci et al. [16] extended their method to rigid-fluid coupling by changing the density contribution with considering the volume of boundary particles. In this paper, we use the same as Akinci's method to handle the instability caused by static boundary particles. In addition, since snow and water is close in density, we simply adopt the original SPH density computation and directly correct the positions with non-penetration constraint to ensure this two types of particles not intersecting with each other.

2.3 Phase Change

The intense phase change of high temperature snow is crucial to our research. Most of previous researchers only handled unilateral phase change such as melting or freezing. In Fujisawa's research [17], they used CIP method to advect the volume fraction at each grid cell of the fluid to accomplish the ice melting animation. Nakasone [18] proposed an approach for high performance melting process based on SPH and Shape Matching method. However, in real world, the phenomena of bilateral phase change is intense when the temperature is close to melting/freezing point. In Glisser's research [10], their implicit SPH solver is capable of dealing with the melting/freezing process of snow simultaneously. However, since their solver was designed for "snow", the behavior of melted snow water remains grainy like sand instead of real water. In our research, we resolve this problem by using two different solvers for the corresponding phases of particles. By doing this, the melted water in our simulation can be simulated as fluid and interact with surrounding objects, and vice versa, new frozen fluid particles is also possible to interact with others as rigid body as well.

Chapter 3

Overview of Proposed Method

3.1 Method Overview

As we mentioned in the introduction, the nature of snow is very complicated because it contains water that cannot be ignored. In our research, we treat water and snow separately and take the phase change into consideration aiming to simulate more realistic snow scenes in natural environment. In the simulation part, we reference to Macklin's research [4] building the system totally based on Position Based Dynamics. With direct operations on the position information, our approach is able to perform more stable multi-phase material simulation that has not been achieve in the past. For the computation, we exploit the power of GPGPU to speed up the process of constraint solving for PBD. And because the phase change of multi-phase material requires transferring particles to the corresponding array, we propose a lock-free solution to reduce the negative effect on performance. In Chapter 4, we will firstly introduce our assumption on snow material and the modeling technique. Then in Chapter 4.2, we will explain discretization methods for fluids and solids respectively in our simulation. In Chapter 5, Position Based Dynamics will be formally introduced with the explanation of how to solve constraints. And in Chapter 6, we will discuss the detailed process related to phase change and how we handle the unstable results caused by sudden state changing. Finally, in Chapter 7, we will describe the GPU implementation of our algorithm.

3.2 Flow of Computation

As shown in Figure 3.1, the solver will perform an integration to predict the position of each particle based on external forces such as gravity at the start of current time step. Then it will use this predicted position to calculate the position correction on Δx in order to satisfy the constraints. Before the beginning of the iteration, a search will be conducted to acquire the neighbor information of each particle. Based on this information, heat will be transferred in between the nearby particles and determine the phase of a particle and the inter-connection of DEM particles. For each iteration, the fluid solver will try to satisfy the density constraint for fluid and the solid solver will solve the non-penetration constraint and stretch constraint for solid. After complete the computation of solid-solid and fluid-fluid corrections, the solver will process the rigid-fluid coupling calculation to get solid-fluid correction $\Delta \boldsymbol{x}^i_{solid-fluid}$ and fluid-solid corrections $\Delta \boldsymbol{x}^i_{fluid-solid}$. We add up the corrections

to get the final correction at ith iteration that

$$\Delta \boldsymbol{x}^i_{solid} = \Delta \boldsymbol{x}^i_{solid-solid} + \Delta \boldsymbol{x}^i_{solid-fluid} \tag{3.1}$$

and

$$\Delta \boldsymbol{x}^i_{fluid} = \Delta \boldsymbol{x}^i_{fluid-fluid} + \Delta \boldsymbol{x}^i_{fluid-fluid} \tag{3.2}$$

where $\Delta \boldsymbol{x}_{fluid-fluid}$ is the correction of a fluid particle affected by other fluid particles, and $\Delta \boldsymbol{x}_{solid-solid}$ is the correction of a solid particle affected by other solid particles. After the iteration is completed, we can obtain the correction in normal direction. And then our solver will perform the computation of tangential correction in order to simulate the friction. Finally, the system will finalize the correction and additionally modify the velocity to perform the viscosity of fluid.

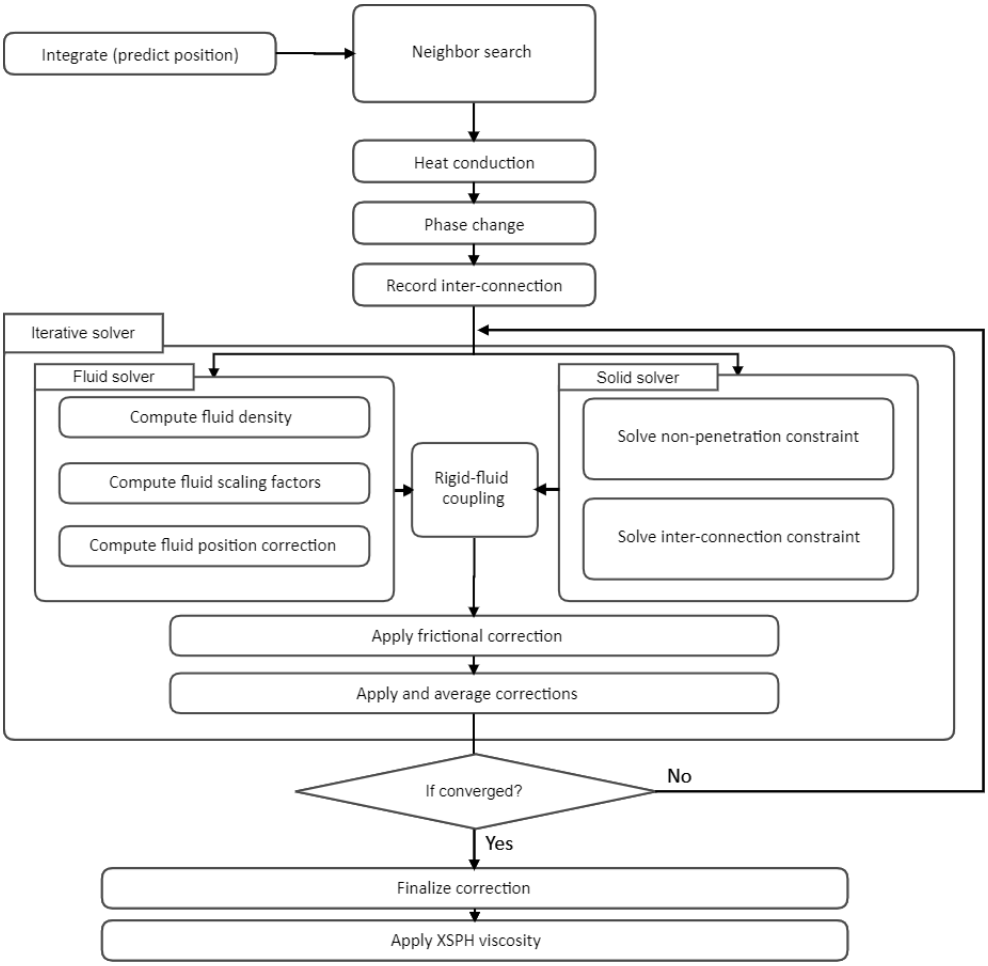


Figure 3.1: Flow of computation

Chapter 4

Modeling and Discretization Methods

4.1 Modeling

4.1.1 Water-Snow Modeling

Snow in natural environment are constantly surrounded with water. As the temperature is near the melting point, the mixture of snow and water behaves totally different with typical snow. This type of snow is typically called watery snow. The movement of the solid part of watery snow can be easily affected by the water flow like the floating ices. In our research, we modeled snow with considering this two different states. As shown in Figure 4.1, we assume snow particles (gray) are always accompanied with pre-melted water (light blue) and have environmental water (blue) nearby. The environmental water in our system is approximated by the fluid particles. In the other side, the pre-melted water does not exist in the simulation system. Instead, it is an abstract concept that allows us to connect snow particles when the temperature is low enough to form ice blocks.

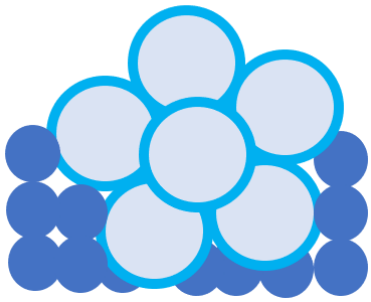


Figure 4.1: Watery snow modeling

When simulating phase changes, these particles are transported between two arrays, solid and fluid, which allows the solver to use different constraints to handle the movement of these particles. Unlike the environmental water, the pre-melted water in our simulation is not treated as particles. Therefore, the pre-melted water will disappear when the solid particle melted.

4.1.2 Snow-to-Ice Formulation

In reality, snow becomes ice blocks when it melts and is re-frozen by the low temperature outside. Following this concept, we connect snow particles together in order to perform this effect. As shown in Figure 4.2, our solver traverses the neighbors of each snow particle. If the distance of the adjacent particles is within the efficient radius h and both of their temperature is under $T_{linking}$, these two particles will be marked as connected and assigned with a stretch constraint. However, naively connecting all candidates makes the ice blocks too hard sometimes. Alternatively, we propose a parameter n_{max} to control the maximum number of connections for each particle. (Note that n_{max} in our simulation is not always a constant, it varies as the temperature of particle changes.) Since the correction of stretch constraint depends on the distance difference and is elastic, we break the connection when the distance of two particle exceeds the initially recorded distance x_{rest} to perform the effect of breakable snow ice blocks. In Chapter 5 and Chapter 6, we will discuss more details about how we compute the correction of the stretch constraint and the transferring algorithm while the phase of a particle changes.

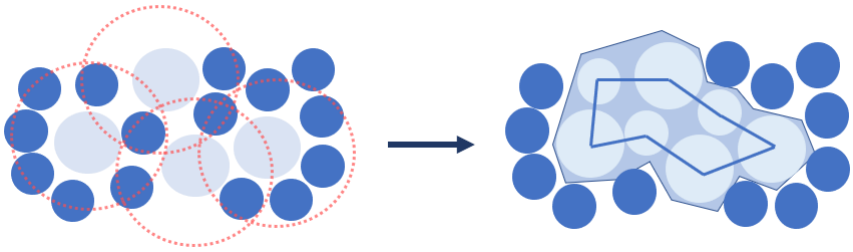


Figure 4.2: Snow-to-Ice Formulation (Connected with stretch constraint)

4.2 Discretization Methods

Since our method incorporates different phases of water, it is necessary to use different discretization methods in order to distinguish the solid water and fluid water. In the following sections, we will explain the fundamentals of Discrete Element Method (DEM) and Smoothed Particle Hydrodynamics (SPH).

4.2.1 Discrete Element Method

DEM is a simple but useful technique widely used to simulate granular material, that can be assumed as a collection of solid particles. It simulates discrete particles which represent standalone objects such as sand or snow in our case. Generally, DEM mainly computes the normal and tangential forces on the collision of two particles as shown in Figure 4.3. Where the normal forces mainly come from the rigid body collision, the tangential forces are mostly caused by the frictional force at the surface contact of two objects as shown in the below figure.

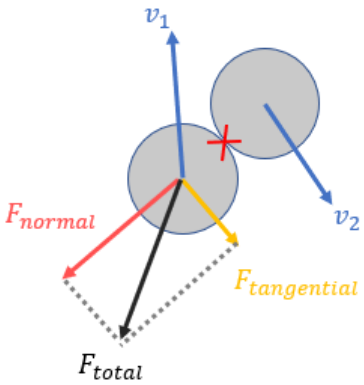


Figure 4.3: Discrete Element Method

The total forces works on a single particle can be calculated as:

$$\boldsymbol{F}_{total} = \boldsymbol{F}_{normal} + \boldsymbol{F}_{tangential} \tag{4.1}$$

where $\boldsymbol{F}_{tangential}$ is computed based on Coulomb's law of friction that

$$\boldsymbol{F}_{tangential} = \mu \boldsymbol{F}_{normal} \tag{4.2}$$

and

$$\mu = \begin{cases} \mu_s, & |\boldsymbol{v}_{relative}| = 0 \\ \mu_d, & \text{else} \end{cases} \tag{4.3}$$

where μ_s is the coefficient of static friction, μ_d is the coefficient of dynamic friction, and $\boldsymbol{v}_{relative}$ is the relative velocity between two collision particles.

Since we use a position based solver in our simulation, we do not use the normal force at the collision. Instead, we use normal position correction to determine the tangential position correction. We will discuss this later in chapter 5.

4.2.2 Smoothed Particle Hydrodynamics

SPH is a well-known computational method for Lagrangian based fluid simulation. Although SPH was firstly developed to simulate phenomenons in astrophysics, it was found to be useful in simulating a wide range of fluid soon after its publication.

The core concept of SPH is to partition fluid into many particles where each particle stores physical quantities such as velocity or pressure. Since values evaluated from particles are typically not continuous, SPH makes the boundaries smoothed by symmetrical kernel functions W in the range of a given effective radius h as shown in Figure 4.4. At an arbitrary location $\boldsymbol{r}$, the quantity A is interpolated by summing up contributions from nearby particles with the following equation:

$$A(\boldsymbol{r}) = \sum_j m_j \frac{A_j}{\rho_j} W(\boldsymbol{r} - \boldsymbol{r}_j, h) \tag{4.4}$$

where j is the index of nearby particles, m is the mass, and ρ is the density. One of the advantage of using smooth kernel is its simplicity to calculate the derivatives of physical

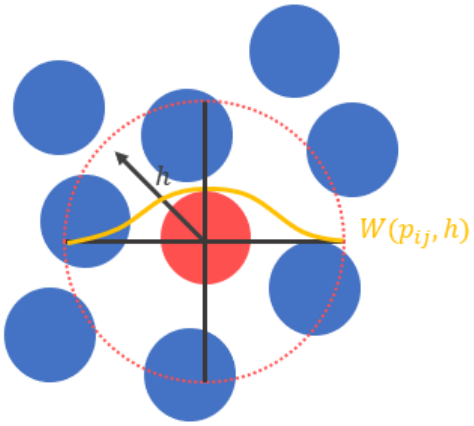


Figure 4.4: Smoothed Particle Hydrodynamics

quantities. For example, the gradient ∇A of a quantity can be directly computed with the gradient of kernel function ∇W as below:

$$\nabla A(\boldsymbol{r}) = \sum_j m_j \frac{A_j}{\rho_j} \nabla W(\boldsymbol{r} - \boldsymbol{r}_j, h) \tag{4.5}$$

and for the laplacian $\nabla^2 A$ is

$$\nabla^2 A(\boldsymbol{r}) = \sum_j m_j \frac{A_j}{\rho_j} \nabla^2 W(\boldsymbol{r} - \boldsymbol{r}_j, h) \tag{4.6}$$

In our research, we use the same kernel functions proposed by Müller et at. [19]. We use the poly6 kernel function to calculate the density, and the spiky kernel for gradient calculation. The equation of both kernel functions are listed below:

$$W_{poly6}(\boldsymbol{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3, & 0 \le r \le h \\ 0, & \text{otherwise} \end{cases} \tag{4.7}$$

$$\nabla W_{spiky}(\boldsymbol{r}, h) = -\frac{45}{\pi h^6} \begin{cases} \frac{1}{r}(h - r)^2 - (h - r), & 0 \le r \le h \\ 0, & \text{otherwise} \end{cases} \tag{4.8}$$

The reason of separating into different kernels is because that the gradient of poly6 becomes zero when r is close to zero. The lack of repulsion force makes the pressure computation tends to generate particle clusters as reported in Müller's early research [19]. Instead, the spiky kernel gives positive numbers which consequently causes the repulsion force increased when the distance r decreases.

