Simulation

Deformation Simulation

Rina Fumoto

Introduction

Deformation is one of the most important aspects of the physically-based simulation. This report will review relevant literature and describe the method used for this project and implementation details. It also includes analysis and reflection on the method and implementation at the end.

Literature Review

Solid Mechanics

Physically-based deformation has a long history in Computer Graphics (Nealen et al. 2006). The essential properties for simulating deformation are stress and strain. Stress is defined as the force intensity and strain is given by a normalised displacement (Bergstrom 2015). The relation between these properties is called a constitutive model and this distinguishes the response of different materials. For example, Hooke's law states linear elasticity (Müller 2008) and St. Venant-Kirchhoff and Neo-Hookean model are constitutive models for hyperelastic materials (Sifakis and Barbic 2012; Wolper et al. 2019). There are a few models for controlling plasticity, such as Johnson-Cook (Bergstrom 2015), Cam-Clay, von Mises and Drucker-Prager models (Wolper et al. 2019). These models can be used to simulate the deformation of different types of materials.

Model Discretization

To simulate deformation, models must be discretised. There are different methods for model discretisation and these can be categorised into Lagrangian and Eulerian methods (Nealen et al. 2006). Lagrangian methods include mesh-based methods, such as the Finite Element Method, Boundary Element Method and Mass-Spring, and mesh-free methods, such as the Smoothed Particle Hydrodynamics. Eulerian methods are usually used for fluids.

Material Point Method

Another method is the Material Point Method (MPM), which combines the Lagrangian and Eulerian methods. It was introduced by Sulsky et al. (1995) as an extension of the Fluid Implicit Particle (Brackbill and Ruppel 1986) for simulating solid mechanics. The first use of MPM in graphics was in 2013 by Stomakhin et al. where they simulated snow. Since then, it has been used to simulate various behaviours of materials, such as diffusion (Han et al. 2021), fracture (Hu et al. 2018; Wang et al. 2019; Wolper et al. 2019; Wolper et al. 2020; Fei et al. 2021; Han et al. 2021), friction (Guo et al. 2018; Han et al. 2019; Fei et al. 2021) and phase transition (Stomakhin et al. 2014; Ding et al. 2019).

Method

The following method uses the traditional MPM based on Disney's paper (Stomakhin et al. 2013) to simulate the deformation of elastoplastic materials in two-dimension.

Particles to Grid

The first step is to transfer the mass and velocity from particles to the grid. Interpolation is done by using dyadic products of one-dimensional cubic B-splines:

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

where $\boldsymbol{i} = (i, j, k)$ is the grid index, $\boldsymbol{x}_p = (x_p, y_p, z_p)$ is the evaluation position, h is the grid spacing and

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

$N_{\boldsymbol{i}}^{h}(\boldsymbol{x}_p)$ will be represented as w_{ip} for later equations in this report.

Using the above equations, the particle masses are rasterized as follows:

$$m_{\boldsymbol{i}}^{n} = \sum_p m_p w_{ip}^{n} \tag{3}$$

Different from FLIP, MPM uses normalized weights for velocity as the weighting does not result in momentum conservation:

$$\boldsymbol{v}_{\boldsymbol{i}}^{n} = \frac{\sum_p \boldsymbol{v}_p^n m_p w_{ip}^n}{m_{\boldsymbol{i}}^{n}} \tag{4}$$

Compute Particle Volumes and Densities

The particle volumes and densities are computed only once during the first timestep.

First, estimate a cell's density as $\frac{m_{\boldsymbol{i}}^0}{h^2}$, then weight it back to the particles as

$$\rho_p^0 = \sum_{\boldsymbol{i}} \frac{m_{\boldsymbol{i}}^0 w_{ip}^0}{h^2} \tag{5}$$

Then, estimate particle volumes as

$$V_p^0 = \frac{m_p}{\rho_p^0} \tag{6}$$

Update Grid

The stress-based forces on the grid are computed as follows:

$$\boldsymbol{f}_{\boldsymbol{i}} = -\sum_p V_p^0 \frac{\partial \Psi}{\partial \boldsymbol{F}_E}(\boldsymbol{F}_{Ep}^n, \boldsymbol{F}_{Pp}^n)(\boldsymbol{F}_{Ep}^n)^T \nabla w_{ip}^n \tag{7}$$

where $\boldsymbol{F}_{Ep}^n$ and $\boldsymbol{F}_{Pp}^n$ are the elastic and plastic part of the gradient density and

