Master's Project

Crack Propagation using Material Point Method and J-integral

Rina Fumoto

MSc Computer Animation and Visual Effects
National Centre for Computer Animation
Bournemouth University

24th August 2022

Abstract

Dynamic fractures can be seen in various situation in the real life but recreating it in a computer is difficult as there are many properties to be considered and simulating it requires complicated computations. Recently, Material Point Method (MPM) has been one of the popular methods due to its ability to automatically support arbitrarily large topological changes, handle natural collisions and simulate various materials. This project combines MPM with J-integral to calculate crack tip parameters which are used to determine the crack propagation. This is implemented as a custom solver in Houdini and the solver is used to create a Houdini Digital Asset. The developed asset provides a user to simulate a reasonable crack propagation with some user controls.

Keywords: Material point method (MPM), crack propagation, dynamic fracture, J-integral, VFX, Animation, Houdini, custom solver, digital asset.

Acknowledgements

I would like to take this opportunity to thank the people who have supported me not only with this project but also with my degree over the year.

I would like to thank my supervisors, Jon Macey and Jian Chang, for their guidance and supervision with this project, as well as the staff at the National Centre for Computer Animation for their work, help and support throughout my course.

Finally, a big thanks to all my friends for all their support and encouragement.

Contents

List of Figures

List of Tables

1 Introduction

Different types of cracks and fractures can be found in our everyday life. The fracture patterns changes depending on the situation and reproducing these detailed patterns in Computer Graphics is challenging as it requires a complex computation for the mechanics behind it. A common approach for generating fractures in Computer Graphics is to use a pre-fractured object or pre-defined fracture patterns. This allows more controls for artists but it does not consider any physics behind it. Physically based fracture simulation is highly desired in Computer Graphics to increase realism and plausibility. There have been many studies undertaken for physically based fracture simulation in Computer Graphics and Engineering. A major approach for fracture simulation currently used is a mesh-based method, such as Finite Elements Method and Boundary Elements Method. These methods handle cracks by remeshing the objects whenever the crack propagates. Another popular method recently used is a meshless method called Material Point Method (MPM). This method combines Lagrangian particles with Eulerian grids. The simulation object is represented as particles and these particles are advected by the background grid. This method is suitable for computing large deformations and extreme material distortion.

This project focuses on physically based crack propagation simulated using MPM. Crack handling is applied to the traditional MPM using multiple velocity fields and propagation is computed with the crack tip properties calculated using J-integral. The overall aim of this project is to develop a digital asset in Houdini with various user controls to allow users to simulate physically based crack propagation with different settings. This thesis will first review previous works for cracks and fractures. After that, the method and tool used for this project will be introduced. The following sections will describe how the final digital asset was implemented in detail and discuss the results. Finally, the overall project conclusions and ideas for future development will be provided.

2 Previous work

Fracture mechanics is a part of solid mechanics, which has been studied in Engineering as well as in Computer Graphics. Early work on fracture in Computer Graphics is the paper by Terzopoulos and Fleischer (1988), which introduced inelastic deformation with fracture. Since then, many methods for fracture simulation have been proposed. The methods can be separated into two categories, non-physically based and physically based methods.

2.1 Non-physically Based

The practical approach for simulation in Computer Graphics is a geometry-based method. This method pre-defines the fracture pattern on geometry and glues the pieces together using constraints. When simulating the geometry as rigid bodies, the constraints break when a certain impact is applied. The majority of geometry-based methods use Voronoi to apply fracture patterns to geometries (Raghavachary 2002; Hellrung et al. 2009). This approach is popular due to its simplicity, speed and flexibility. However, it requires careful preparation as the pattern does not align with the impact. Su et al. (2009) and Müller et al. (2013) improved this method to apply the pattern in real-time based on the impact.

Another non-physically based approach is an example-based method. This method uses a real-life object as an example to produce a realistic fracture pattern. An image of a crack pattern taken from a real-life object is often used for this method to map onto a 3D model or indicate control points for the pattern (Martinet et al. 2004; Desbenoit et al. 2005). Glondu et al. (2012) use statistics taken from a real-life object to generate a crack pattern.