Similar to the above problem, the laplacian of spiky and poly6 is negative numbers in range $-h \le r \le h$ which doesn't fit our expectation in calculating the temperature of a particle. we address this problem by using the viscosity kernel instead:

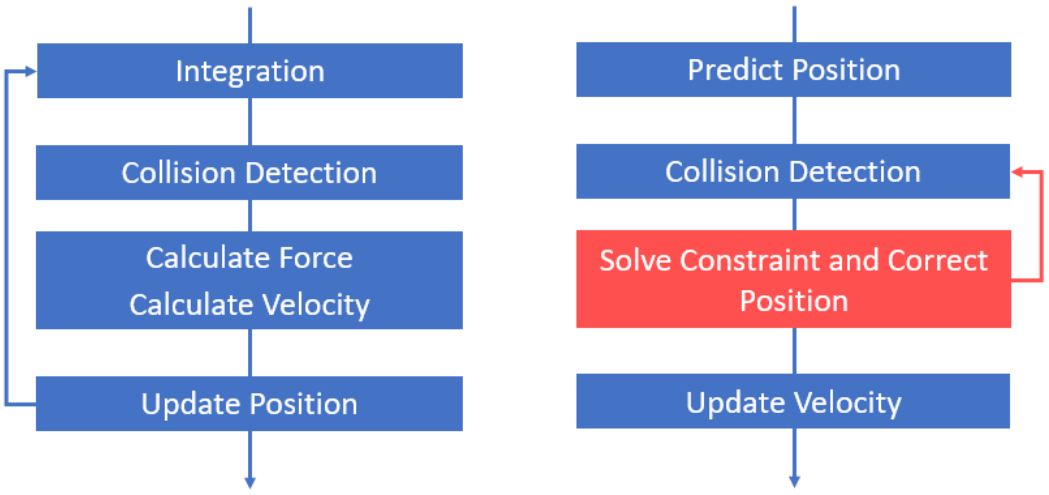
$$\nabla^2 W_{viscosity}(\boldsymbol{r}, h) = \frac{45}{\pi h^6} \begin{cases} (h - r), & 0 \le r \le h \\ 0, & \text{otherwise} \end{cases} \tag{4.9}$$

Chapter 5

Position Based Dynamics

Position based dynamics (PBD) is a numerical approach widely used in modern real-time computer graphics simulation. Traditionally, the motion of dynamics objects is determined by the accumulated forces and the equations derived from Newton's law of motion. These methods are typically called force based methods, which are notorious for their expensive computational cost and the necessity in small time steps. In contrast, position based dynamics reduces the cost by directly manipulating the position of objects to satisfied given constraints. In the following sections, we will introduce the basic of PBD and the constraints. Then we will explain how to use PBD to simulate fluid and do the rigid-fluid coupling effects. Lastly, we will discuss different types of solvers with mentioning their advantages and disadvantages.

5.1 Simulation Loop



(a) Force Based Simulation (b) Position Based Simulation

Figure 5.1: Simulation loops

In force-based simulation, solvers must firstly compute the acceleration of objects and then integrates with the time step Δt in order to have the velocities to update the positions. A traditional force-based method has a simulation loop like Figure 5.1a. When Δt is large,

the errors in explicit methods will be accumulated. A solution to address this problem is to use implicit methods. However, some implicit methods are too complicated to implement and require massive computational resources to increase the accuracy of simulation. Plus, in computer graphics research, accuracy is not as important as the appearance problems in most of the cases. For example, users will put their attention on the existence of the deformation of a elastic ball when it collides with other balls instead of how accurate the bouncing speed is. Therefore, PBD offers a solution based on constraint dynamics as illustrated in Figure 5.1b. The biggest difference between these two methods is that PBD initially predicts the positions of objects and then projects the positions until they match the given constraints. By doing this, PBD has higher stability even when the penetration depth is deep. In contrast, traditional force-based methods need the information of internal structure of the objects which means it is necessary to solve the complex mass-spring system with continuously changing the times steps until the system is stable. Consequently, force-based simulation usually spends more time than position-based simulation because of the non-negligible sub-step integration. In the next section, we will explain how PBD resolves this problem by defining constraints to simulate different behaviors.

5.2 Constraints

The interaction of the objects in a system is the key factor to determine whether a simulation is realistic or not. One classic example is the collision response when objects contact with each other. In order to match user's expectation such as an object sliding over the ground keeps the same distance to the ground, constraints are introduced to restrict the motion of objects in a simulation system. In PBD, the constraint is represented as a function $C(\boldsymbol{x})$ where $\boldsymbol{x}$ is the positions of target objects affected by this constraint. There are two types of constraint function: equality and inequality constraint where the equality constraint is represented as:

$$C(\boldsymbol{x}) = 0 \tag{5.1}$$

and inequality constraint is:

$$C(\boldsymbol{x}) \geq 0 \; or \; C(\boldsymbol{x}) \leq 0 \tag{5.2}$$

Generally, positions of objects in equality constraints will always be corrected until the constraint function is satisfied during the simulation. In the other hand, inequality constraint $C(\boldsymbol{x}) \geq 0$ will only be corrected when the constraint value violates given conditions. In the following contents, we will introduce the basic concept of constraints and how PBD solves them.

Assume a new position $\boldsymbol{x}'$ is computed after the integration that $\boldsymbol{x}' = \boldsymbol{x} + \Delta\boldsymbol{x}$. In order to behave correctly, this new position $\boldsymbol{x}'$ must also satisfy the constraint function which yields:

$$C(\boldsymbol{x}') = 0 \tag{5.3}$$

In PBD, the position is projected in the gradient direction of the constraint function ∇C. $C(\boldsymbol{x}')$ can be approximated by Taylor expansion which yields

$$C(\boldsymbol{x}') = C(\boldsymbol{x} + \Delta\boldsymbol{x}) \approx C(\boldsymbol{x}) + \nabla_{\boldsymbol{x}} C(\boldsymbol{x}) \cdot \Delta\boldsymbol{x} = 0 \tag{5.4}$$

And let the correction be

$$\Delta \boldsymbol{x} = \lambda \nabla C(\boldsymbol{x}) \tag{5.5}$$

Substitute Equation 5.5 to Equation 5.4, we get

$$\lambda = -\frac{C(\boldsymbol{x})}{|\nabla_x C(\boldsymbol{x})|^2} \tag{5.6}$$

yielding the scaling factor s in case of a constraint composed of a number of positions to be

$$s = \frac{C(\boldsymbol{x}_1, ..., \boldsymbol{x}_n)}{\sum_j |\nabla_x C(\boldsymbol{x}_1, ..., \boldsymbol{x}_n)|^2} \tag{5.7}$$

where the position correction is

$$\Delta \boldsymbol{x}_i = -s \nabla_{\boldsymbol{x}_i} C(\boldsymbol{x}_1, ..., \boldsymbol{x}_n) \tag{5.8}$$

In PBD, the position correction is weighted with $w_i = 1/m_i$ which makes Equation 5.7 be

$$s = \frac{C(\boldsymbol{x}_1, ..., \boldsymbol{x}_n)}{\sum_j w_j |\nabla_{\boldsymbol{x}_i} C(\boldsymbol{x}_1, ..., \boldsymbol{x}_n)|^2} \tag{5.9}$$

Consequently, we can derive the final term of $\Delta \boldsymbol{x}$ as the following

$$\Delta \boldsymbol{x}_i = -s w_i \nabla_{\boldsymbol{x}_i} C(\boldsymbol{x}_1, ..., \boldsymbol{x}_n) \tag{5.10}$$

In our research, we use the stretch constraint to simulate spring-like effects when connecting discrete snow particles into stretchable icy blocks. The function of stretch constraint is defined as:

$$C(\boldsymbol{x}) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\| - d = 0 \tag{5.11}$$

Following the inductions above, the correction of particle i is

$$\Delta \boldsymbol{x}_i = -k_{stretch} \frac{w_i}{w_i + w_j} (\|\boldsymbol{x}_i - \boldsymbol{x}_j\| - d) \frac{\boldsymbol{x}_i - \boldsymbol{x}_j}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|} \tag{5.12}$$

where $k_{stretch}$ is added here to control the stiffness of the connection.

The non-penetration constraint we use to handle the collision response of particles has similar constraint function as Equation 5.11. There is a slight difference that non-penetration constraint is inequality constraint which may only be applied when particles collided with each other. It is defined as the following

$$C(\boldsymbol{x}) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\| - (r_i + r_j) \geq 0 \tag{5.13}$$

that r_i and r_j are the radius of particle i and j. Consequently, the correction of non-penetration constraint is given by

$$\Delta \boldsymbol{x}_i = -k_{bounce} \frac{w_i}{w_i + w_j} (\|\boldsymbol{x}_i - \boldsymbol{x}_j\| - (r_i + r_j)) \frac{\boldsymbol{x}_i - \boldsymbol{x}_j}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|} \tag{5.14}$$

where k_{bounce} is the bouncing stiffness. In our research, we typically set $0.5 \leq k_{bounce} \leq 1$ to perform snow-like particle collision which can also increase the stability when the phase change is very intensive.

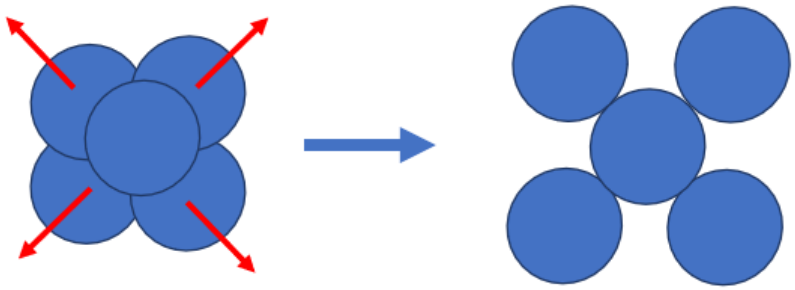


Figure 5.2: Density constraint

5.3 Position Based Fluid

Although SPH has various advantages, the nature of using particles makes it unstable when clustering occurs. To address this problem, many previous researches use stiffness equations to handle the incompressibility of fluid. However, such methods requires small time steps to prevent overshooting problems. Position based fluid overcame this issue by applying density constraints on fluid particles. In PBF, the density ρ_i is computed as using Equation 4.4

$$\rho_i = \sum_j m_j W(x_{ij}, h) \tag{5.15}$$

where j is the neighbor of particle i, and x_{ij} is the distance between particle i and j. Considering that incompressible fluid tends to relax its density when it is too dense (as Figure 5.2), the density constraint of each particle is defined with the rest density ρ_0 which yields inequality function such that

$$C_i(\boldsymbol{x}_1, ..., \boldsymbol{x}_n) = \frac{\rho_i}{\rho_0} - 1 \leq 0 \tag{5.16}$$

To obtain the correction of position, the gradient of $C_i(x)$ with respect to particle k is given by

$$\nabla_{\boldsymbol{x}_k} C_i = \frac{1}{\rho_0} \begin{cases} \sum_j \nabla_{\boldsymbol{x}_k} W(x_{ij}, h), & k = i \\ -\nabla_{\boldsymbol{x}_k} W(x_{ij}, h), & k = j \end{cases} \tag{5.17}$$

Substituting 5.17 into Equation 5.7, we get the final term of position correction that

$$\Delta \boldsymbol{x}_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j) \nabla W(x_{ij}, h) \tag{5.18}$$

Since scaling factors λ_i, λ_j may be divided by zero, a relaxation parameter ϵ is added to the denominator which makes it becomes

$$\lambda_i = -\frac{C_i(\boldsymbol{x}_1, ..., \boldsymbol{x}_n)}{\sum_k \|\nabla_{\boldsymbol{x}_k} C_i\|^2 + \epsilon} \tag{5.19}$$

One advantage of using inequality density constraint is the stability. It omits the negative pressure when the constraint value is large (meaning particles are loose within the effective radius). In addition, since our research requires particles being switched between different

phases, it can also ease the intense changes at the number of neighbors of SPH particles when they are newly added in.

Finally, we apply the XSPH viscosity as the following to get coherent motion of SPH particles [4].

$$\boldsymbol{v}_i^* = \boldsymbol{v}_i - \epsilon^{visc} \sum_{j=1}^{k} [\frac{m_i}{\rho_i} (\boldsymbol{v}_i - \boldsymbol{v}_j) W(x_{ij}, h)] \tag{5.20}$$

where the computation traverses all nearby k particles with ϵ^{visc} being the viscosity parameter, and v_i, v_j being the velocities after the correction of position.