$$\frac{\partial \Psi}{\partial \boldsymbol{F}_E}(\boldsymbol{F}_{Ep}^n, \boldsymbol{F}_{Pp}^n) = 2\mu_0 e^{\xi(1-J_{Pp}^n)}(\boldsymbol{F}_{Ep}^n - \boldsymbol{R}_{Ep}^n) + \lambda_0 e^{\xi(1-J_{Pp}^n)}(J_{Ep}^n - 1)J_{Ep}^n(\boldsymbol{F}_{Ep}^n)^{-T} \tag{8}$$

is the derivative of the elastoplastic potential energy density where μ_0 and λ_0 are the initial Lame coefficients, ξ is a dimensionless plastic hardening parameter, J is a determinant of $\boldsymbol{F}$ and $\boldsymbol{R}$ is the rotation matrix of the polar decomposition of $\boldsymbol{F} = \boldsymbol{RS}$, which can be expressed as $\boldsymbol{R} = \boldsymbol{UV}^T$ using the singular value decomposition $\boldsymbol{F} = \boldsymbol{U\Sigma V}^T$.

The gradient of the interpolation function ∇w_{ip} can be computed as:

$$\nabla w_{ip} = \nabla N_i^h(\boldsymbol{x}_p) = \begin{bmatrix} N'\left(\frac{1}{h}(x_p - ih)\right) N\left(\frac{1}{h}(y_p - jh)\right) \\ N\left(\frac{1}{h}(x_p - ih)\right) N'\left(\frac{1}{h}(y_p - jh)\right) \end{bmatrix}$$
$$N'(x) = \begin{cases} \frac{3}{2}x^2\frac{|x|}{x} - 2x, & 0 \le |x| < 1 \\ -\frac{1}{2}x^2\frac{|x|}{x} + 2x - 2\frac{|x|}{x}, & 1 \le |x| < 2 \\ 0, & otherwise \end{cases} \tag{9}$$

After computing the stress-based forces, external forces are added.

Using the computed forces, update the grid velocities as follows:

$$\boldsymbol{v}_i^* = \boldsymbol{v}_i^n + \Delta t m_i^{-1} \boldsymbol{f}_i^n \tag{10}$$

When a collision is detected, compute the local normal $\boldsymbol{n}$. If $v_n = \boldsymbol{v}_i^* \cdot \boldsymbol{n} \ge 0$, the bodies are separating so no collision is applied, otherwise, update the velocity as follow:

$$\boldsymbol{v}_i^{n+1} = \boldsymbol{v}_i^* - \boldsymbol{n} v_n \tag{11}$$

Update Deformation Gradients

First, update the elastic part of the deformation gradient temporarily by the following equations:

$$\begin{aligned} \nabla \boldsymbol{v}_p^{n+1} &= \sum\nolimits_i \boldsymbol{v}_i^{n+1} \left(\nabla w_{ip}^n\right)^T \\ \boldsymbol{F}_{Ep}^* &= \left(\boldsymbol{I} + \Delta t \nabla \boldsymbol{v}_p^{n+1}\right) \boldsymbol{F}_{Ep}^n \end{aligned} \tag{12}$$

Then, compute the singular value decomposition of $\boldsymbol{F}_{Ep}^*$:

$$\boldsymbol{F}_{Ep}^* = \boldsymbol{U}_p \boldsymbol{\Sigma}_p^* \boldsymbol{V}_p^T \tag{13}$$

To push the part of $\boldsymbol{F}_{Ep}^*$ that exceeds the critical deformation threshold, clamp the singular values to the permitted range as follow:

$$\boldsymbol{\Sigma}_p^{n+1} = clamp\left(\boldsymbol{\Sigma}_p^*, [1 - \theta_c, 1 + \theta_s]\right) \tag{14}$$

where θ_c and θ_s are critical compression and stretch.

Finally, update the elastic and plastic part of the deformation gradient as:

$$\begin{aligned} \boldsymbol{F}_{Ep}^{n+1} &= \boldsymbol{U}_p \boldsymbol{\Sigma}_p^{n+1} \boldsymbol{V}_p^T \\ \boldsymbol{F}_{Pp}^{n+1} &= \boldsymbol{V}_p \boldsymbol{\Sigma}_p^{n+1^{-1}} \boldsymbol{U}_p^T \boldsymbol{F}_{Ep}^* \boldsymbol{F}_{Pp}^n \end{aligned} \tag{15}$$

Grid to Particles

First, update the particle velocities using the combination of PIC and FLIP as follows:

$$\begin{aligned} \boldsymbol{v}_p^* &= (1 - \alpha) \sum\nolimits_i \boldsymbol{v}_i^{n+1} w_{ip}^n + \alpha \left(\boldsymbol{v}_p^n + \sum\nolimits_i \left(\boldsymbol{v}_i^{n+1} - \boldsymbol{v}_i^n\right) w_{ip}^n \right) \\ &= \sum\nolimits_i \boldsymbol{v}_i^{n+1} w_{ip}^n + \alpha \left(\boldsymbol{v}_p^n - \sum\nolimits_i \boldsymbol{v}_i^n w_{ip}^n \right) \end{aligned} \tag{16}$$

where α is the PIC-FLIP blending ratio.

