A Simple Material Point Method

Paul Makl
University of Puget Sound
1500 North Warner
Tacoma, Washington
paulnmakl@gmail.com

Etan Bard
University of Puget Sound
1500 North Warner
Tacoma, Washington
ebard@pugetsound.edu

ABSTRACT
This paper outlines a modular framework created to utilize the Material Point Method to simulate particle-level physical interactions between granular solids. As seen, we implemented this framework with the desire to simulate various physical properties of snow, and more importantly, create a robust system in which intricate interactions between particles can be done.

Categories and Subject Descriptors
H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms
Theory

Keywords
ACM proceedings, LaTeX, text tagging

1. INTRODUCTION
Snow provides a unique challenge to the field of computer graphics because of its unique physical properties. Due to these properties, simulating realistic snow is currently an open area of research. Current simulations exist for solid and liquid dynamics, however, granular solids such as snow and sand are a more complex undertaking. Snow is further complicated due to the fact that it can be compressed, and can act more like a fluid or a solid based on its temperature. The vast majority of graphics research does not provide a unique way of dealing with complex snow dynamics, and instead is often modeled using either solid or liquid simulators.

Recently, a group of researchers at UCLA have achieved visually compelling results by combining a Material Point Method [MPM] and a elasto-plastic constitutive model to create realistic looking snow. This model was used to create the stunning snow in the Disney movie Frozen.

Our goal was to unpackage exactly what this method encompassed, and create our own system that would be simple to run, easy to configure, and provide somewhat believable dynamics. While our method was heavily influenced by the work of [Stomakhin], we took a unique approach that we believe can be used in conjunction with more sophisticated approaches to create visually stunning effects while using less processing power. Our final approach resulted in a framework that achieved these desired results. Written in C++ and using the OpenGL, FreeGLUT and GLU libraries for visualizations, our chosen toolsets allowed for low level work to simulate thousands of particles.

2. BACKGROUND AND RELATED WORK
Researchers have found it hard to simulate the dynamics of snow due to the highly variable characteristics of the snow depending on a multitude of factors. Sometimes, snow behaves more like a solid, and other times it behaves more akin to a liquid. Dry powdery snow, which is commonly good for skiing on, can be compared more to a liquid than a solid; the snow does not stick together well. As the temperature rises and the snow becomes more compacted, the snow behaves much more like a solid, fracturing when thrown against a wall in a ball. This is further complicated by the many types of snow in between, which all depend on the volume of compactness, humidity and numerous other variables. To create a framework that can model all types of snow, including but not limited to fine powder, compacted snow and falling snow, would require all of these variables to be accounted for, and a lot of number crunching in the background to get realistic results. Different approaches have been taken to simulate some of these dynamics.

2.1 Accumulation Work
Some work has been done on the accumulation of snow, where other environmental objects remain static [Feldman]. Flows are discretized and represented as an Eulerian grid. Cells that contain surfaces in them are marked so that snow cannot go through them. Forces associated with the cells in the Eulerian grid are set at the start of the simulation based on the desired wind velocities. Snow accumulates based on the angle of the surface, and the amount of snow in neighboring cells. To compute the flow fields for the snow, Naiver-Stokes equations for incompressible fluids are used. Since snow is compressible, this creates a problem for creating realistic snow effects.

2.2 A Material Point Method for Snow Simulation

The material point method is used traditionally for modeling solids. However, the method can be modified to work with granular solids and specifically snow, as was done in the research paper [Stomahkin]. The method can be broken down into various steps; the first step involves rasterizing every particleâĂŹs velocity to the grid. This is used to compute grid based forces for all other steps. Forces are then applied to different cross sections of the grid to update grid velocities. This could include wind, current, gravity or other forces. Then deformation gradients, velocities and volumes are updated, all dependent on the collisions that occurred. Following these changes, the positions of particles are finally updated, and the process begins again. The paper utilizes an elasto-plastic constitutive model combined with a hybrid eulerian-lagrangian system. This approach removes the need for Lagrangian mesh connectivity.

