Snow Simulation

Zev Battad MeiXing Dong
Rensselaer Polytechnic Institute

1 Introduction

Though there has been much research into the accumulation of snow and the rendering of snow, the animation of snow and its unique characteristics has not been completely addressed. One of the difficulties of modeling snow is the fact that snow has characteristics from both fluids and solids. Consequently, neither fluid nor solid models are ideal for snow without extensions. The method we present is based on a grid based fluid model. However, we extend the model such that the particles' characteristics, such as volume and velocity, influence the grid cells. This hybrid scheme allows us to recreate both solid- and fluid-like behavior.

2 Related Work

2.1 Fluid Simulation

Early work on fluid simulation focused only on the appearance of the surface. The surface was usually a parametric function that could be animated over time to simulate waves. After this, O'Brien and Hodgins [3] combined a particle system with a height field to simulate splashing fluids. However, most of the methods did not incorporate interactions between the fluid and solid bodies and moving objects. Additionally, they did not have the full realism that would be afforded by a physics-based solution.

Foster and Metaxas [1] improved a method by Harlow and Welch [2] such that the modeled fluid could interact with static objects. They provided a solution to the Navier-Stokes equations that describe the behavior of a body of fluid. This allowed their model to be much more realistic as compared to a height field model.

2.2 Snow Simulation

Most of the prior research done on snow has been focused on the accumulation of and the rendering of snow. These methods can create scenes that have accurate snow coverage and accumulation. However, there is little consideration for the animation of snow. These methods have been extended to handle animation and interactions with objects; however, the models often had to be simplified to accommodate the extensions. For instance, a popular technique is to use height fields [5]. Recently, Stomakhin et al. investigated the modeling of realistic snow dynamics using a Material Point Method [4]. This method captured both the solid characteristics and the liquid characteristics of snow.

3 Method Outline

3.1 Liquid Representation

Our framework is based loosely based on the material point method. The fluid is represented partially as a grid of cells, each of which has a velocity, force (that acts upon it), and mass, and partially as particles, each of which has a position x_p, velocity v_p, mass m_p, volume V_p, and density ρ_p .

To initialize the system, the particles are given starting velocities and mass values. The degree of incompressibility is set by specifying the critical density of a cell. If the density of a given cell is at or past this value, then the velocities and forces of neighboring cells are adjusted accordingly to avoid allowing the entrance of more particles into the cell. This is detailed in step (4) of the update procedure below.

3.2 Update Procedure

For each timestep, the following update procedure is executed.

1. **Rasterize particle data to the grid.** This step is exactly the first step of the procedure by Stomakhin et al. [4]. The mass is transferred from the particles to the grid using weighting functions to interpolate values from the scattered particles, as follows:
$$m_i^n = \sum_p m_p w_{ip}^n$$
where $m_i^n$is the mass of grid cell i at timestep n, and w_{ip}^n is the weight given by the interpolating function given grid cell i and particle p.
Velocity is transferred using normalized weights.
$$v_i^n = \sum_p v_p^n m_p w_{ip}^n / m_i^n$$
where v_i^n is the velocity of grid cell i at timestep n, v_p^n is the velocity of particle p at timestep n. In general, a superscript will denote the timestep, and subscripts denote whether the term refers to a particle (p), a cell (i), or both (ip).
Both summations are over all of the particles, denoted by the subscript p.
2. **Compute particle volumes and densities.** This step is also taken directly from the procedure by Stomakhin et al. [4], and is only performed during the *first timestep.* A cell's density is $\rho_i^0 = m_i^n / h^3$, where h is the grid spacing. The spacing is assumed to be the same in all three spatial dimensions. The cell densities can then be weighted back to particles as $\rho_p^0 = \sum_i m_i w_{ip}^0 / h^3$. Particle volumes are given as $V_p^0 = m_p / \rho_p^0$.
3. **Compute grid forces.** The only force present in the system is that of gravity, so the force acting on a grid cell is $F_i^n = -9.81 m_i^n$.
4. **Equalize grid forces and velocities.** This step is how the degree of compressibility is set and enforced. The force and velocity of each grid cell is adjusted according to the densities of the neighboring cells. This is detailed in a following section.
5. **Update grid velocities.** The grid velocities were updated using an explicit integration method: $v_i^{n+1} = v_i^n + \Delta t m_i^{-1} f_i^n$.
6. **Update particle velocities.** The new particle velocities are $v_p^{n+1} = (1-\alpha) v_{PIC_p}^{n+1} + \alpha v_{FLIP_p}^{n+1}$, where $v_{PIC_p}^{n+1} = \sum_i v_i^{n+1} w_{ip}^n$ and $v_{FLIP_p}^{n+1} = v_p^n + \sum_i (v_i^{n+1} - v_i^n) w_{ip}^n$. This is also directly from the procedure by Stomakhin et al. [4]. They used $\alpha = 0.95$, and we did as well.
7. **Particle-based body collisions.** The only explicit collision accounted for is of that with the ground, or the lower boundary of the simulation bounding box. Any particle whose y coordinate is at or below the ground has its velocity set to zero in the downward direction (negative y). The other two velocity components are left unchanged.