5.4 Position Based Friction

In order to simulate the friction in Position Based Dynamics, we incorporate the model proposed by Macklin [20] which solves the interpenetration of DEM particles with Equation 5.13. After the non-penetration correction is completed, the relative correction of two particles i and j in the perpendicular direction can be computed as the following that

$$\Delta \boldsymbol{x}_\perp = [(\boldsymbol{x}_i^* - \boldsymbol{x}_i) - (\boldsymbol{x}_j^* - \boldsymbol{x}_j)] \perp \boldsymbol{n} \tag{5.21}$$

where $\boldsymbol{x}_i^*$, $\boldsymbol{x}_j^*$ is the corrected position after all constraints being resolved, $\boldsymbol{x}_i, \boldsymbol{x}_j$ is the initial position in current time step, and $\boldsymbol{n} = \boldsymbol{x}_{ij}^* / |\boldsymbol{x}_{ij}^*|$. As the definition in Coulomb's law of friction in Equation 4.2, the frictional correction can be divided into static and kinetic friction which yields

$$\Delta \boldsymbol{x}_\perp = \frac{-w_i}{w_i + w_j} \begin{cases} \Delta \boldsymbol{x}_\perp, & |\Delta \boldsymbol{x}_\perp| < \mu_s d \\ \Delta \boldsymbol{x}_\perp \cdot min(\frac{\mu_k d}{\Delta \boldsymbol{x}_\perp}, 1), & \text{otherwise} \end{cases} \tag{5.22}$$

where w_i, w_j is the inverse of mass, μ_k, μ_s is the coefficient of kinetic and static friction respectively, and d is the penetration depth of particle i, j. Notice that we make a slight change in the direction of friction with a negative sign at the Equation 5.22 which represents the restriction on the tangential correction. With a couple of experiments, we believe the friction here should be the correct one instead of the one used in Macklin's paper [20].

5.5 Rigid-fluid Coupling

Coupling is the technique to handle the interaction between different objects. Since the discretization methods varies by the material, the solver must take additional computations in order to perform the influence from another material. In our research, we handle the rigid-fluid coupling effect in a complete position-based manner. In the following subsections we are going to discuss our method of two-way rigid-fluid coupling.

5.5.1 Rigid to Fluid

An intuitive way to handle the rigid to fluid correction is to take the density contribution of rigid particles into consideration. Similar to Equation 5.15, the corrected fluid density $\hat{\rho}_i$ is computed as the following that

$$\hat{\rho}_i = \sum_j m_j W(x_{ij}, h) + \sum_k m_k W(x_{ik}, h) \tag{5.23}$$

where j is the index of neighbor SPH particles, and k is the index of neighbor DEM particles. As the density changes with surrounding materials, the density constraint will automatically move the fluid particles inward or outward until it satisfies the given condition.

However, such correction sometimes results in clustering or interpenetration because the density constraint (Equation 5.16) does not guarantee to keep particles to be separated. Therefore, we address this problem with using non-penetration constraint as described in Equation 5.13 which yields the position correction be

$$\Delta \boldsymbol{x}_{sph} = \frac{1}{\rho_0}(\sum_j (\lambda_i + \lambda_j)\nabla W(\boldsymbol{x}_{ij}, h) + \sum_k (\lambda_i + \lambda_k)\nabla W(\boldsymbol{x}_{ik}, h)) + \Delta \boldsymbol{x}_{non\_penetration} \quad (5.24)$$

where λ_j and λ_k is the scaling factor of nearby fluid particle j and solid particle j respectively derived from Equation 5.19, and $\Delta \boldsymbol{x}_{non\_penetration}$ is the non-penetration correction computed by Equation 5.14.

5.5.2 Fluid to Rigid

Similar to rigid-to-fluid correction, our approach uses non-penetration constraint to handle the interaction from fluid particles to rigid particles. A slight difference is that we do not take the fluid density into consideration since it will make the solid particles behave like fluid. Therefore, the correction of a solid particle i becomes

$$\Delta \boldsymbol{x}_{dem} = \Delta \boldsymbol{x}_{sph \to dem} + \Delta \boldsymbol{x}_{dem \to dem} \quad (5.25)$$

where $\Delta \boldsymbol{x}_{sph \to dem}$ is the correction SPH particles contribute and $\Delta \boldsymbol{x}_{dem \to dem}$ is the contribution of correction from nearby DEM particles.

5.6 Solver

Generally, there are two different solvers can be used to solve the constraint of PBD. One iterates all positions sequentially and uses the updated estimates to converge the linear system, is known as the Gauss-Seidel iteration. Another one is the Jacobian method which simultaneously uses old estimates in every iteration. The advantage of Gauss-Seidel iteration is that it can converge faster than Jacobian method. However, for every candidate x_i at nth iteration, it must wait until the computation of x_{i-1} is finished, which causes read-write problem if it is implemented in parallel. In contrast, Jacobian method suits better to parallel programming because there is no need to access the updated value.

In our research, we use Jacobian solver to compute the position correction for each particles. Since a particle might be influenced by multiple constraints, a typical solution to prevent over aggressive convergence is to average the sum of correction $\Delta \boldsymbol{x}_i = \sum_{j=0}^{n_i} \Delta \boldsymbol{x}_j$ with the number of constraints n_i applied on particle i that:

$$\Delta \tilde{\boldsymbol{x}}_i = \frac{1}{n_i} \Delta \boldsymbol{x}_i \quad (5.26)$$

Furthermore, in order to fasten the convergence of Jacobian solver, a global successive over-relaxation parameter (SOR) ω_{SOR} is multiplied to the correction such that:

$$\Delta \tilde{\boldsymbol{x}}_i = \frac{\omega}{n_i} \Delta \boldsymbol{x}_i \quad (5.27)$$

where the value of ω is typically set to $1 \leq \omega \leq 2$. In some cases, if the simulation becomes too unstable, sacrificing the convergence speed by setting ω smaller than 1 can increase the stability in some degrees.

Chapter 6

Phase Change

Matters can generally be classified as three basic phases such as solid, liquid, and gas, which are influenced by the pressure and temperature, or more specifically the energy. The process transits the phase of a matter from one to another is called phase change. In natural environments, snow melts when it contacts with hotter objects or receiving high pressure from the accumulations. The constantly varying phases create water, which serves as lubricants, making the behavior of wet snow completely different with dry snow. Consequently, simulating the phase change effect is necessary in our research. In this chapter, we will introduce our method of heat conduction and the particle transition algorithm.

6.1 Heat Conduction

In order to perform the phase change effect, it is necessary to compute how heat transfer between nearby particles. Typically, there are three mechanisms of heat transfer (advection, conduction and the radiation). In our research, we only simulate the conduction effect since the advection is automatically handled by SPH as particles move from one place to another. Considering having multiple materials (ice and water) in our simulation, we reference to the heat conduction model for heterogeneous material developed by Hochstetter et al. [21] that

$$\frac{dT_i}{dt} = \frac{1}{\rho_i c_i} \sum_j \frac{m_j}{\rho_j} \frac{4k_i k_j}{k_i + k_j} (T_j - T_i) \nabla^2 W_{viscosity}(\boldsymbol{x}_{ij}, h) \tag{6.1}$$

where c is the heat capacity, k_i, k_j is the heat conductivity of particle i, j, T is the temperature, and $W_{viscosity}$ is the viscosity kernel same as the one used in XSPH viscosity.

6.2 Particle Transition

6.2.1 Particle Blending

In practice, a different solver will be applied to perform the new characteristics when a particle is transferred from one phase to another. However, the sudden change on particle phase typically makes the simulation result unstable due to the difference at the applied constraints. As shown in Figure 6.1, when a fluid particle suddenly becomes a rigid particle, the nearby particles must be pushed away in order to satisfies the non-penetration constraint even though they might have fulfilled the density constraint in the previous time step. This

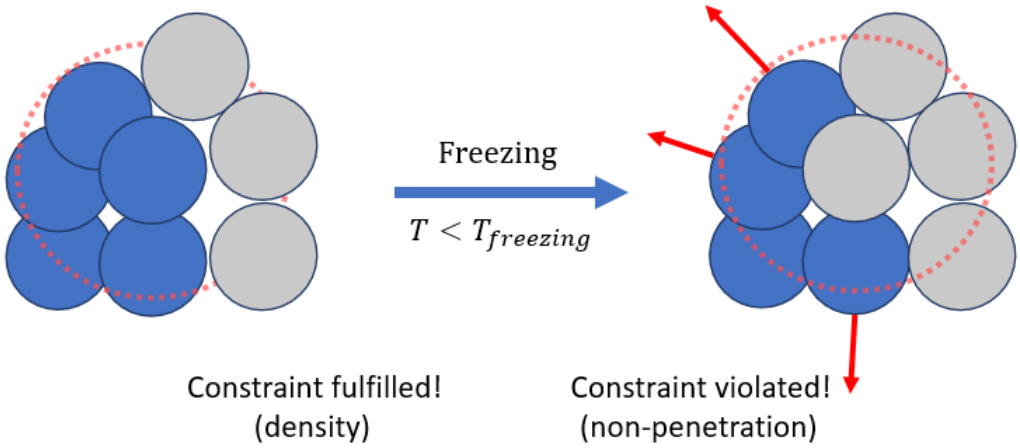


Figure 6.1: Unstable transition

phenomenon does not only occur in fluid-to-rigid transition, but also can be found in rigid-to-fluid transition. Therefore, we adopt the solution proposed by Wolde Lübke et al.[22] introducing a blending coefficient $0 \leq \alpha \leq 1$ and blending speed ω that

$$\alpha_t = \alpha_{t-1} + \omega \tag{6.2}$$

to mix the contributions computed from previous and current phases. For example, after taking the modification of the blending process, the correction of a newly transferred fluid particle becomes

$$\Delta \boldsymbol{x}_i = \alpha \Delta \boldsymbol{x}_{sph} + (1 - \alpha) \Delta \boldsymbol{x}_{dem} \tag{6.3}$$

where $\Delta \boldsymbol{x}_{sph}$ and $\Delta \boldsymbol{x}_{dem}$ is the correction computed by fluid solver and solid solver respectively. By doing this, the particle will act similar to the old phase at first and gradually be corrected until its position satisfying the constraint in the new phase.

6.2.2 Two-way Blending Mechanism

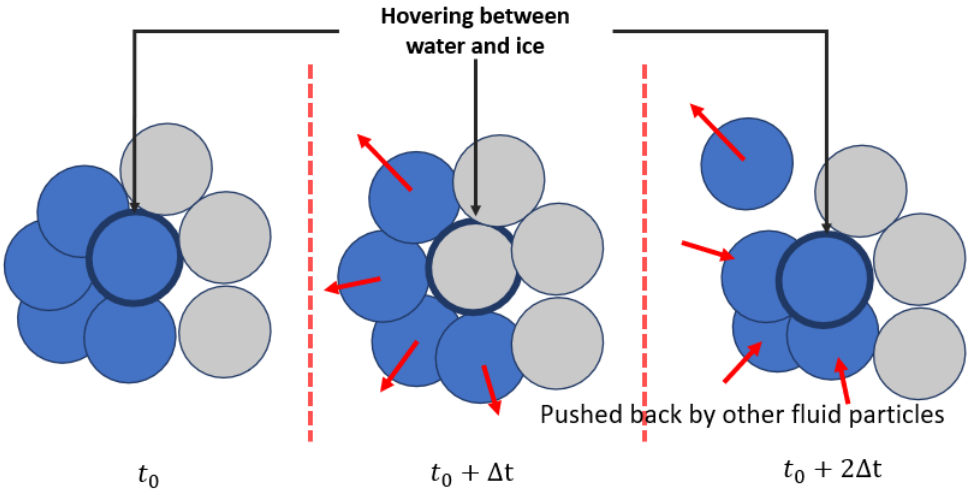


Figure 6.2: Rapid phase change

Since our target is to simulate the bilateral phase change effect, there is a special case we must take into consideration. Suppose that there is a particle whose temperature is near

to the melting point T_{melt} and locates just at the center between high and low temperature particles. Following the previous simulation flow, which directly changes the phase without considering the latent heat, can cause this particle hovering rapidly between two phases as shown in Figure 6.2.

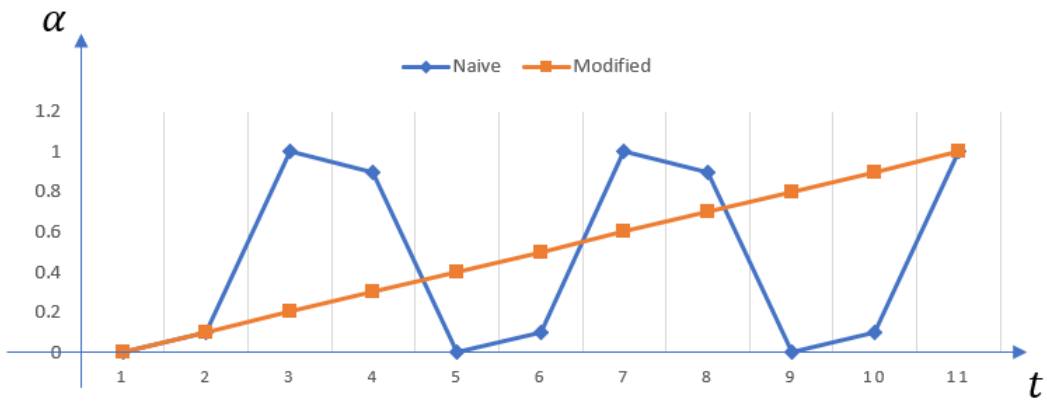


Figure 6.3: Blending threshold

Although we have additionally modified the correction with the blending parameter α, the simulation could still remain extremely unstable due to the sudden changes on the contribution. To address this problem, we use a threshold to mimic the latent heat effect which blocks the phase change procedural when

$$\alpha \leq \alpha_{threshold} \tag{6.4}$$

This mechanism helps the transition process to be flatten as shown in Figure 6.3. The only disadvantage is that it may not be as sensitive as using a true latent heat model. The phase change process may last for a couple of frames until it is unlocked.

6.2.3 Uneven Blending

Although we have resolved the instability problem with previous mentioned methods, we still noticed our bilateral phase change simulation was unstable when we were testing our program. We found that the correction contributed by density constraint and non-penetration constraint is unbalanced. This causes the revised corrections to be completely inclined to one of the contribution, more specifically, the non-penetration correction. Unlike the density constraint is sensitive to the number of nearby particles and their masses, the non-penetration constraint is sensitive to the distance of two particles. When the particles are very close and fulfill the density constraint, the influence constraint will be much stronger than the density constraint which means the correction of density constraint may be ignored due to the big difference on the value. Therefore, we adjust our blending process as the following

$$\Delta \boldsymbol{x}_{new\_sph} = \alpha \Delta \boldsymbol{x}_{sph} \tag{6.5}$$

for newly melted particles. And

$$\Delta \boldsymbol{x}_{new\_dem} = \alpha \Delta \boldsymbol{x}_{dem} + (1 - \alpha) \Delta \boldsymbol{x}_{sph} \tag{6.6}$$

for newly frozen particles. The reason why we only ignore the non-penetration correction in Equation 6.5 is that fluid is typically softer than solid. After rendered as transparent material,it is much more difficult to observe the existence of fluid particles near the boundary. In addition, as mentioned in Section 5.5, $\Delta \boldsymbol{x}_{sph}$ has constantly taken the non-penetration correction into consideration. Therefore, there is no need to emphasize the non-penetration effect in Equation 6.5.