Our project is based on a simplified version of this approach. While the most impressive results we found in our research came from the Stomakhin paper, we believe their model is more focused on creating an aesthetically pleasing result than creating a physically accurate representation of snow. The model itself âĂIJignores root causes, and we instead concentrate on deriving an empirical model based on phenomenological observationsâĂİ [Stomakhin 3]. While this approach is appropriate for creating visual effects in movies, it is not sufficient in the realm of physical interpretation.

3. IMPLEMENTATION / METHOD

The goal of our project was to create a robust framework that can provide the tools and layout necessary for simulating physical properties of granular solids on the particle level, with the desire being to represent select physical properties of snow with some degree of accuracy. To accomplish this, we set out to use C++ for lower level control of the simulation. C++ also better couples with the OpenGL library than other higher level languages, the library we used to display the simulation. FreeGLUT was also used along with GLU to allow for the perspective view of the camera, create a window in a cross-platform environment, and also allow for keyboard input during playback. Although the C++ environment proved to be challenging to set up, the end result was a cross-platform project capable of being run and edited in both the Mac and Windows environments, albeit using XCode on Mac and Eclipse on Windows.

3.1 Overview of the Framework

Although our framework is highly customizable, it follows a general, overarching architecture. At the highest level is a module that switches between two other modules. One that draws components to the screen, and another that calculates new physical properties. Both of these modules together have access to a shared data structure, in our case we called it the environment. The environment stores information about everything that the physics module needs to operate on, and everything that the drawing module needs to draw.

3.2 List of Major Components and Purposes

Our implementation is loosely based on the work in the Stomahkin paper. By using a modular design capable of relatively easily switching in and out components, even the way certain snow dynamics are handled, can be switched on and off easily inside of the code. This modular design also allows for expandability, and upgradeability to the existing framework.

3.2.1 Switch Module

Starting from the top, Switch is what allows for the modular design, and it is based on the main loop of execution required for GLUT to function. Our framework exists within this main loop, with various components called repeatedly through each iteration of the loop. This loop of execution allows components to be called in a certain order once per frame. Given the name, the main function of the loop is to âĂŸswitchâĂŹ between the Draw module and Physics module. If desired, the switch loop can also write particle information to a file for playback purposes. Due to the nature of the loop, much of the code at this level is easily understood and manageable; functions can be disabled and enabled as necessary with expected results.

3.2.2 Environment Module

The environment is a collection of multiple data structures that keep information for every particle in our system. The environment also includes functionality to maintain track of hundreds to millions of these particles, such as sorting algorithms and data structures to contain conglomerations of particles.

3.2.3 Particles

The smallest unit in our project is the actual particle, and each particle has multiple variables with the potential for expansion. All particles are contained in a list, accessible to functions in the environment, and externally from both the Draw and Physics Modules.

3.2.4 Nodes

Nodes hold forces that could represent wind currents, gravity, or any other outside forces that could affect the particles. Nodes in our system are utilized heavily by the Physics Module for the Eulerian grid as will be seen.

3.2.5 Zoning

Particles are âĂŸzonedâĂŹ based on their location. These zones are the same size as the Eulerian grid (see below for details). Each zone is, thus, a fixed distance along the x, y, and z-axes. For our simulations, this size is fixed at 1 unit for a cube shaped zone, called a box. Each box has an id number which refers to it. Sorting particles into the correct box is as simple as rounding their locations down to the nearest box. Zoning serves as critical to providing efficient collision detection, but can provide rapid subsetting of particles based on their location for a variety of external functions to utilize.

3.2.6 Sorting

Particles are sorted through the sort manager. The sort manager sorts all the particles first on their boxID x-value, then the y-value, then the z-value. The result is a 3-variable

sort. This is done every frame through a variation of shaker sort.

3.2.7 Physics Module

The physics module is a collection of multiple functions which modify the state of particles and nodes in the environment. Each function iterates over the relevant data in the environment to perform its desired function. An example of a physics function is collision detection, which utilizes the environmentâĂŹs automatic particle zoning to determine which subset of particles should be used to check for inter-collisions.