Implicit collisions, such as that of particles being crammed into a single cell, are handled through the equalizing of grid forces and velocities in step (4).

8. **Update particle positions.** Each updated particle position is $x_p^{n+1} = x_p^n + \Delta t v_p^{n+1}$.

3.3 Equalize Grid Forces and Velocities

The equalization method is the following:

For every cell, horizontal and vertical components of the cell's force are examined. If the current cell has a horizontal or vertical component, we check the direction and the cell which the component points to (e.g., positive y velocity will go into the top cell, positive x velocity will go into the cell to the right). If the cell that the component of force or velocity points to has a density above the critical density threshold, we examine the fully packed cell's component of force or velocity.

If the force or velocity in the packed cell is in the same direction as the current cell, we then check whether it exceeds or matches the force or velocity of the current cell. If it does, then we would be pushing mass into the packed cell at a rate that is equal to or smaller than that of mass leaving the packed cell. In this case, there is no need to adjust the force or velocity of the current cell. However, if the packed cell's force or velocity is less than the force or velocity of the current cell, then we would be pushing mass into the packed cell at a greater rate than the rate of mass leaving that cell. Thus, we reduce the current cell's component of force or velocity in the direction of the packed cell such that it matches the component of the force or velocity of the packed cell in the same direction.

3.4 Compressibility

Fluids are often modeled as incompressible, and this is sufficient for most purposes. Snow, however, can often be compressed. We aimed to model this behavior. As detailed in step 4 of the update procedure, additional particles are disallowed from entering a cell at critical density. However, if the critical density has not been reached yet, then particles can enter the cell freely. To test this behavior, we simulated a loosely packed snowball being dropped onto the ground (Figure 1). Another scene we tested was that of a snowball being thrown at a standing tower of snow (Figure 2). In yet another scene, a snow tower is allowed to compress with only the influence of gravity (Figure 3).

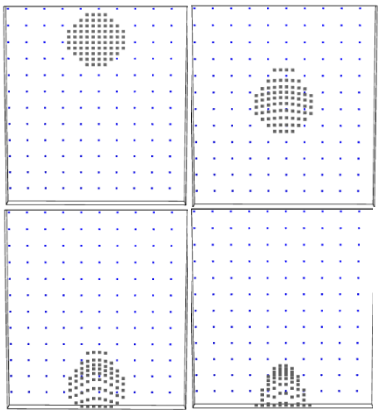

Figure 1: Falling snowball. A snowball is allowed to fall. When it reaches the ground, it compresses and deforms into a pile of snow.