6.2.4 Inter Connection Mechanism

As we described in Chapter 4, our model links DEM particles with stretch constraints in order to perform the elastoplastic property of snow. Nearby DEM particles j will be connected to particle i when

$$T_i \leq T_{linking} \cap T_j \leq T_{linking} \cap distance(x_i, x_j) \leq h \tag{6.7}$$

where $T_{linking}$ is the temperature a particle starts to link its neighbor, and h is the effective radius. Generally, $T_{linking}$ should locates in range $T_{linking} \leq T_{freezing}$ which means the links only exist in between low temperature DEM particles.

Since the physical memory is limited, naively recording all particles within h could be a very huge load for the system. Therefore, we use a maximum connection n_{max} to control the number of connections as illustrated in Figure 6.4. The connection record will be marked as occupied when the nearby particles access and fill in the empty spaces.

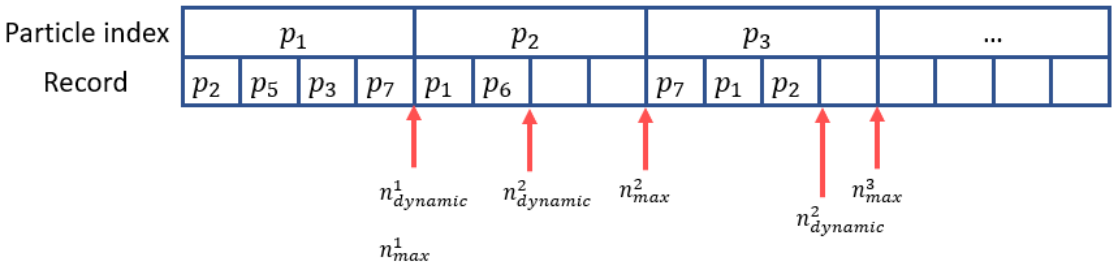


Figure 6.4: Connection control

However, restricting all particles with the same number of connection n_{max} results in ice-like effects which does not fit with the fluffy image of snow. Therefore, we introduce an dynamic connection number $n_{dynamic}$ that

$$n_{dynamic} = Round(\frac{T - T_{freezing}}{T_{homo} - T_{freezing}})n_{max} \tag{6.8}$$

where T_{homo} is the homogeneous freezing temperature that a whole material will freeze uniformly when the temperature is below T_{homo}. By doing this, a DEM particle can control the number of connection based on the current temperature T. In Chapter 7.2.3, we will discuss our GPU implementation of how we fill in the record in parallel computation.

Chapter 7

GPU Implementation

Graphics Processing Unit (GPU) is a computer hardware which is firstly designed to concurrently execute pipelined instructions in order to increase the performance of image generation. In recent years, many applications has utilized the nature of parallel execution to accelerate their solutions such as video encoding, or machine learning. In this chapter, we will introduce the programming model of GPU. After then, we will discuss our phase change algorithm and the inter connection mechanism on GPU.

7.1 Programming Model

GPGPU is a technique that processes general-purpose computations on GPU. Various APIs have been provided to support the use of GPGPU. CUDA is the application programming interface developed by NVIDIA that supports developers to use c-style code to execute concurrent programs on GPU. The programming model of CUDA is illustrated in Figure 7.1. In CUDA, a computational grid is composed of multiple thread blocks where each block contains multiple threads and each thread launches a *kernel* function in parallel.

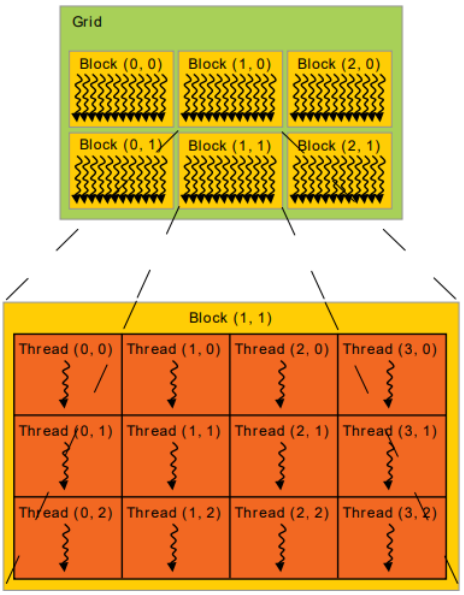


Figure 7.1: CUDA Programming Model (referenced from [1])

There are three types of memory in CUDA (Figure 7.2). Each thread has its own local memory and registers for read/write purpose. Inside each blocks, there is a shared memory for threads to exchange information cooperatively. Global memory is the only one can be read/written by CPU and will be used when the parameters need to be visible across kernels of multiple blocks.

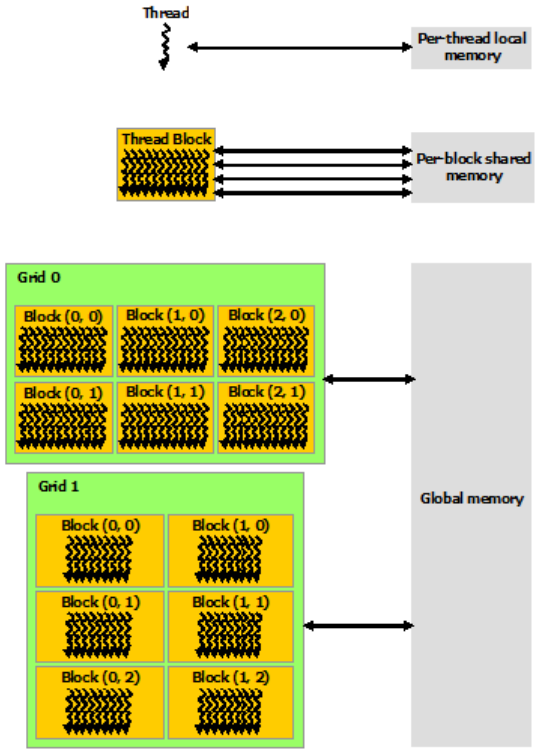


Figure 7.2: CUDA Memory Hierarchy (referenced from [1])

Before launching a CUDA program on GPU, all essential data must be transferred from host (CPU) to device (GPU). The data of host and device are stored independently until the OS calls the driver to synchronize them. The I/O overhead is the biggest bottleneck when there are too much of them. In our research, we provide an algorithm completely designed for GPU computation. Except necessary I/Os for parameter settings, the whole simulation is performed on the device.

7.2 GPU-based Phase Change

7.2.1 Predicates

To efficiently handle the phase change effect on GPU, we store the information of particles into two different structure of arrays like the following:

```
struct ParticleDeviceData
{
  float3* m_d_positions;
  float3* m_d_velocity;
//...
    /* heat conduction parameters */
  float*  m_d_T;
  float*  m_d_new_T;
//...
```

```
    /* inter-connection parameters*/
  uint*   m_d_connect_record;
  uint*   m_d_iter_end;
  float*  m_d_connect_length;
//...
    /* scan and compact parameters */
    uint*   m_d_predicate;
    uint*   m_d_scan_index;
}
```

where each particle set represents a specific phase with *predicate* to determine whether this particle is going to be used in this time step. Initially, we allocate two $m + n$ arrays which save m SPH particles and n DEM particles with reserved extra spaces for the particle transitions as shown in Figure 7.3.

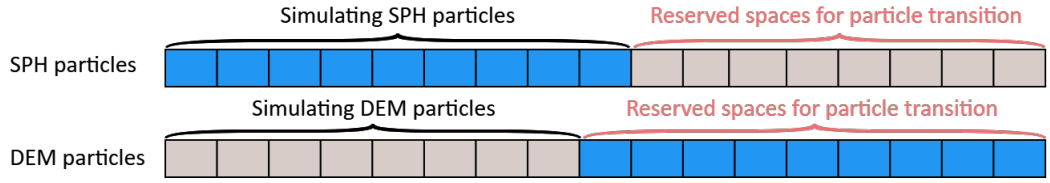


Figure 7.3: Memeory view

When the phase of a particle changes, the data will be copied to the tail of corresponding array like Figure 7.4. Then, the old place will be marked as 0 (not using) and the new place will be marked as 1 (using).

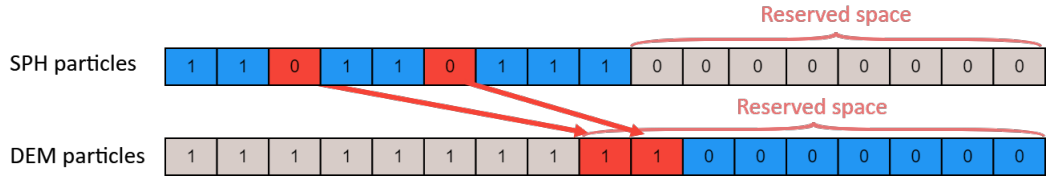


Figure 7.4: Particle transition

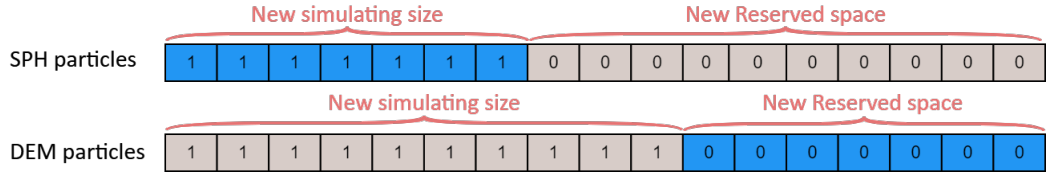


Figure 7.5: Particle transition (compact process)

Based on the mark, we move those 0s to the back and 1s to the front as shown in Figure 7.5. There are several advantages of having contiguous region of data. Firstly, it is much easier to manage the data by iteratively traversing the array. Secondly, the computation is much faster because coalesced memory access on GPU is far efficient than strided access.

7.2.2 Scan and Compact

In GPU computation, the compact process can be accomplished by using the exclusive scan [23]. As illustrated in Table 7.1, each element i in the output array is the sum of all previous

elements in the input array. By extending this feature, we perform the exclusive scan to the

Input	3	1	4	1	5	9	2	6
Output	0	3	4	8	9	14	23	25

Table 7.1: Exclusive scan

predicate array mentioned in the previous subsection which generates outputs, for example, as in Table 7.2. Using this output, we can obtain the index that scatters the original data

Input	1	1	0	1	1	0	1	1
Output	0	1	2	2	3	4	4	5

Table 7.2: Exclusive scan result of the predicate array

into the compacted data as Figure 7.6. We use this mechanism to implement our phase

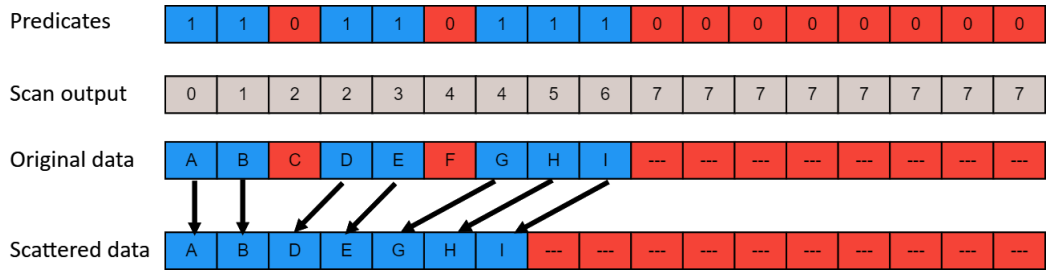


Figure 7.6: Scatter

change algorithm by exchanging particles in between two particle sets. Once we have the compacted particle data, we can reduce the computation cost by only traversing the useful data.

7.2.3 Lock-free Connection Mechanism

To prevent the read/write issues on writing connection record, we use the *atomicCAS* (compare and swap) operation to protect the corresponding data. Once a particle finds an empty space (UINT_MAX), the *atomicCAS* operation will swap the value of the record with the particle *index1*. After then, the current distance of two particles *dist* will be store in the corresponding places and the *iter_end*, which serves as the current usage of connections, will be incremented. If there is no space for *index1* to fill in, then the record will be reset to UINT_MAX (empty). The following code shows how to fill in the empty space:

```
//  CUDA atomic operations returns old values
int atomicCAS(int* address, int compare, int val);
int atomicExch(int* address, int val);
int atomicAdd(int* address, int val);

...

// traverse all record spaces to find empty to fill
for (uint i = n_max * index0; i < n_max * (index0 + 1); ++i)
{
    if (atomicCAS(&connect_record[i], UINT_MAX, index1)  == UINT_MAX)
    {
```

```
13          bool valid = false;
14
15          // search and fill in index1's record to prevent one-way connection
16          if (iter_end[index1] < (index1 + 1) * n_max)
17          {
18              // locate avaible space at index1's record and fill
19              for (uint j = index1 * n_max; j < (index1 + 1) * n_max; ++j)
20              {
21                  if (atomicCAS(&connect_record[j],
22                                UINT_MAX,
23                                index0) == UINT_MAX)
24                  {
25                      connect_length[j] = dist;
26                      atomicAdd(&iter_end[index1], 1);
27                      valid = true;
28                      break;
29                  }
30              }
31          }
32          if (valid)
33          {
34              connect_length[i] = dist;
35              atomicAdd(&iter_end[index0], 1u);
36          } else
37          {
38              // reset record index if invalid
39              atomicExch(&connect_record[i], UINT_MAX);
40          }
41      }
42  }
```

When the other threads are trying to access the same record after the value has been changed by the *atomicCAS* (line 11 and line 21), other accesses will be rejected because the space there is no longer marked as available (UINT_MAX). Those threads will then iterate to the next location to find whether there is an empty space or not until it reaches the end of the record memory (determined by Equation 6.8). After all threads finish the connection creation process, the connection count will be updated so that we can directly jump over all excessive connection attempts in the next time step by referencing to the connection count.

Chapter 8

Results & Conclusion

This chapter shows our simulation results in Section 8.1 which covers experiments testing the functions of our research. Then, we will make the conclusion, discussion of the limitations, and the future work in Section 8.2.