After updating the particle velocities, handle collision same as the grid velocities using the equation 11 on $\boldsymbol{v}_p^*$ instead of $\boldsymbol{v}_i^*$.

Finally, update the particle positions as:

$$\boldsymbol{x}_p^{n+1} = \boldsymbol{x}_p^n + \Delta t \boldsymbol{v}_p^{n+1} \tag{17}$$

Implementation

I was going to implement the simulation in Houdini at the beginning. However, there are many components to understand to be able to implement it in Houdini, which may take too long to complete the implementation in time. Therefore, I decided to implement in C++ with NGL, which I am more familiar with, to focus on the MPM method and completion of the simulation instead of learning new technical skills.

Class Diagram

Figure 1: Class Diagram

This class diagram only includes the MPM class, which has all the variables and functions used for the simulation. Other classes in the program are used for GUI and visualisation of the simulation.

Eigen library is used for singular value decomposition. The private “eigenVec3” function is used to convert the NGL vectors to Eigen vectors as the results of the decomposition are Eigen matrices, which cannot be calculated with NGL vectors. The private “Interpolate” and “bSpline” functions are used to calculate the interpolation in equations 1 and 2 and “dInterpolate” and “dBSpline” functions are used to calculate the gradient of the interpolation in equations 9. Other private functions are used to operate the steps described in the method section. The public functions are called from NGLScene class to respond to button actions from GUI. Collision handling is simplified as no solid object is added to the simulation except for the solid cells around the simulation frame. It is handled by setting the velocities on the solid surfaces to zero.

GUI

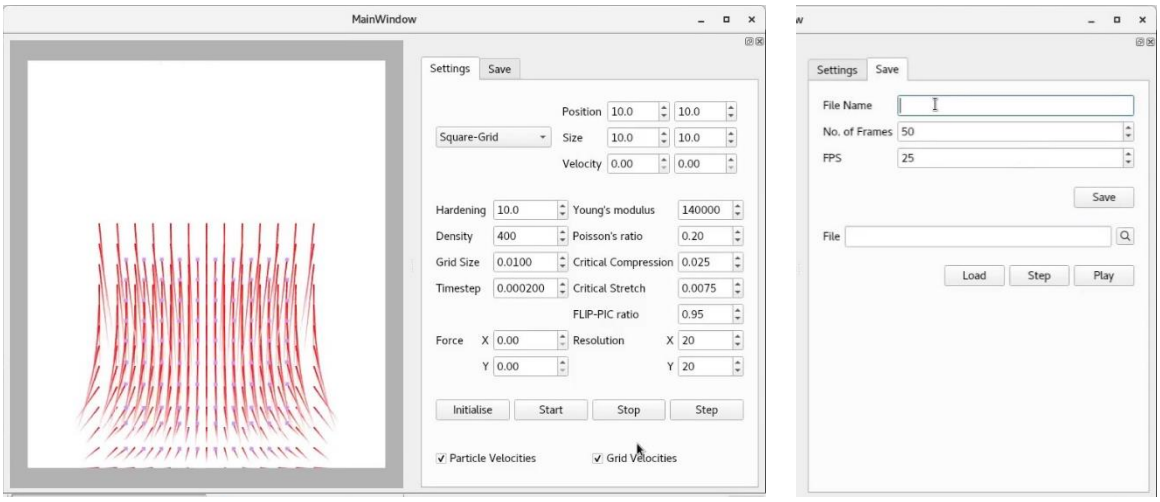

Figure 2: GUI. Visualisation and simulation settings on the left and saving and playing the simulation on the right.

The simulated particles are visualised using NGL and GUI was implemented using Qt as shown in the figure above to ease the testing process. There are four shapes to initialise the particles. I used Houdini to generate circle shapes and exported to CSV to be loaded to the simulation. The simulation settings can be edited via GUI to test the implementation with different parameters and the simulation can be played and paused by a button click. The step button allows simulating only one timestep. The particle velocities and grid velocities can be visualised with red lines with the checkboxes as shown on the left image in the figure 2. This helped to find out errors in the implementation.