3.2.8 Eulerian Grid

The eulerian grid utilizes Nodes from the Environment Module. In the current framework, velocities are stored on nodes. The concept behind this grid is to promote fluid-like movement of particles. This is accomplished over various steps that are called through method calls to the Physics Module from Switch, and are outlined in the initial stages of the Material Point Method. The first stage interpolates velocities of particles in zones that border each node. The resulting forces stored in each node are then extrapolated back on to the particles to determine their new locations. By doing this, particles can influence the movement of other nearby particles, even without contact. This produces fluid âĂŸvortex swirlsâĂŹ and other traits characteristic of liquids, and also falling snow.

3.3 Lagrangian Particle Descriptions

Each particle stores information about itself, such as itâĂŹs current velocity and position. To add to this, we also store a unique description for each particle which differentiates our model from other implementations. Each particle has a field called mass which is a floating point number between zero and one. When the mass is closer to zero, the particles velocity between time steps is effected entirely by the Eulerian grid. When the mass is 1, the effects of the Eulerian grid are minimal, and only its stored velocity and collisions with other particles will changes its position. By manipulating this variable to any value between 0 and 1, we can modify how the particle will behave. This allows for the representation of multiple types of snow within the same simulation.

Particles also contain a floating point variable, volume, which impacts the range at which collisions are possible, as well as the compatibility of the snow. The higher the volume, the more the snow can be compressed. Coupling this with mass allows for snow to accumulate after falling, with the densest snow closer to the ground.

3.4 Collision Detection

To accomplish more complex particle-to-particle interactions, collisions are required. Our method detects collisions with help from environmental zoning. For each particle, our collision detection algorithm will check if each particle will collide with particles contained within the particleâĂŹs own zone, and a neighboring zone. The latter will only occur if an updated position exists outside of the particleâĂŹs current zone. Sorting allows for this rapid zoning.

3.5 Draw Module

The draw module utilizes OpenGL to draw objects at the position of the particles, resulting in a visual of the interactions happening between frames as the simulation progresses. In the current simulation, particles are represented as low-polygon spheres. However, the programmer is not restricted to this representation of a particle, and can change the object to draw particles as any mesh or object, or a point for increased performance, in the draw module.

3.6 Playback

Playback is a separate project which allows for simulation files generated by the main project to be played back in real-time speeds. During playback, control of the camera is also possible, allowing for a free-flying camera through the simulation, as well as frame controls to go back, forward, or restart. This project is important for simulations containing thousands to millions of particles, as in these simulations each frame can take several minutes to do all calculations. Playback allows simulations with hundreds of thousands of particles to be played back at a real-time speed.

A simple approach to storing particle data for playback was taken. Instead of recording the video of the scene, Individual particle positions were recorded and stored in a file. This file contains the position of every particle, at every timestep of the simulation. These can reach gigabytes for larger simulations.

4. EVALUATION / RESULTS AND ANALYSIS

Our system is hard to analyze because for three reasons. First, it does not offer the same functionality as other models, it is not âĂIJfeature completeâĂİ, and we were unable to compare our system to other implementations. Many of these systems required computing power beyond our grasp, and others were proprietary systems developed for private companies. Thus, the only comparisons we can make are on paper.

4.1 Overal Performance

In the end, our modular framework drastically simplified the basic stages of the Material Point Method, and provided an intuitive design through which simulations could be manipulated and performed. Although the performance of our computers was stretched to the limits attempting many of the more complex simulations, these simulations were in fact still conductible.

4.2 Development Environment

The choice to code in C++ proved to be a wise decision. Pointers were taken for granted, as much of the modular design required passing around thousands of references to particles. OpenGL is also tightly coupled to C/C++, so we could code without worrying about binding functions to access OpenGL as is required in other higher-level languages.

4.3 Zoning

The code to establish correct zones for every particle every frame is a linear operation based on the number of particles, and as will be seen in the improvement to collision detection times, is a good investment.

4.4 Sorting

Though technically the sorting algorithm we used is a quadratic sort, it is much faster on already sorted data. This is beneficial to the nature of particles, as they do not change locations quickly, and take several frames to travel through a box. Building on this further, the variables being sorted upon are not continuous, with a discrete number of intervals possible. This improves the initial shaker sort, but also secondary sorts which are modified selection sorts done on the y and z-axes. The end result is a comprehensible sorting algorithm that is also efficient.

