# Simulating Material Damage

Viktor Tholén, vikth187@student.liu.se

December 2020

## Abstract

This report explains how a simple material damage simulation was created using Unity. The method is based on a simplified version of the Material Point Method and produced realistic results with room for improvements and extensions.

## 1 Introduction

Simulating the damage of materials is a difficult process to model and is an important topic in the visual effects industry. The way materials structure is defined varies between different methods. Solving fluid dynamics was originally developed as the Particle in Cell(PIC) method in 1957. This method could not solve problems in solid dynamics. Hence, another method was developed called the Material Point Method(MPM). Since 1993, this method has developed further and is today a popular addition to simulation applications.

## 2 Background

Traditional MPM is thoroughly explained in the paper *The Material Point Method for Simulating Continuum Materials* by Jiang et al [1]. This method was inherited from the previously developed Langrangian Finite Element Method (FEM) which was widely used for elasto-plastic solids. Although FEM could solve mesh distortion, fracture and self-collisions, it was very computationally expensive. Instead, MPM uses a hybrid Eulerian/Lagrangian method that uses a Eulerian grid with Langrangian particles. In this case, Eulerian means that the observer follows an individual particle and follows it through the simulation. On the contrary, Langrangian refers to the observation of a point in space where particles are passing by. This differs the particles from the grid in the way they are observed. MPM can simulate a variety of materials such as elastic objects, honey, snow, sand and other viscoelastic fluids. The method has found success within the simulation industry and has been integrated into the production framwork of Walt Disney Animation Studios, where it was featured in movies like Frozen, Big Hero 6 and Zootopia.

Hu et al. introduced 2018 an extension of MPM in *A Moving Least Squares Material Point Method with Displacement Discontinuity and Two-Way Rigid Body Coupling* [2]. The Moving Least Squares(MLS) MPM simplifies key equations which improves computation time significantly. It also enables additional phenomena that are not supported by traditional MPM, such as cutting and open-boundaries. The primary difference is the replacement of the weight gradient $\Delta w_{ip}$ which affects most calculations, but there are other differences too. However, the stress, interpolation and mass/momentum transfer is the same for both methods. Note that the MLS-MPM is identical to MPM in many ways and only modifies small parts of certain equations.

This report is based on the MLS-MPM approach and will formulate equations accordingly. as there seemingly is no disadvantage with the MLS-MPM compared to the MPM version.

## 2.1 Stress

The stress is related to the internal forces that is created from deformation. There are several different stress definitions, one of them is Cauchy stress $\sigma$. The Cauchy stress is related to the Piola-Kirchoff stress with:

$$\sigma = \frac{1}{J} P F^T, \tag{1}$$

where J is the determinant of the deformation gradient $F$ which characterizes infinitesimal volume change. $P$ is related to the strain energy function $\Psi(F)$ and gives the following relationship with the commonly used Neo-Hookean energy density model:

$$P = \mu(F - F^{-T}) + \lambda log(J) F^{-T}, \tag{2}$$

Here, $\mu$ and $\lambda$ are called the Lamé parameters and controls the material elasticity, these can be tuned empirically to achieve different materials.

## 2.2 Interpolation

Transferring quantities between the grid and the particles requires an interpolation method. The idea is that particles that are further away from a grid node contribute less than closer particles. For a finite element approach, looping through every node for all particles is unnecessary. Hence, only grid nodes within a 3x3x3 proximity will be taken into account when interpolating weights. Note that cubic interpolation requires 5x5x5 proximity which is more computationally expensive.

The most common interpolation kernels are the cubic and quadratic B-splines in Eq 3 and 4 respectively. Note that cubic interpolation requires 5x5x5 proximity which is more expensive, but provides wider coverage compared to the faster quadratic kernel.