While testing the simulation, I found out it does not work properly with a large timestep but testing with a small timestep makes the testing time longer. Therefore, I made a save function to save the particle positions as a geo file at each frame. The saved simulation can be played with the application or imported into Houdini. Sample simulations with different settings are included in the source code and these can be played by loading it on the save tab. A sample Houdini scene for comparing those simulations can be found in test.hipnc.

Analysis & Reflection

The implemented system successfully simulates the deformation of objects with different shapes and materials in various situations. Simulating with large timesteps and small grid sizes resulted in particles to be exploded. However, simulating with small timesteps takes a long time. I found that finding a suitable timestep is very important for the MPM simulation. Changing the parameters for the material produced different simulations as shown in figure 3.

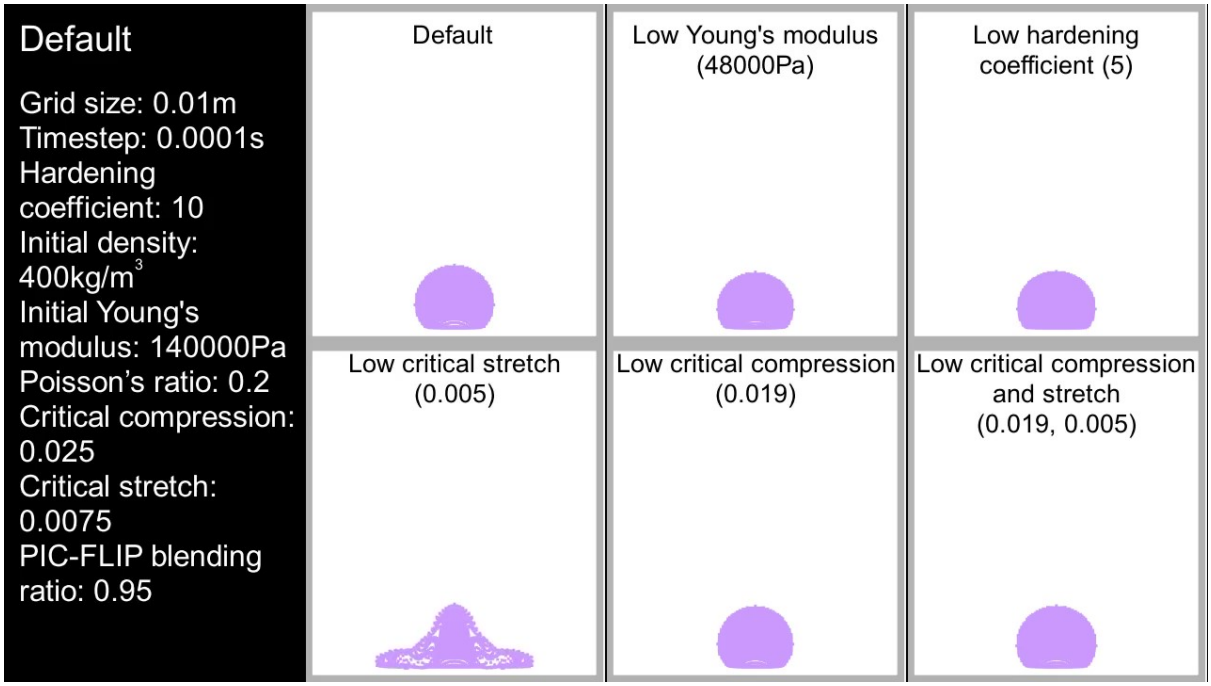

Figure 3: Simulation with different parameter settings.

Simulation changed dramatically with low critical stretch but other parameter changes produced similar results. Even though the results look similar, there are slight differences in each simulation when looking at these closely. The difference should be more obvious with larger simulations but it could not be tested due to the time constraint.

Although the system can simulate deformation, there are many improvements that can be made. For example, the simulation can be integrated into 3D simulation and made into a Houdini plugin to allow more user controls. This project used the traditional method introduced in 2013 but there are many improved approaches introduced in recent papers like Moving Least Squares MPM (Hu et al. 2018). Also, different constitutive models can be applied. This system is suitable for testing a simulation with a small number of particles as the simulation takes a very long time. The performance can be improved with GPU optimization (Gao et al. 2018; Wang et al. 2020). In addition, it can be improved to simulate interactions between multiple materials and other behaviours like diffusion and fracture.