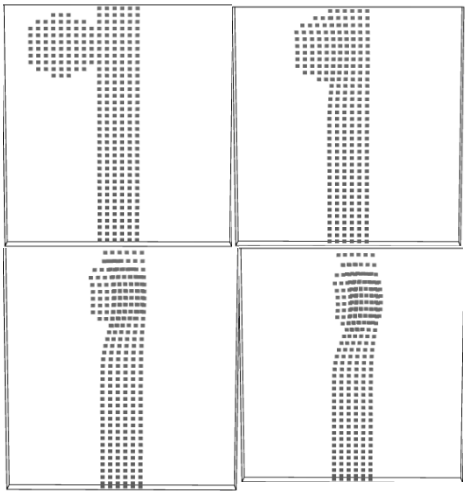

Figure 2: Snowball and Tower. A standing tower of snow is absorbing the impact of a snowball being thrown at it, demonstrating the compressibility of the material. Each frame is separated by a time interval of 0.02.

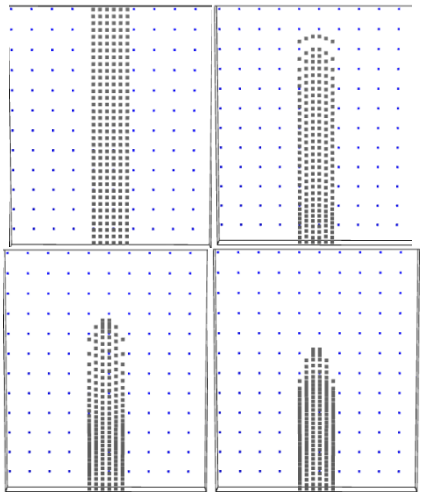

Figure 3: Tower of Snow. The tower of snow is initially loosely packed, then becomes packed as gravity acts upon the snow particles. Each frame is separated by a time interval of 0.2.

In Figure 1, the snowball deforms when it reaches the ground and ends as a pile of packed snow. In Figure 2, the standing tower of snow absorbs the snowball and becomes much more packed in the area of collision. Figure 3 shows a loosely packed (not at critical density) tower of snow collapsing and compacting due to gravity.

3.5 Cohesion

Another characteristic of snow that we attempted to capture is its ability to stick to itself, as well as other objects and surfaces. To demonstrate the former case, we tested our system on a scene where a snowball is initially in contact with a tower of snow, with no initial velocity, and gravity is the only acting force (Figure 5). The latter case is demonstrated by part of a snowball and tower simulation where the snow had been pushed to the right bounding wall and appears to experience friction as it is pulled by gravity (Figure 4).

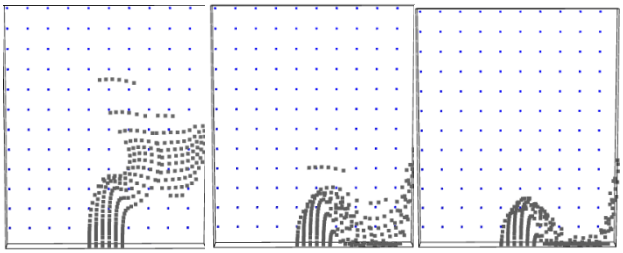

Figure 2: Collapsed snow tower. The snow tower has been knocked over by a snow ball. The snow "sticks" to the wall where they collide. Each frame is separated by a time interval of 0.1.