8.1 Results

8.1.1 Environment

Our experiment environment is listed in Table 8.1. The snow particles is the rendered as point sprites and the water is rendered using Screen Space Fluid Rendering [24].

CPU	Intel® Core™ i7-8700
RAM	16 GB
GPU	NVIDIA GeForce GTX 1080
VRAM	8 GB
Graphics API	OpenGL 4.1
CUDA version	11.0

Table 8.1: Environment

Table 8.2 is the global parameters for SPH particles.

Parameter	value
Particle mass	0.01 (kg)
Fluid density	1000 $(kg/m^3))$
Effective radius	0.0362783(m)
Per kernel particles	20

Table 8.2: SPH parameters

Both Table 8.3 and Table 8.4 are unchanged parameter simulation while the former table is the heat-relative parameters and the latter is the PBD parameters in all below simulations. Notice that PBD parameters do not have units because there is no actual physics-based material used in PBD.

Parameter	Value
Time step	$0.00125(sec)$
Number of iterations	3
Snow heat capacity	2090 $(J/kg\ C^{\circ})$
Water heat capacity	4182 $(J/kg\ C^{\circ})$
Melting point	0 C°
Homogeneous freezing temperature	-30 C°

Table 8.3: Unchanged heat parameters

Parameter	Value
SPH viscosity	0.01
DEM viscosity	0.1
Blending speed	0.01
Static friction	0.5
Kinetic friction	0.35
Interlink stiffness	0.1
Non-penetration stiffness	0.25
SOR coefficient	0.25

Table 8.4: Unchanged PBD parameters

8.1.2 Snow drop

This subsection shows our result of interlink usage for snow simulation. The temperature of snow in the following experiments is set to $-10\ C^{\circ}$. Figure 8.1 demonstrates the simulation without using interlinks. Figure 8.2 uses interlinks for DEM particles but the number of connection is always a constant. Figure 8.3 changes the number of connection based on the homogeneous freezing temperature T_{homo} and the temperature of particle.

Comparing with none interlinked snow drop, experiments using interlinks shows snow is be able to keep its shape after dropping down to the ground which performs the elastoplastic property. Also, in Figure 8.3, we can observe that snow can be fluffy but not lose the ability to keep its shape by using the dynamic number of connection we proposed in Chapter 6.2.4.

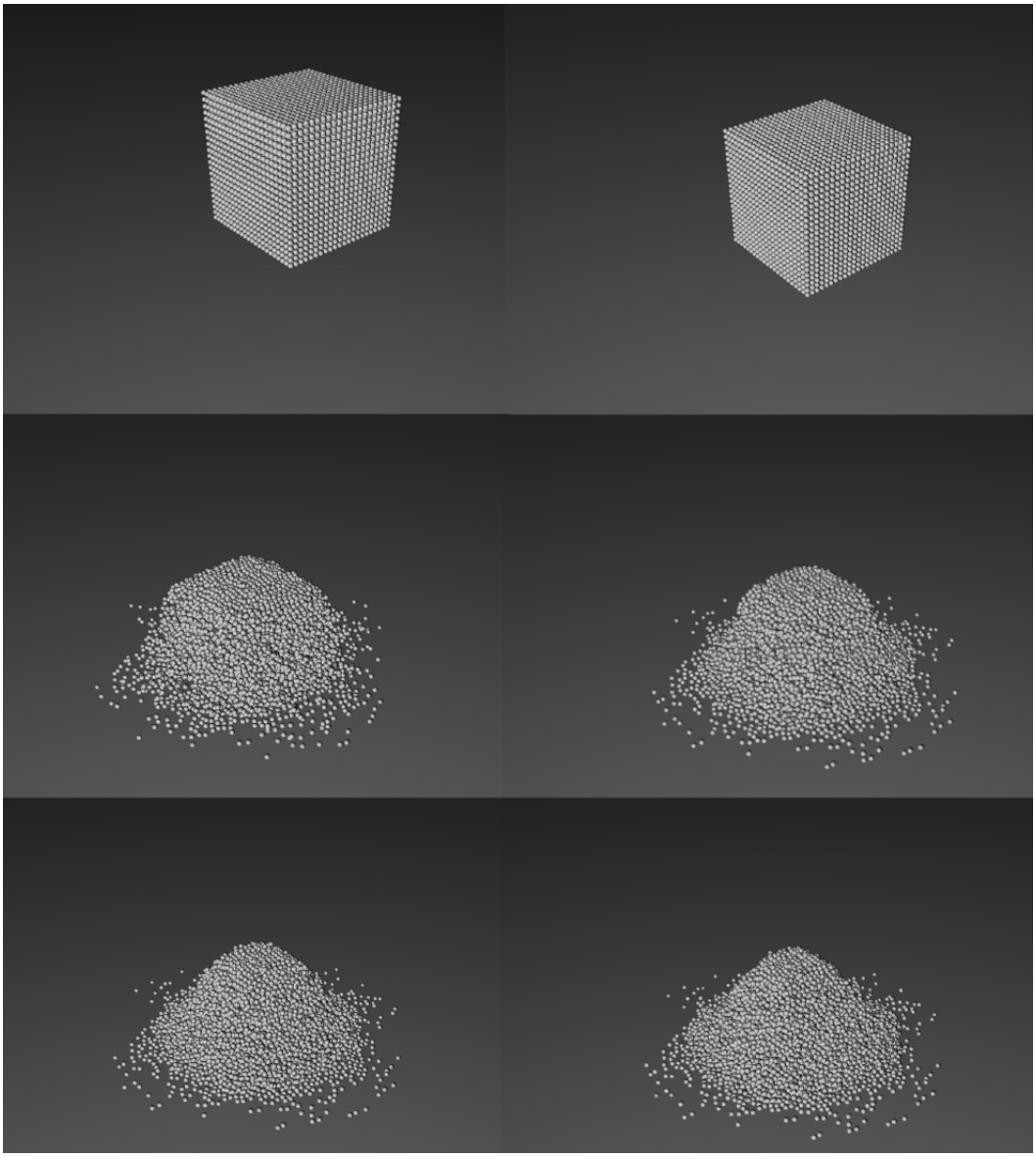


Figure 8.1: None interlink snow drop (200 frames per image)

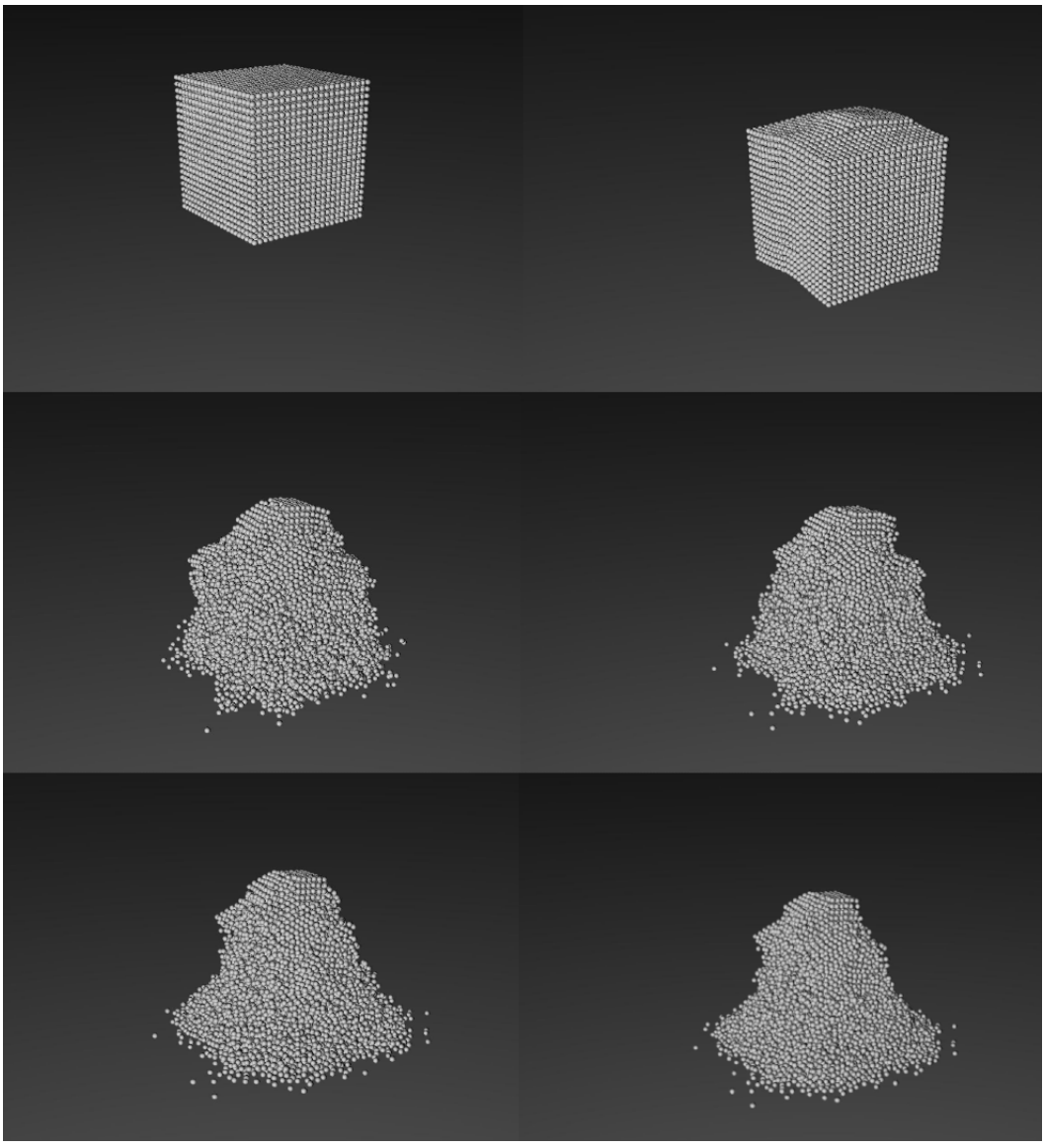


Figure 8.2: Full interlinked snow drop (200 frames per image)

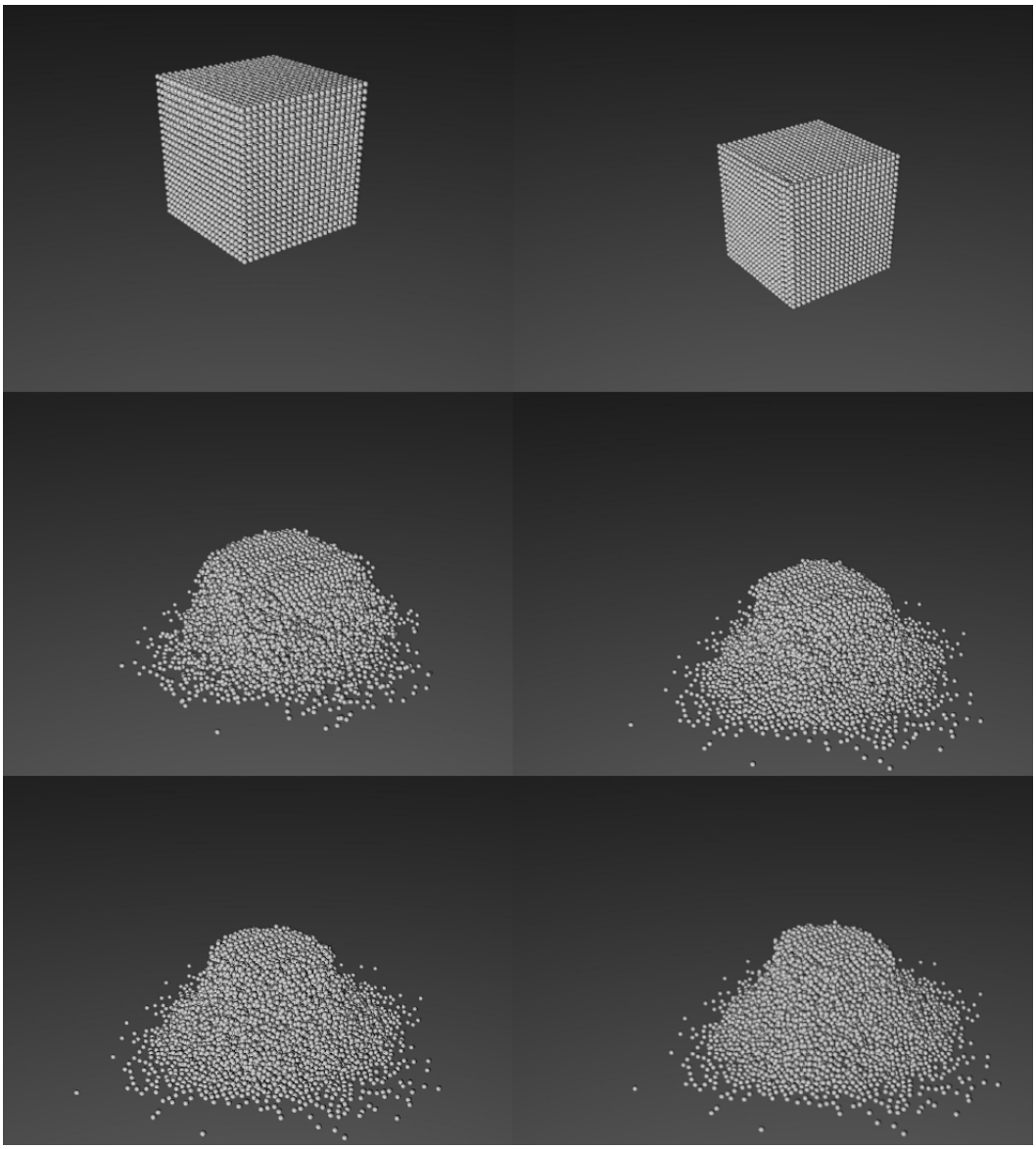


Figure 8.3: Dynamic interlinked snow drop (200 frames per image)

8.1.3 Snow drop with melting

The scene shown in Figure 8.5 demonstrates the melting process of snow drop into hot water. Figure 8.6 is the thermograph of the last frame in Figure 8.5. Figure 8.4 shows the processing time from frame 0 to frame 5000. Table 8.5 lists our simulation parameters of this experiment. And Figure 8.4 illustrates the transition of the number of particles. In order to accelerate the simulation, we set the heat conductivity 100 times bigger than regular settings.