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

$$N(x) = \begin{cases} \frac{3}{4} - |x|^2 & 0 \le |x| < \frac{1}{2} \\ \frac{1}{2}(\frac{3}{2} - |x|)^3 & \frac{1}{2} \le |x| < \frac{3}{2} \\ 0 & \frac{3}{2} \le |x| \end{cases}, \quad (4)$$

Typically, one dimensional splines extended to multiple dimensions is used due to its simplicity and ease to use in computations. The one dimensional dyadic product is defined as:

$$N_i(x_p) = w_{ip} = N(x_p - x_i)N(y_p - y_i)N(z_p - z_i), \quad (5)$$

where $w_{ip}$ is the weight, $i = (i, j, k)$ is the grid index and $x_p = (x_p, y_p, z_p)$ is the particle's position.

## 3 Method

The method consists of three main steps that is done for every time step during the simulation. Before that, several components need to be initialized. Firstly, the index based grid needs to be created where every node has a mass $m_i$ and a velocity $v_i$. The particle system stores all the particles which has a position $x_p$, velocity $v_p$, affine velocity matrix $C$, volume $V$ and a mass $m_p$. After the initialization, a P2G step is done followed by calculating the initial volumes.

### 3.1 Particle to Grid

The governing equations are the conservation of mass and momentum. These equations determine the motion of the system and are supplemented with a constitutive model that relates strain to stress. From the governing equations, the mass and momentum equations are derived according to the Affine Particle in Cell (APIC) method with:

$$m_i = \sum_p w_{ip} m_p \quad (6)$$

$$m_i v_i = \sum_p w_{ip} m_p (v_p + C_p(x_i - x_p)) \quad (7)$$

Equation 6 is used to transfer the mass to the grid and equation 7 is used to transfer grid velocities despite still containing the mass term. In addition to the momentum, the grid velocities need to account for internal forces $f$. These forces can either be added directly to a force term and later applied in the Grid Update step, but can also be added to the grid velocity for convenience. The force equation is defined as:

$$f_i = -\sum_p V_p \sigma_p w_{ip} (x_i - x_p) \quad (8)$$

### 3.2 Grid Update

In this step, all grid nodes are looped through and updated. Since the grid velocity variable still contains mass, we divide the grid velocities by $m$. Here, any external forces are also added to the grid velocities, such as gravity. Lastly, the simulation domain's boundaries need to be enforced with conditions that simply sets border velocities to zero.

### 3.3 Grid to Particle

The Grid to Particle(G2P) aims to transfer the grid quantities back to the particles and this is done similarly to the P2G step. Here, the particle velocities are simply updated with:

$$v_p = \Sigma_i w_{ip} v_i \tag{9}$$

The next step is to update the particle's positions with their new velocities:

$$x_p^{n+1} = x_p^n + \Delta t v_p^{n+1} \tag{10}$$

Apart from the particle, some important matrices that are used in the P2G step needs to be updated to prepare for the next time step. The deformation gradient $F$ is a matrix that is stored on the particle. It represents how deformed a material is locally and needs to be updated in this step to preserve volume. The updated deformation gradient is defined as:

$$F_p^{n+1} = (I + \Delta t C_p^{n+1}) F_p^n \tag{11}$$

where $n$ is the current time step, $I$ is the identity matrix and $C_p$ is the affine matrix. $C_p$ is updated with:

$$C_p^n = (D_p^n)^{-1} \Sigma_i w_{ip}^n v^{n+1} (x_i - x_p^n)^T, \tag{12}$$

where $D_p = \frac{1}{4}\Delta x^2$ for cubic interpolation kernels and $D_p = \frac{1}{3}\Delta x^2$ for quadratic kernels.

### 3.4 Pinning

Keeping certain particles static in the simulation can create different results. This technique is called pinning and needs to be applied to the nodes instead of the particle. It works by increasing the density of nearby nodes to a high number. On top of that, velocities of pinned nodes are set to zero instead of updated in the P2G step.

## 4 Implementation

The method was first implemented as an add-on for Blender which uses Python as its scripting language. The slow nature of Python without support for parallel computations within Blender caused very long simulation times. As this caused major problems when debugging, the method was translated into the game engine Unity. Unity provides use of C# instead of Python which is significantly faster. Additionally, parallel CPU computations can be done with ease with the Unity Burst library. To visualize the result, thousands of particles has to be displayed on the screen. This quickly becomes a bottleneck when the simulation computational time can be done in real time. Thus, GPU instancing is used within Unity to reduce the number of draw calls. Neither of these methods are available to the Blender scripting environment without exploring the underlying higher level frameworks.