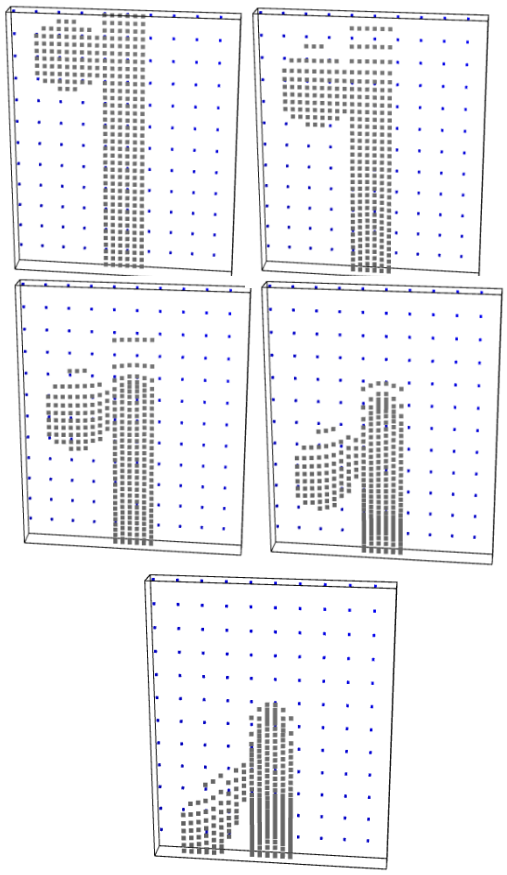

Figure 5: Snowball and tower (No initial velocities). The snowball starts in contact with the tower. As both compress and fall to the ground, the contact border is maintained, showing that our snow will stick to itself. Each frame is separated by a time interval of 0.1s.

4 Challenges

The largest stumbling block we faced was our lack of knowledge in the realm of materials science and deformation. The procedure developed by Stomakhin et. al. [4] depends on the use of Cauchy stresses and deformation gradients. Though we could glean the general logic behind their mathematics, our understanding was not sufficient for a complete implementation.

We chose instead to focus on specific physical characteristics that their method was trying to capture, such as snow compressibility and cohesion. Starting with a test scene and an idea of what the behavior should be, we tuned our system to yield results similar to what we desired.

The first scene we worked with using this method was the falling snowball scene. A compacted pile of snow on the ground was the desired end result, and so we incorporated collision with the ground plane to halt the downward motion of the particles. However, all of the particles would simply stop only when they hit the ground, rather than stopping in the proximity of other snow particles, so we implemented adjustments to the velocities and forces depending on the fullness of a given cell.

5 Results and Discussion

Results and Discussion

Critical density is decided by a multiplier against the density of a single particle. We found a critical density multiplier of x4.0 to allow for reasonable compression of snow geometry relative to its initial state. Below x4.0, it was found that the snow's geometry remained more rigid relative to its initial state and did not allow us to view its compressed shape as it relaxed. Above x4.0, it was found that the snow's geometry compressed too greatly; many of the particles accrued at the bottom of the scene or in the bottom couple of cells and the snow's relaxed geometry was flattened (Figure 6).

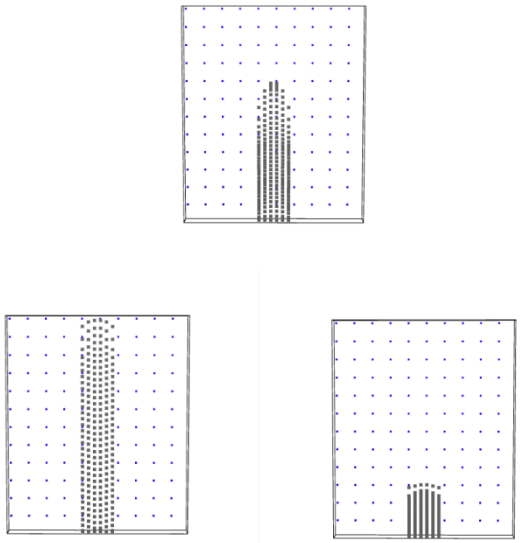

Figure 6: Snow tower with varying critical densities. The topmost image shows the snow tower with a critical density multiplier of x4.0. The bottom left shows the rigidity of the structure with a critical density of x2.0. The bottom right shows the flattened geometry of the structure with a critical density of x8.0. Each frame is taken 0.5s into the simulation.

Particle generation density decided how many particles could be initially generated per cell of a 3-dimensional grid. Less may be generated in individual cells depending on shape of the simulation's initial configuration. Particle generation densities of 8, sparse, and 27, dense, were considered. It was found that dense particle generation allowed for more detailed simulations, as there were more representative particles per volume of snow (Figure 7).