From Figure 8.5, we can observe the size of snow shrinks over time. This shows our melting process is successfully executed. In the performance graph (Figure 8.4), the computation time increases when the snow contacts with the hot water because the solver starts to handle the phase change effect. In Figure 8.7, we can see that DEM particles gradually becomes SPH particles as the heat transferred between particles as shown in Figure 8.6. Notice that the jitters in Figure 8.4 are caused by the number of simulated interlinks and the difference of execution time of each CUDA kernel.

Parameter	Value
Snow temperature	-10 $(C°)$
Water temperature	50 $(C°)$
Snow heat conductivity	2500 (W/mK)
Water heat conductivity	600 (W/mK)
Number of water particles	64000
Number of snow particles	10648

Table 8.5: Parameters of snow drop melting scene

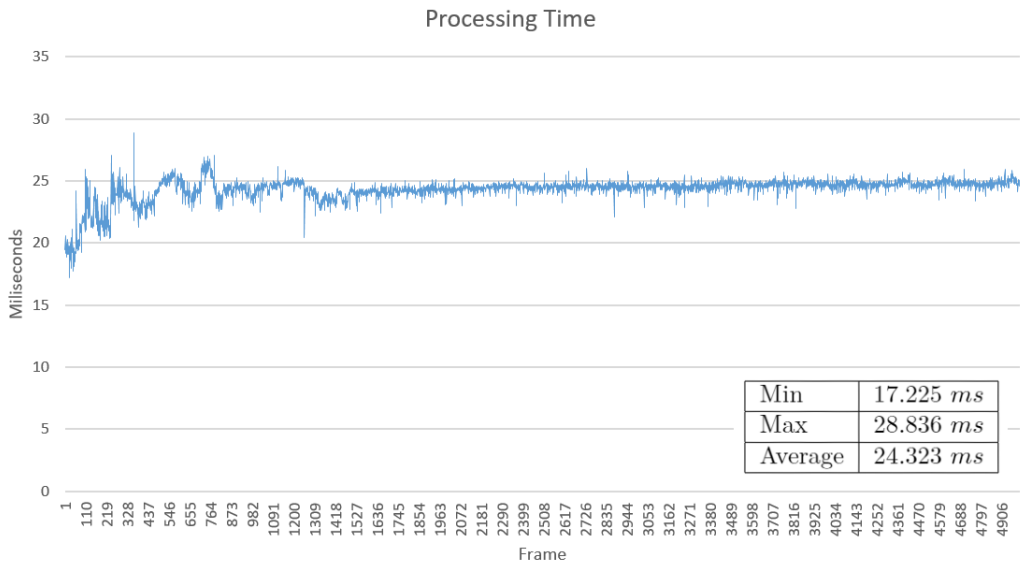


Figure 8.4: Performance of snow drop melting

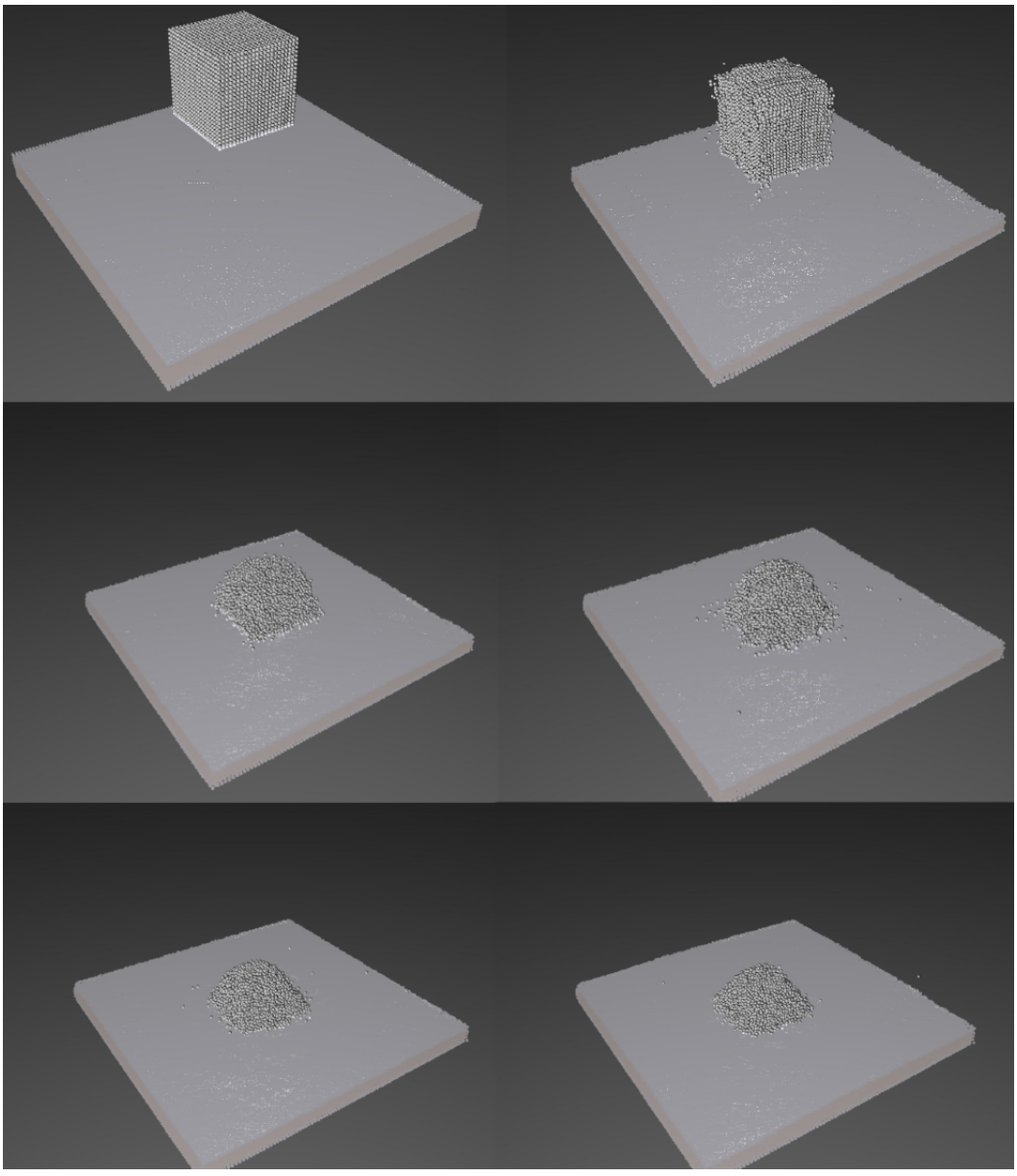


Figure 8.5: Snow drop melting (200 frames per image)

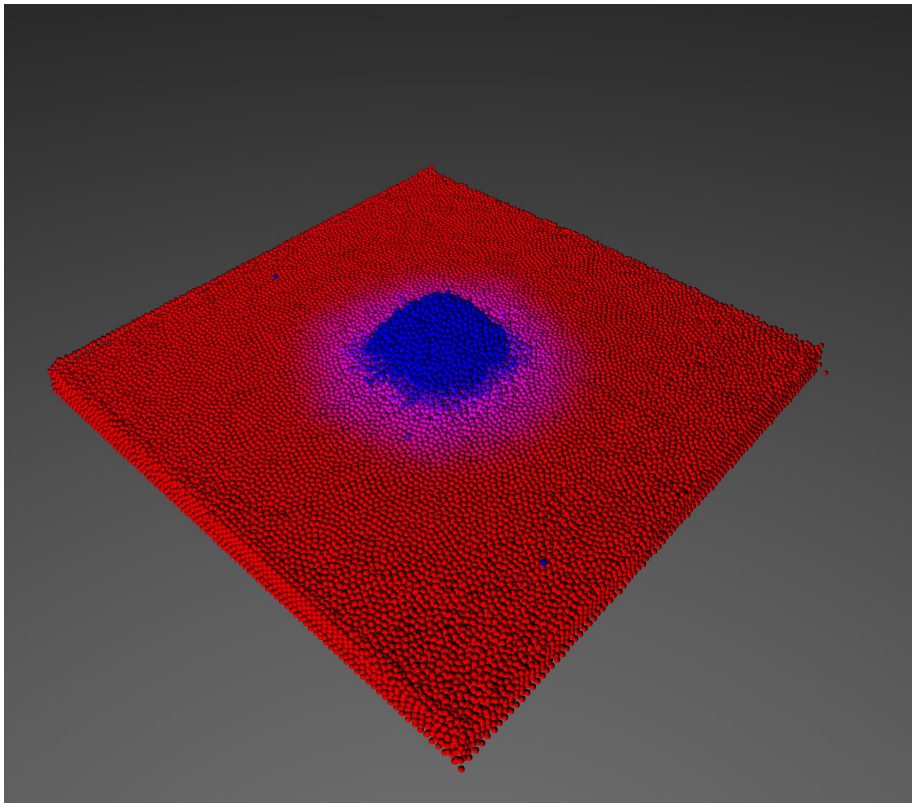


Figure 8.6: Snow drop melting (Thermograph)

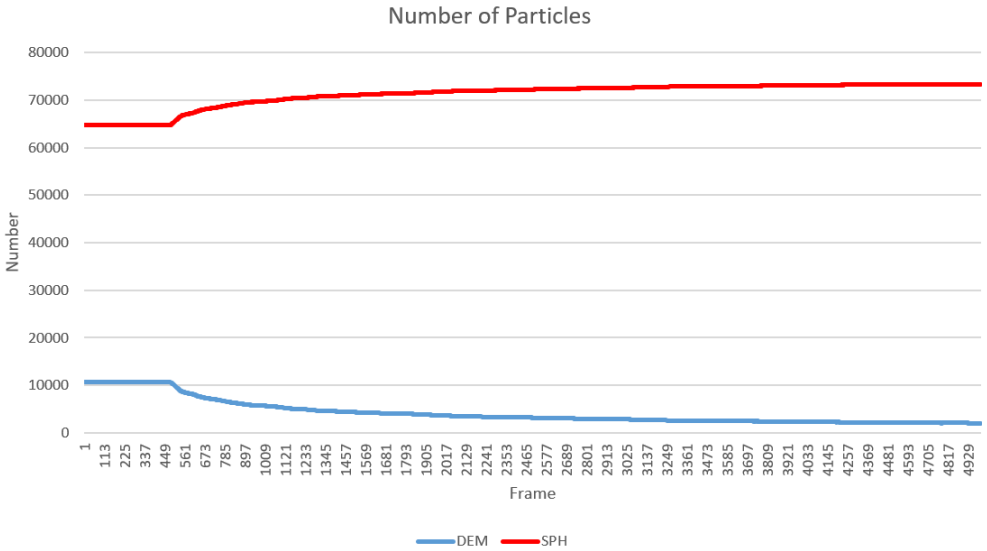


Figure 8.7: Number of particles of snow melting in water

8.1.4 Snow Drop Freezing

This scene shows the freezing process of cold snow drop in low temperature water as illustrated in Figure 8.9. Figure 8.10 is the thermograph of the frame 5000. Figure 8.8 is the processing time per frame from frame 0 to frame 5000. And Figure 8.8 illustrates the transition of the number of particles.

In contrast with Section 8.1.3, this experiment demonstrates our freezing process is also successfully proceeded as well. Comparing with Figure 8.4, the processing time is more stable in Figure 8.8. This is mainly because the snow temperature in this experiment is under the homogeneous freezing temperature T_{homo} in Chapter 6.2.4 so that the processing time for interlinks did not vary a lot even when the shape changed. Another thing to notice is that the shape of snow is twisted from frame 400. It is simply because the connection of a particle is less as it is near to the edge. Therefore, they are typically able to fall down without the correction from other particles which results in the deformed shape as shown in Figure 8.9. To prevent this artifact, we need to introduce another constraint such as volume constraint to keep its shape.

Parameter	Value
Snow temperature	-100 $(C°)$
Water temperature	1 $(C°)$
Snow heat conductivity	2500 (W/mK)
Water heat conductivity	600 (W/mK)
Number of water particles	32400
Number of snow particles	10648