4.5 Collision Detection

Detecting collisions between particles needed to be both accurate and fast as they are the slowest part of our framework. To accomplish this, our detection algorithm is aided by the environmentâĂŹs sorting and zoning functions. By sorting particles into zones based on their location in the grid, determining which zone a particle is in, as well as finding subsets of neighboring particles is rapid process. This has some interesting implications for performance of our system. First and foremost, this drastically improved performance over brute forced collision detection between all particles in our system. It also means that as you decrease the size of each zone, fewer particles will be able to be encapsulated in each zone, increasing the speed of collision detection. In traditional zone based approaches to collision detection, as you decrease the size of each zone, the overall accuracy of the algorithm drops. Our algorithm, on the other hand, considers where the particle is going. This decreases the amount of accuracy lost by creating smaller collision zones.

4.6 drawing

Drawing proved to be a test of the OpenGL framework for rendering polygons more than an exercise on improving code. Each particle is represented with about 25 vertices, but our playback simulation handled over 100,000 particles with ease.

4.7 Discussion

The only comparable system to ours is the one described in the Stomahkin paper. However, comparing our two models is hard to do. First, we have no way of running the systems side by side to compare their performance. Second, the model in Stomahkin paper handles more of the physical properties of snow than ours does. Most notably, this model handle the fracture of snowballs. This was because we were not able to implement the elastoplastic model, that was the primary method for handling fracture. Given that we did not handle these physical properties, our system would certainly render scenes faster than the one described in the paper. Our system is somewhat comparable to other systems that modeled snow accumulation [Feldman].

Our system was more comparable to the Feldman paper in some respects. Both of our systems relied heavily on an Eulerian grid, however ours took some aspects from the Stomahkin paper, such as the lagrangian components of storing particle velocities.

5. FUTURE WORK

Our framework have proven to provide the robustness quintessential to modeling particle interactions, and looks promising for simulating snow to a more accurate degree. The direction this project should continue is to look towards increasingly more accurate particle dynamics. The design we have created is easily and readily expandable to perform a variety of simulations. This could be better streamlined with the creation of a project to create scene design files to read into a simulation builder, similar to how the simulation builds video text files for playback. This would allow for more complex scenes to be modeled.

The Eulerian grid could be used to interpolate and extrapolate more than just velocities of particles. An example of this could be temperature transfer through the grid, which could somewhat simulate radiant and convective heat transfer. This also ties in to creating a more realistic and accurate simulation.

Displaying the simulations with OpenGL could also be improved. Utilizing shaders and casting shadows would make scenes appear much more realistic.

Another area for future development is in finding a way to reduce the size of the files produced by the simulation for playback purposes. There are a few potential solutions, the first of which is to store the particles in some sort of binary form instead of text files. Another option is removing many of the elements of the file that make it somewhat human readable. Each of the particle positions is separated by a comma, and each particle position is on a new line. These commas add up for simulations that last many times steps with hundreds of thousands of particles.

6. CONCLUSION

As expected, snow dynamics remain a challenge to simulate using traditional models designed to model fluids and solids. By combining the Eulerian grid with Lagrangian particle interactions we were capable of modeling some physical properties, though not to a visually impressive extent. Although our visual results and actual physical dynamics of snow were not exactly on the same calibre as the results of the Stomahkin paper, we have created a highly modular framework for simulating low-level particle interactions, and took a more physics based approach. The end result was a framework better catered to the potential for expansion. Should this project be berthed with a more robust series of physical equations, the potential for improved results is likely.

7. ACKNOWLEDGEMENTS

We would like to thank many of the professors at the University of Puget Sound for helping us finish our research. Carl Toews helped us decipher some of the more complex math that we couldnâĂŹt understand, and meticulously listened to our early concepts. We would also like to thank Phil Howard for helping us comprehend some of the finer points of the material point method, and with managing a large scale physics simulation project. His advice on realistic goals was also quite helpful and provided direction for the project.

8. SOURCES