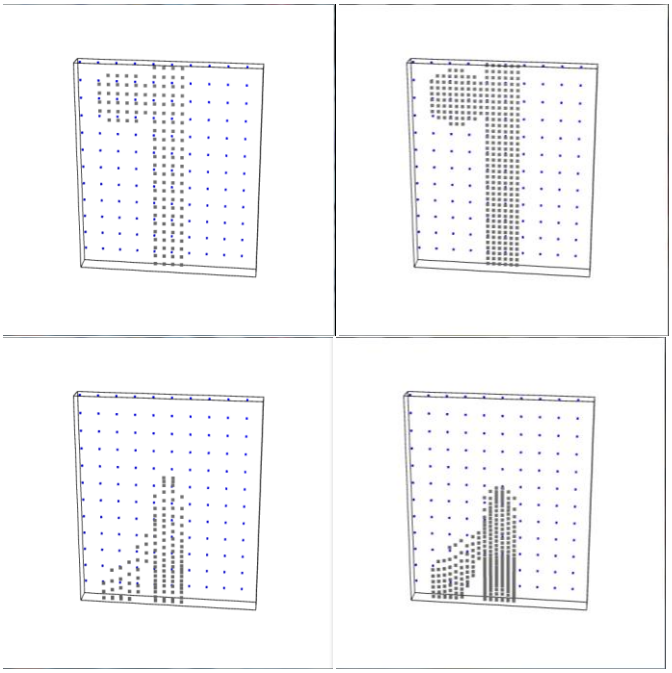

Figure 7: Sparse and Dense Snowball Tower The lefthand images are the snowball and tower scene with sparse initial particle generation. The righthand are with dense generation, showing more of the snow's geometry's detail.

As our system uses explicit integration for velocity updates, it is susceptible to errors due to timestep size. A timestep of 0.001 seconds was used. It was found that timesteps greater than this exaggerated the error in explicit integration, leading to inaccurately updated particle velocities. (Figure 8).

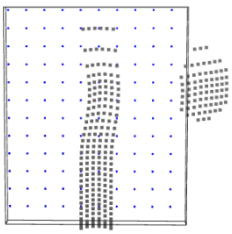

Figure 8: Example of error due to high timestep. The snowball started at the left of the tower and was tossed to the right. Due to the high timestep of 0.01s, the snowball's particle's horizontal velocities were not lowered sufficiently and the snowball passed through the tower.

Computation time was dependant mainly on the number of cells and number of particles used. Complexity of particle configurations and complexity of particle interactions did not affect the computation time as much as particle count did. (Table 1).

Between the two of us, we estimate that about 70 hours were spent on this project. We started with the fluid code from homework two.

The work done by MeiXing is as follows:

- Modification of the particle class such that the particles store information about position, velocity, volume, and density.
- Modification of the fluid grid such that each cell holds a mass, velocity, acting force, and position.
- Implemented the rasterization of particle data to the Eulerian grid.
- Incorporated collision with the ground, or lower boundary of the bounding box in the simulations.
- Implemented explicit time integration for updating the cell velocities.

The work done by Zev is as follows:

- Incorporated Cauchy stress term and handled the updating of particle deformation gradients. (This was not used in the final system.)
- Created all test scenes.
- Ran tests to investigate changes in behavior due to different values for the critical density, timestep, initial particle generation density, and other parameters.
- Collected screenshots from tests for use in report and presentation.

6 Limitations and Future Work

Though our model generates examples that follow the behavior we desired, such as having compressibility and cohesion, there is still much work to be done. Our simulations do not reach stable steady states. Many of the particles retain small velocities that do not disappear, and therefore can force themselves into packed grid cells. The particles are eventually all packed into the cells bordering the ground when the simulation is allowed to run to completion (Figure 6).