Table 8.6: Parameters of snow drop freezing scene

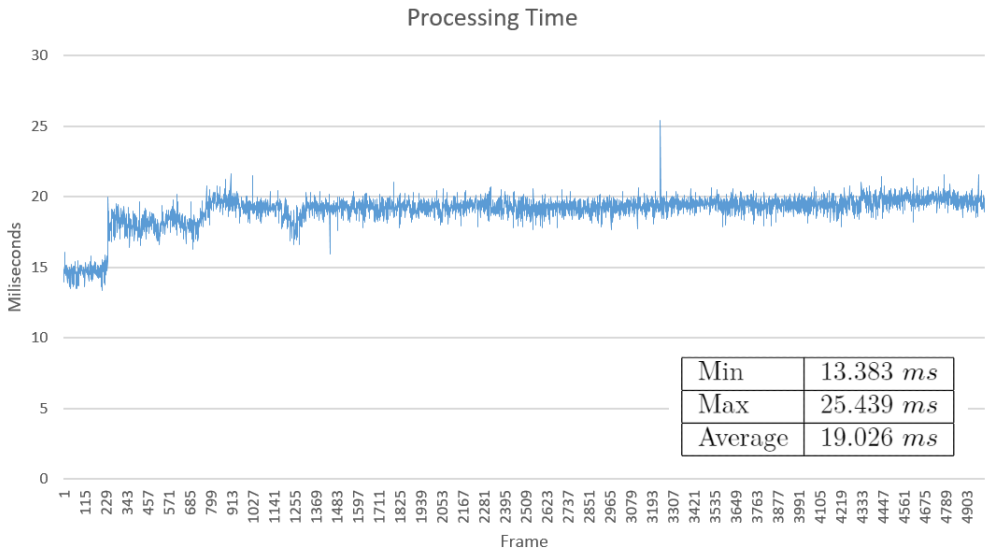


Figure 8.8: Performance of snow drop freezing

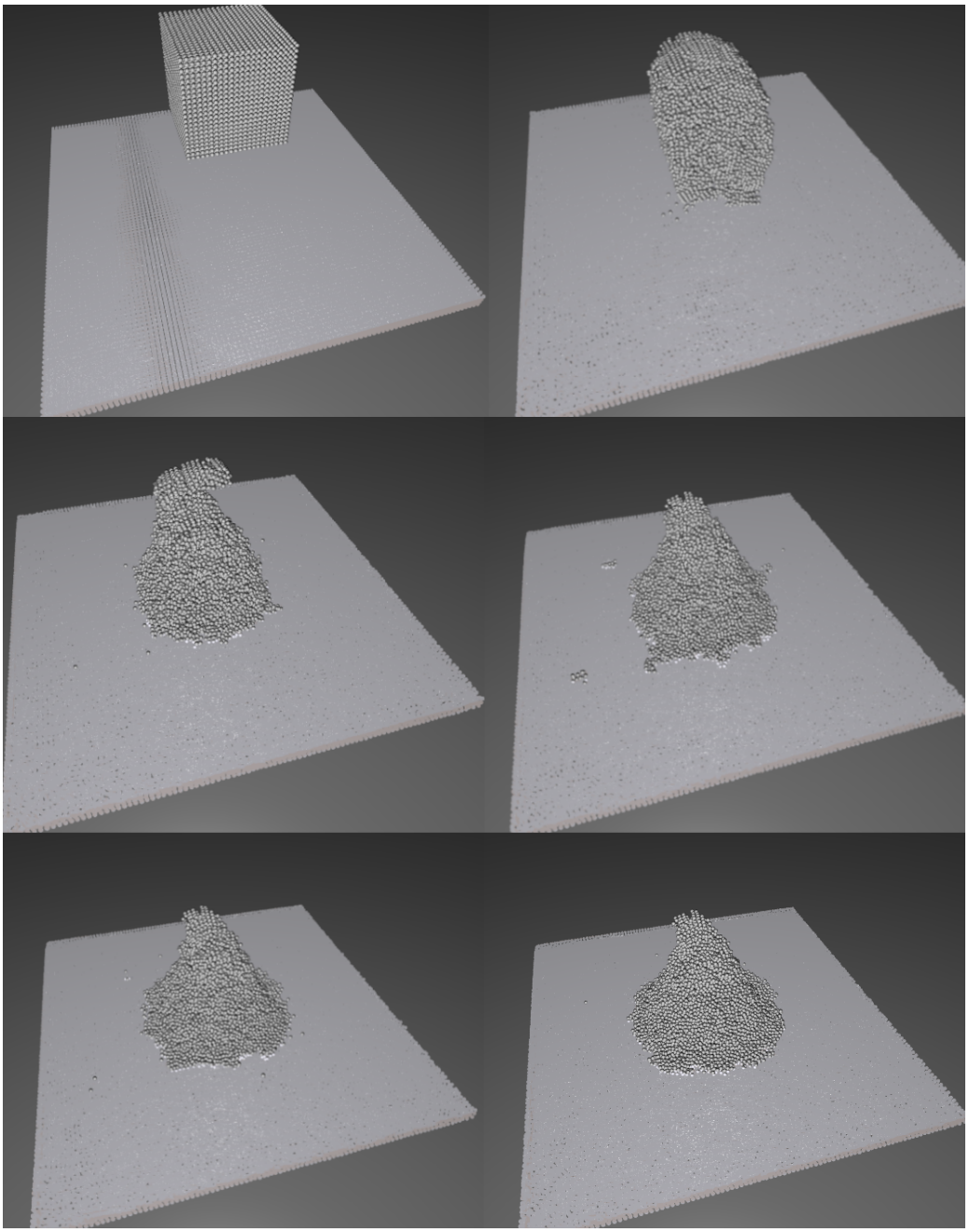


Figure 8.9: Snow drop freezing (400 frames per image)

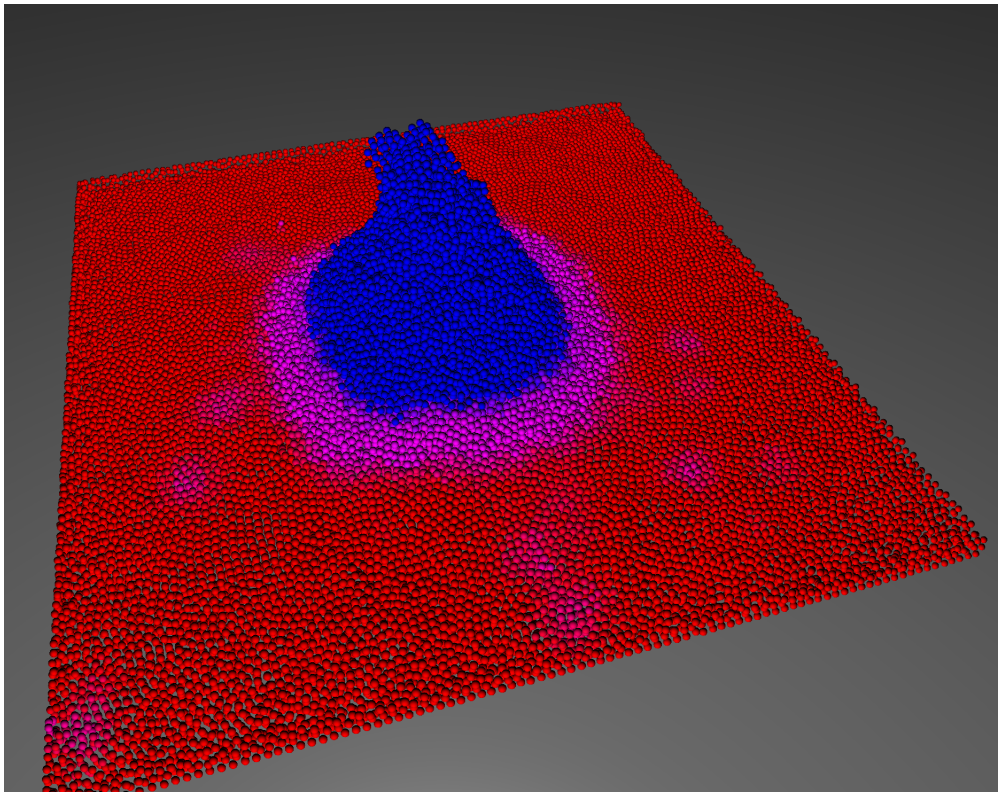


Figure 8.10: Snow drop freezing (Thermograph)

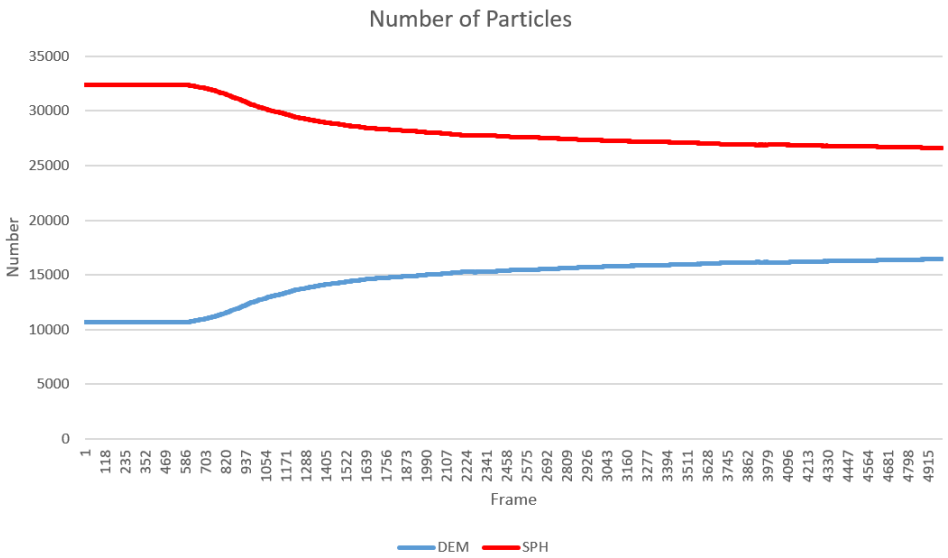


Figure 8.11: Number of particles of water frozen by snow

8.1.5 Water drop into snow

This experiment shows the scene that water drops into the snow. The parameters we used in this simulation are listed at below Table 8.7. Figure 8.13 and Figure 8.14 are the results captured every 500 frames. Figure 8.12 shows the per frame processing time of water drop scene. And Figure 8.15 illustrates the transitions of the number of particles. Notice that the high temperature and conductivity are simply used to exaggerate the effect of snow being melted by hot water drops.

In Figure 8.15, we can observe that the snow has drastic phase change effect starting from about frame 400 to frame 1000. This increases the processing time a lot as shown in Figure 8.12. Also, the computation cost increases as long as the SPH particles increase because it typically takes more steps than DEM particles. From about frame 3700, the freezing speed starts exceeding the melting speed. This tells us our phase change algorithm is a bilateral process instead of a unilateral process.

Finally, we output our result into Houdini to generate the surfaces and rendered with ray tracing as shown in Figure 8.16.

Parameter	Value
Snow temperature	-10 $(C°)$
Water temperature	100 $(C°)$
Snow heat conductivity	2500 (W/mK)
Water heat conductivity	600 (W/mK)
Number of water particles	5888
Number of snow particles	147928

Table 8.7: Parameters of water drop scene

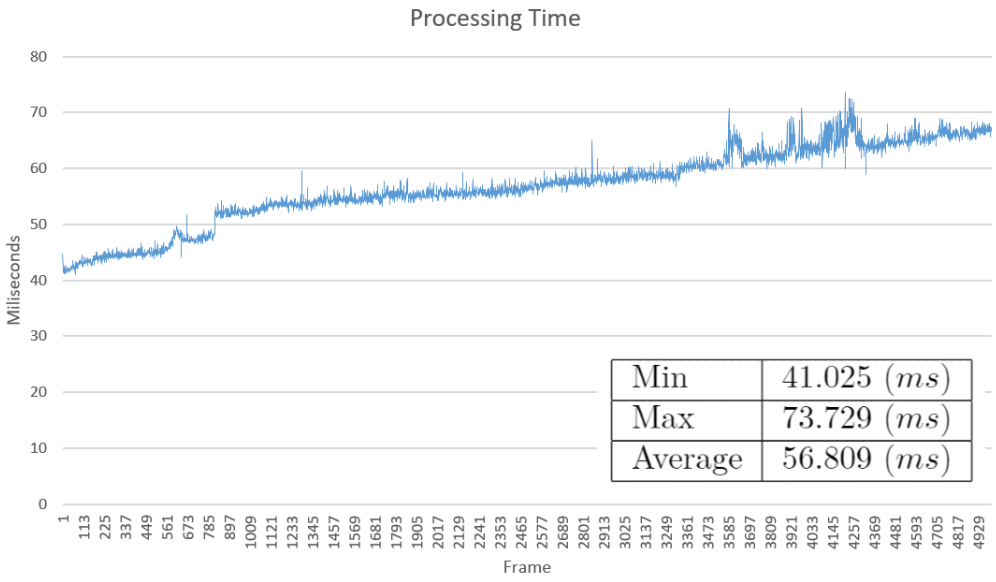


Figure 8.12: Performance of water drop

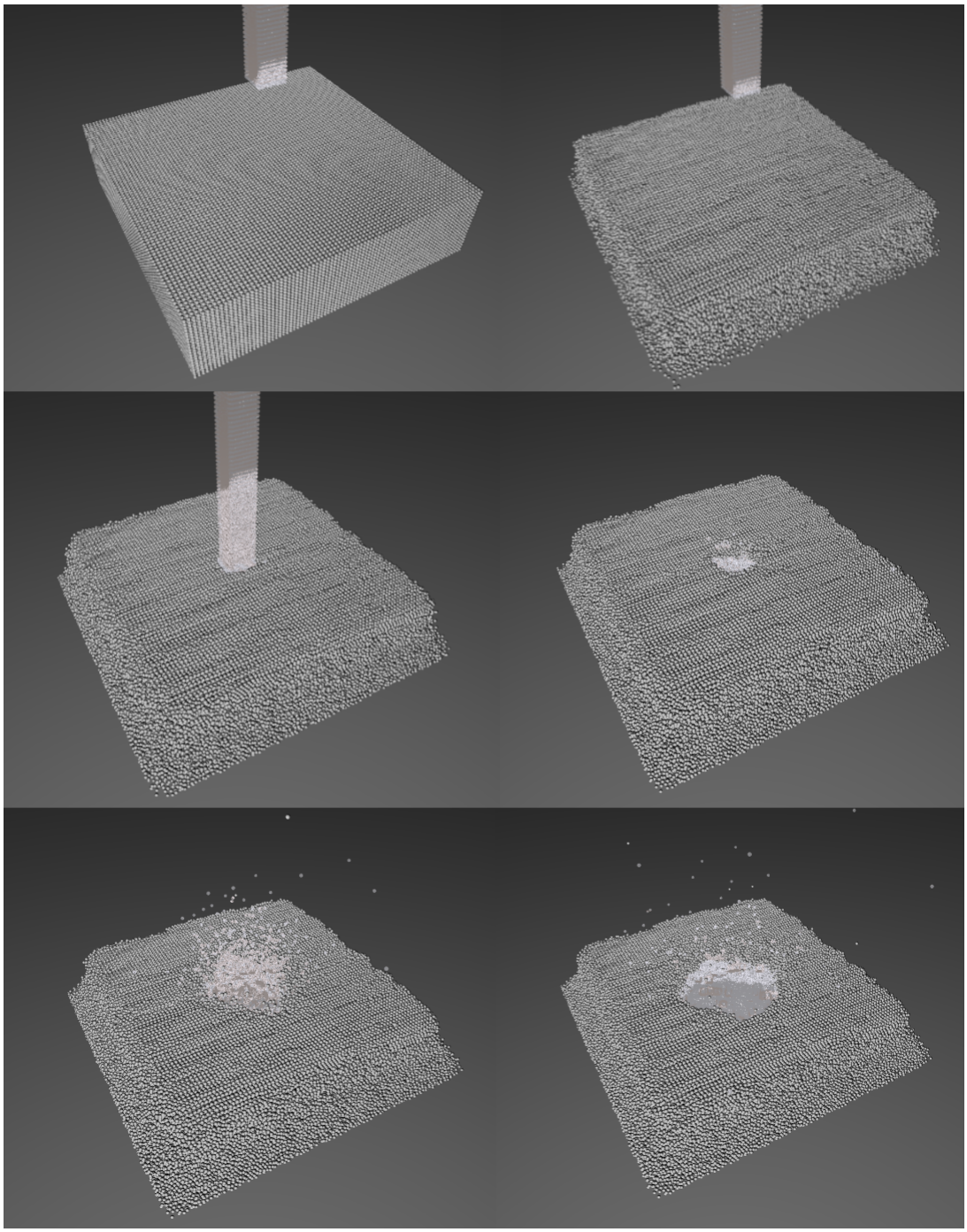


Figure 8.13: Water drop 1 (250 frames per image)