Conclusion

This report covered the related background and previous works, the detailed implementation method and analysis and reflections of the implementation. Overall, the project can be considered successful due to the fact that the initial objective to implement deformation simulation using MPM has been achieved.

References

Bergstrom, J.S., 2015. Mechanics of solid polymers: theory and computational modeling. William Andrew.

Brackbill, J.U. and Ruppel, H.M., 1986. FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. Journal of Computational physics, 65(2), pp.314-343.

Ding, M., Han, X., Wang, S., Gast, T.F. and Teran, J.M., 2019. A thermomechanical material point method for baking and cooking. ACM Transactions on Graphics (TOG), 38(6), pp.1-14.

Fei, Y., Guo, Q., Wu, R., Huang, L. and Gao, M., 2021. Revisiting integration in the material point method: a scheme for easier separation and less dissipation. *ACM Transactions on Graphics (TOG)*, *40*(4), pp.1-16.

Gao, M., Wang, X., Wu, K., Pradhana, A., Sifakis, E., Yuksel, C. and Jiang, C., 2018. GPU optimization of material point methods. ACM Transactions on Graphics (TOG), 37(6), pp.1-12.

Guo, Q., Han, X., Fu, C., Gast, T., Tamstorf, R. and Teran, J., 2018. A material point method for thin shells with frictional contact. ACM Transactions on Graphics (TOG), 37(4), pp.1-15.

Han, C., Xue, T. and Aanjaneya, M., 2021, October. A Lagrangian Particle‐based Formulation for Coupled Simulation of Fracture and Diffusion in Thin Membranes. In Computer Graphics Forum (Vol. 40, No. 7, pp. 97-108).

Han, X., Gast, T.F., Guo, Q., Wang, S., Jiang, C. and Teran, J., 2019. A hybrid material point method for frictional contact with diverse materials. Proceedings of the ACM on Computer Graphics and Interactive Techniques, 2(2), pp.1-24.

Hu, Y., Fang, Y., Ge, Z., Qu, Z., Zhu, Y., Pradhana, A. and Jiang, C., 2018. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics (TOG)*, *37*(4), pp.1-14.

Müller, M., Stam, J., James, D. and Thürey, N., 2008. Real time physics: class notes. In ACM SIGGRAPH 2008 classes (pp. 1-90).

Nealen, A., Müller, M., Keiser, R., Boxerman, E. and Carlson, M., 2006, December. Physically based deformable models in computer graphics. In Computer graphics forum (Vol. 25, No. 4, pp. 809-836). Oxford, UK: Blackwell Publishing Ltd.

Sifakis, E. and Barbic, J., 2012. FEM simulation of 3D deformable solids: a practitioner's guide to theory, discretization and model reduction. In Acm siggraph 2012 courses (pp. 1-50).

Stomakhin, A., Schroeder, C., Chai, L., Teran, J. and Selle, A., 2013. A material point method for snow simulation. *ACM Transactions on Graphics (TOG)*, *32*(4), pp.1-10.

Stomakhin, A., Schroeder, C., Jiang, C., Chai, L., Teran, J. and Selle, A., 2014. Augmented MPM for phase-change and varied materials. ACM Transactions on Graphics (TOG), 33(4), pp.1-11.

Sulsky, D., Zhou, S.J. and Schreyer, H.L., 1995. Application of a particle-in-cell method to solid mechanics. Computer physics communications, 87(1-2), pp.236-252.

Wang, S., Ding, M., Gast, T.F., Zhu, L., Gagniere, S., Jiang, C. and Teran, J.M., 2019. Simulation and visualization of ductile fracture with the material point method. Proceedings of the ACM on Computer Graphics and Interactive Techniques, 2(2), pp.1-20.

Wang, X., Qiu, Y., Slattery, S.R., Fang, Y., Li, M., Zhu, S.C., Zhu, Y., Tang, M., Manocha, D. and Jiang, C., 2020. A massively parallel and scalable multi-GPU material point method. ACM Transactions on Graphics (TOG), 39(4), pp.30-1.

Wolper, J., Chen, Y., Li, M., Fang, Y., Qu, Z., Lu, J., Cheng, M. and Jiang, C., 2020. AnisoMPM: Animating anisotropic damage mechanics: Supplemental document. ACM Trans. Graph, 39(4).

Wolper, J., Fang, Y., Li, M., Lu, J., Gao, M. and Jiang, C., 2019. CD-MPM: continuum damage material point methods for dynamic fracture animation. *ACM Transactions on Graphics (TOG)*, *38*(4), pp.1-15.