Programming in Unity can be done in different ways. Here, everything was done in a single script that was attached to an empty object in the world. The script referenced a sphere mesh was used for the instancing and the ball which can be seen in Figure 1. Unity uses a Start() function that is run once at the start of the game and a Update() function which runs every frame until the end. The initialization was therefore done in Start and the simulation was done in the Update function for each time step.

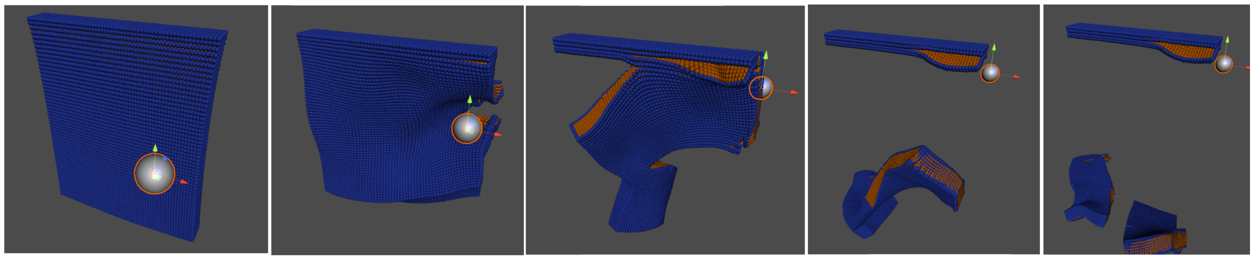

Figure 1: Simulation of a 64x64x8 block of particles that are colored orange on the inside and blue on the outer shell. The top layer of the block is pinned and a ball is driven through the material that acts as an external force to damage the material apart from the gravity.

## 5 Results

The final method was implemented in Unity where a simulation with approximately 32000 particles could be simulated at almost real time speed. The result can be seen in Figure 1 above.

## 6 Discussion

The MPM method can be applied in many different ways. The MLS-MPM can reduce computation time and still create plausible results. The method that is presented here simulates isotropic materials and consists of a fairly simple model. This can be extended to simulate snow by adding snow plasticity relations to the deformation gradient. Exploring the method in another direction would be to acknowledge anisotropic material properties. This was recently done as an extension of MPM by Wolper et al. [3] in *AnisoMPM: Animating Anisotropic Damage Mechanics*. In order for MPM to work for anisotropic materials it requires additional components. Such an MPM would require anisotropic damage state evolution, anisotropic elastic response and anisotropic mechanical degradation. Wolper et al. extends the MPM method by using the traditional quantities $m, x, v$, and $F$, but also introducing the damage $d$, fiber directions $a$, and fiber magnitudes $\alpha$ to use in the simulation.

If the project was done again, switching to Unity should have been done from the beginning as much time was spent in Blender without progress. The goal from the beginning was to achieve anisotropic properties, however, this was not a realistic goal to achieve before mastering the underlying method. Focusing more on the improved MLS-MPM instead was probably a wise choice considering the time frame of the project.

## 7 Conclusions

Simulating material damage is a rather complex process but can produce realistic results that can be used in both the VFX industry to produce a variety of material effects, but also the game industry, due to the MLS-MPM addition. This implementation is rather simple compared to what the Material Point Method is capable of. Thus, the implementation can be extended in a lot of ways although it achieves good results on its own.

## References

[1] Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. The material point method for simulating continuum materials. SIGGRAPH 2016 Course Notes Version 1 (May 2016), 2016.

[2] Yuanming Hu, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Andre Pradhana, and Chenfanfu Jiang. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. ACM Trans. Graph. 37, 4, Article 146 (August 2018), 14 pages. DOI: 10.1145/3197517.3201293, 2018.

[3] Joshuah Wolper, Yunuo Chen, Minchen Li, YuFang, Ziyin Qu, Jiecong Lu, Meggie Cheng, and Chenfanfu Jiang. Anisompm: Animating anisotropic damage mechanics. ACM Trans. Graph. 39, 4, Article 37 (July 2020), 2020.