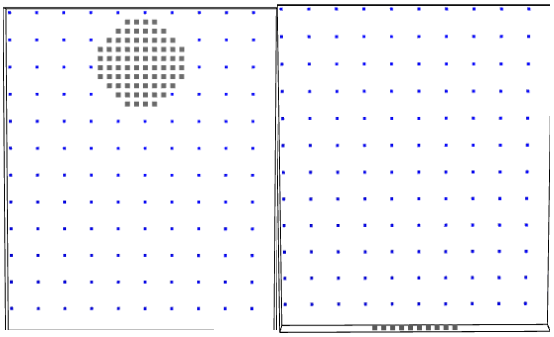

Figure 6: Snowball steady state. The snowball has completely compressed into the area right above the ground plane, an undesirable behavior.

Another problem is that our system does not exhibit realistic physical behavior. When two snowballs of the same size but different density is dropped from the same height, they do not fall at the same rate. Another issue is that particles are allowed to pass through each other, as can be seen in Figure … (snowball thrown at tower), when the snowball's particles end up completely between or coinciding with the snow tower's particles. If this were more realistic, the particles of the snowball and the tower should remain largely separate except for the border where they are colliding with each other.

We also do not account for the spread of material or material explicitly pushing on each other. This is apparent in Figure … (snowball drop), where one would expect the ball of snow to expand in the two horizontal directions as it flattened and compressed, rather than staying in exactly the same x and z coordinates it started out with when the ball was dropped.

There are two possible extensions that are the most straightforward. Our system only deals with two dimensions currently, and we use explicit integration. Modifications can be made such that all three dimensions are considered and a different integration technique, such as implicit integration, is used instead.

References

[1] Foster, N., & Metaxas, D. (1996). Realistic Animation of Liquids. *Graphical Models and Image Processing 58*, 471-483.

[2] Harlow, F., & Welch, J. (1965). Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with a Free Surface. *The Physics of Fluids 8*, 2182-2189.

[3] O'Brien, J., & Hodgins, J. (1995). Dynamic Simulation of Splashing Fluids. *Computer Animation 95*, 198-205.

[4] Stomakhin, A., Schroeder, C., Chai, L., Teran, J., & Selle, A. (2013). A material point method for snow simulation. *ACM Transactions on Graphics.*

[5] Sumner, R., O'Brien, J., & Hodgins, J. (1999). Animating sand, mud, and snow. *Computer Graphics Forum, 18*, 17-26.

Appendix A

Table 1 – Rendering times for multiple test scenes. Note: For all scenes, timestep = 0.001s, particle generation density = 27, critical density multiplier = x4.

Scene	Snowball Drop				Snow Column				Snow Column with Ball			
Simulation Time (seconds)	0.1	0.5	1.0	5.0	0.1	0.5	1.0	5.0	0.1	0.5	1.0	5.0
Real Time (seconds)	4.36	22.9	45.3	225	10.3	49.8	101	514	13.1	65.9	131	679
Number of Particles	76				216				290			
Grid Size	10x12x1				10x12x1				10x12x1			
Cell Size	0.01x0.01x0.01				0.01x0.01x0.01				0.01x0.01x0.01			

Scene	Snow Column with Thrown Ball				Snow Column with Thrown Half Ball				Colliding Snow Balls			
Simulation Time (seconds)	0.1	0.5	1.0	5.0	0.1	0.5	1.0	5.0	0.1	0.5	1.0	5.0
Real Time (seconds)	13.0	65.0	138	667	13.0	67.8	131	672	7.35	38.3	73.7	374
Number of Particles	290				290				148			
Grid Size	10x12x1				10x12x1				10x12x1			
Cell Size	0.01x0.01x0.01				0.01x0.01x0.01				0.01x0.01x0.01			

Computer Specifications: Intel Core i3 2100
Cores: 2 Threads: 4
Memory: DDR3 6144 Mbytes
Graphics: Radeon HD 5670