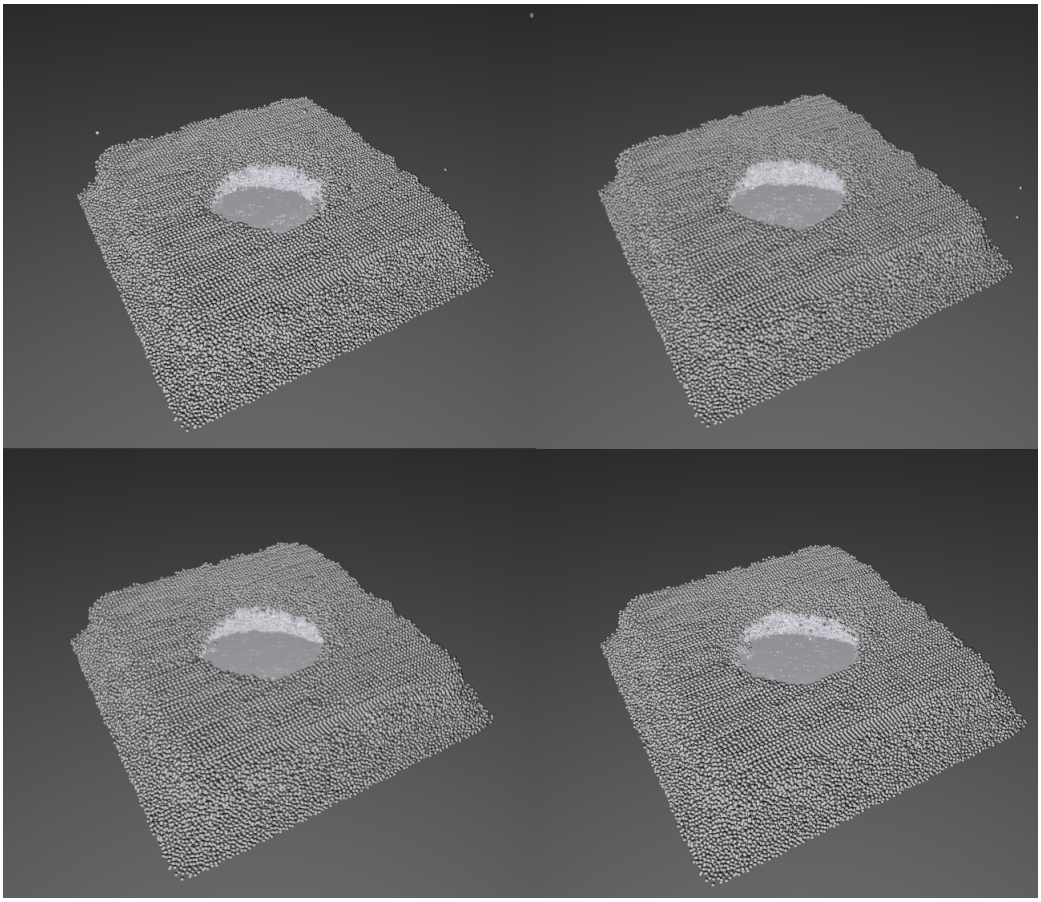


Figure 8.14: Water drop 2 (250 frames per image)

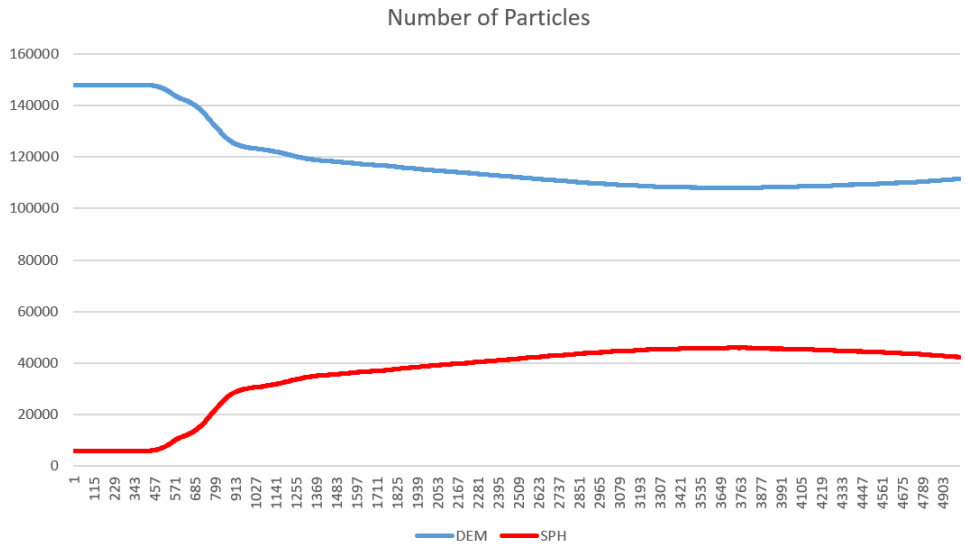


Figure 8.15: Water drop particle numbers

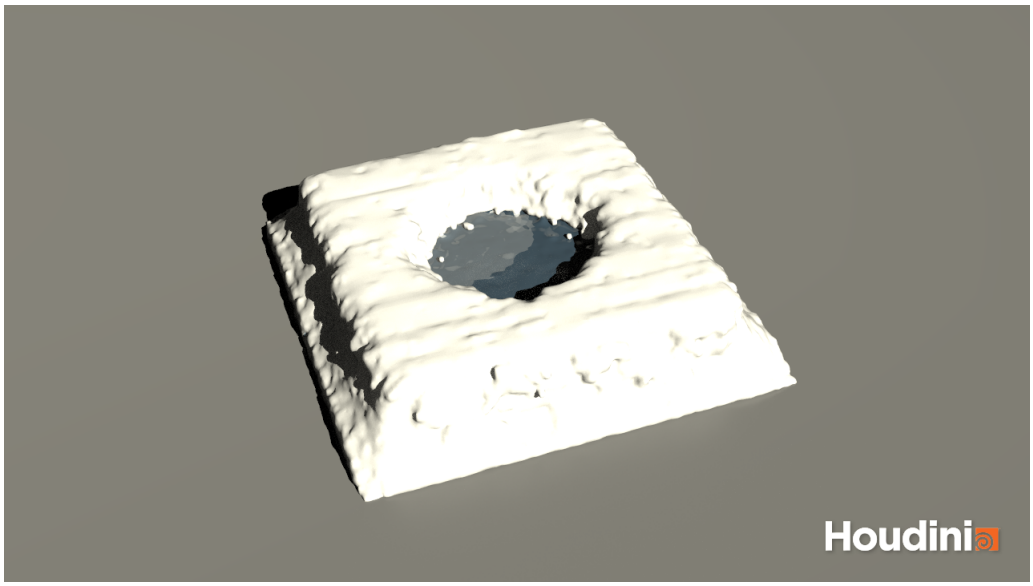


Figure 8.16: Water drop scene (Rendered with Houdini)

8.2 Conclusion & Discussion

In conclusion, our research utilizes the Position Based Dynamics to simulate the behavior of snow which performs the rigid body collision and the elastoplastic characteristic of snow material. Our algorithm shows that the implementation of intense bilateral phase change effect is applicable on GPU. By separating snow into two sets of particles, our solver is able to handle the interaction of fluid and solid with the averaged correction of density constraint, non-penetration constraint and the stretch constraint. Furthermore, we use the blending parameters to smooth the sudden changes caused by varying phases. As a result, our position-based solver provides a stable and controllable solution for snow simulation by considering the phase change effects.

However, there are several limitations in our research. One of the problems is the usage of the stretch constraint. As long as the distance of two particles remains the same, the correction will not be applied because the stretch constraint does not take the topological relationships into consideration. This should be fixed with using the shape matching algorithms [25] or volume constraint. Another limitation is that our method does not perform the infiltration effects well, such as water flow into snow particles, when the number of particles is not sufficient because of the uniform size of the DEM and SPH particles. We think using adaptive size for SPH particles should be able to resolve this problem [26]. Other limitations such as large scale rigid body collisions requires shock propagation in order to converge faster as mentioned in Macklin's paper [20]. We ignore this problem because of the complexity of implementation. Since our simulation incorporates the interaction with fluid, the direction of shock propagation would be hard to estimate. Therefore, we simply change the stiffness of non-penetration constraint to achieve a closer visual effect of snow simulation.

In the future, we would firstly put our effort on reducing the effort of parameter tuning. By using XPBD [27], it should be able to allow users to use real-world parameters in the simulation, especially for the stiffness. And since our simulator is totally designed for GPU computation, the information of environmental objects on CPU should be synchronized to GPU in order to make them be able to interact with the snow.

Acknowledgement

Foremost, I would like to express my sincere appreciation to my supervisor, Professor Makoto FUJISAWA, who continuously shows me constructive hints, directions and materials that pushed me to finish this research in this year. Except those assists related to my research, I would also like to thank him for trusting me be capable of finishing my research. Also, I would like to thank Professor Masahiko MIKAWA, and all the members in both PBCGLab and Social Robotics Laboratory. The discussions you gave me in the seminar truly helped me noticing the details I should be aware of when I am constructing my research.

Furthermore, I would like to thank my mentors in Taiwan National Cheng Kung University, Professor Min-Chun Hu, who had encouraged me to pursue my dream of doing physics-based computer graphics research, and Jim Huang, who taught me those important knowledge of computer science. Without your supports, it would be impossible for me to have a chance to conduct my research in University of Tsukuba.

Lastly, I would like to thank my friends and family who have supported me all the way to here. I sincerely appreciate for your physical and mental endorsement that helped me to overcome my frustrations. It is my honor to have you standing by my side.

References

[1] Nvidia. Cuda c programming guide. `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`. Accessed: 2020-12-25.

[2] Alexey Stomakhin, Craig Schroeder, Chenfanfu Jiang, Lawrence Chai, Joseph Teran, and Andrew Selle. Augmented MPM for phase-change and varied materials. *ACM Transactions on Graphics*, Vol. 33, No. 4, pp. 1–11, jul 2014.

[3] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, Vol. 18, No. 2, pp. 109–118, apr 2007.

[4] Miles Macklin and Matthias Müller. Position based fluids. *ACM Transactions on Graphics*, Vol. 32, No. 4, pp. 104:1–104:12, jul 2013.

[5] Tomoyuki Nishita, Hiroshi Iwasaki, Yoshinori Dobashi, and Eihachiro Nakamae. A modeling and rendering method for snow by using metaballs. *Computer Graphics Forum*, Vol. 16, No. 3, pp. C357–C364, sep 1997.

[6] Paul Fearing. Computer modelling of fallen snow. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00*, pp. 37–46, jul 2000.

[7] N. Maréchal, E. Guérin, E. Galin, S. Mérillou, and N. Mérillou. Heat transfer simulation for modeling realistic winter sceneries. *Computer Graphics Forum*, Vol. 29, No. 2, pp. 449–458, may 2010.

[8] 高橋哲也, 藤代一成. 焼結作用を考慮した雪の踏み散らしシミュレーション. 第 74 回全国大会講演論文集, 第 2012 巻, pp. 143–144, mar 2012.

[9] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Transactions on Graphics*, Vol. 24, No. 3, pp. 965–972, jul 2005.

[10] Christoph Gissler, Andreas Henne, Stefan Band, Andreas Peer, and Matthhias Teschner. An implicit compressible sph solver for snow simulation. *ACM Transactions on Graphics*, Vol. 39, No. 4, pp. 1–16, aug 2020.

[11] N. Foster and D. Metaxas. Controlling fluid animation. In *Proceedings of Computer Graphics International*, pp. 178–188, 1997.

[12] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH '01*, pp. 23–30, 2001.

[13] Mark Carlson, Peter J. Mucha, and Greg Turk. Rigid fluid. *ACM Transactions on Graphics*, Vol. 23, No. 3, pp. 377–384, aug 2004.

[14] Witawat Rungjiratananon, Zoltan Szego, Yoshihiro Kanamori, and Tomoyuki Nishita. Real-time animation of sand-water interaction. *Computer Graphics Forum*, Vol. 27, No. 7, pp. 1887–1893, oct 2008.

[15] Christopher Jon Horvath and Barbara Solenthaler. Mass preserving multi-scale sph. Pixar Technical Memo 13-04, Pixar Animation Studios, 2013.

[16] Nadir Akinci, Markus Ihmsen, Gizem Akinci, Barbara Solenthaler, and Matthias Teschner. Versatile rigid-fluid coupling for incompressible SPH. *ACM Transactions on Graphics*, Vol. 31, No. 4, pp. 1–8, aug 2012.

[17] Makoto Fujisawa and Kenjiro T. Miura. Animation of ice melting phenomenon based on thermodynamics with thermal radiation. In *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia - GRAPHITE '07*, pp. 249–256, 2007.

[18] 仲宗根良. 位置ベース粒子法を用いた高速な融解シミュレーション. Master's thesis, University of Tsukuba, March 2016.

[19] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, p. 154–159, 2003.

[20] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics*, Vol. 33, No. 4, pp. 1–12, jul 2014.

[21] Hendrik Hochstetter and Andreas Kolb. Evaporation and condensation of SPH-based fluids. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pp. 1–9, jul 2017.

[22] Arno in Wolde Lübke. Adaptive particle splitting based on turbulence energy for fluid simulations on gpus. Master's thesis, Nara Institute of Science and Technology, 2013.

[23] Hubert Nguyen. *GPU gems 3*. Addison-Wesley, Upper Saddle River, NJ, 2008.

[24] Wladimir J. van der Laan, Simon Green, and Miguel Sainz. Screen space fluid rendering with curvature flow. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games - I3D '09*, pp. 91–68, feb 2009.

[25] Matthias Müller and Nuttapong Chentanez. Solid simulation with oriented particles. *ACM Transactions on Graphics*, Vol. 30, No. 4, pp. 1–10, jul 2011.

[26] Barbara Solenthaler and Markus Gross. Two-scale particle simulation. *ACM Transactions on Graphics*, Vol. 30, No. 4, pp. 1–8, jul 2011.

[27] Miles Macklin, Matthias Müller, and Nuttapong Chentanez. XPBD: Position-Based Simulation of Compliant Constrained Dynamics. In *Proceedings of the 9th International Conference on Motion in Games - MIG '16*, pp. 49–54, 2016.