2.2 Physically Based

One of the physically based methods is a mass-spring method (Muguercia et al. 2014). This method approximates the object by a set of mass particles and the particles are joined by springs. This is one of the simplest ways to simulate but it is difficult to provide shear or bending resistance and express material properties. Also, this method cannot provide exact fracture position and orientation.

On the other hand, Finite Element Method (FEM) can deal with most of these problems (O'Brien and Hodgins 1999). FEM partitions the object into elements and each element describes the object material by an equation. Stress tensors are computed over these elements, which enables to determine the position and direction of the crack propagation. O'Brien and Hodgins (1999) proposed brittle fracture with FEM focusing on the appearance of the fracture and efficiency, which is important in computer animation. This method has been widely used and extended in Computer Graphics (Bao et al. 2007; Koschier et al. 2014; Müller and Gross 2004). This method is also used for surface fracture (Iben and O'brien 2009) and real-time fracture simulations (Parker and O'Brien 2009; Glondu, Marchal and Dumont 2012; Müller et al. 2001). Since this method requires the continuity of material in elements, there is a limitation when dealing with discontinuity, such as cracks. To handle the crack propagation, remeshing is required throughout the simulation, which brings the computational burden. Extended FEM (XFEM) was developed by Belytschko and Black (1999) to solve the problem by adding enrichment functions containing a discontinuous displacement field. This was improved by Moës et al. (1999) to separate the entire crack from the mesh. Later, XFEM was extended to support cracks with branches by Daux et al. (2000) and to enable high-resolution simulation efficiently by Chitalu et al. (2020). These improvements allow to simulate crack propagation without remeshing but it still has a problem with mesh distortion for large deformation simulations.

Another mesh-based method is Boundary Element Method (BEM) (Hahn and Wojtan 2015; Zhu et al. 2015; Hahn and Wojtan 2016). This method applies boundary conditions and computes stress tensors on the surface mesh. The advantage of this method over FEM is that it can be more efficient as the computation focuses only on the surfaces of the model. However, it has a limit to simulate models with spatially varied elasticity values.

Finally, there is a particle-based method called Material Point Method (MPM) which this paper focuses on. This method was introduced by Sulsky et al. (1995) as an extension of Fluid Implicit Particle (FLIP) method to simulate dynamic solid mechanics. This method discretises the simulation object into particles and rasterises the particle properties to a background grid. The computation is done on the grid and the updated grid is used to update the particle properties. This method gets rid of the complex remeshing problem as it uses particles to represent the object. Since the first use of MPM in Computer Graphics by Stomakhin et al. (2013) for simulating snow, it has been used for simulating various types of solid materials, such as sand (Daviet and Bertails-Descoubes 2016; Klár et al. 2016), foam (Yue et al. 2015), sponge (Ram et al. 2015), cloth, knit and hair (Jiang et al. 2017). The traditional MPM enforces continuous displacements so it cannot include discontinuity, like cracks. There have been many studies to handle discontinuity with MPM in Computer Graphics. Wretborn et al. (2017) extended the traditional MPM to simulate cracks using a multi-body solver. However, this approach requires pre-fracturing the object and glueing them together with constraints. Hu et al. (2018) introduced Moving Least Squares Material Point Method (MLS-MPM) with Compatible Particle-In-Cell (CPIC) algorithm that allows dynamic material cutting with sharp boundaries. Wolper et al. (2019) presented Continuum Damage Material Point Methods (CD-MPM). They adopted Continuum Damage Mechanics (CDM) and phase-field theory for dynamic brittle and ductile fracture. Later, Wolper et al. (2020) introduced AnisoMPM, which extended CD-MPM to encode material anisotropy. Recently, Fan et al. (2022) combined MPM, CDM and rigid body dynamics to enable accurate and robust fracture simulation to complement fast-and-rigid shatter effects.

In Engineering, Nairn (2003) achieved discontinuity in MPM with a method called CRAMP (CRAcks with Material Points). This method describes cracks as massless particles connected with lines and handles displacement discontinuity using multiple velocity fields. A year later, Guo and Nairn (2004) developed a method for calculating fracture parameters, such as J-integral and stress intensity factors. This is used for dynamic crack propagation in both 2D and 3D (Guo and Nairn 2017). A crack is represented by a line in 2D and by a surface in 3D. 3D crack propagation is more complicated as fracture parameters must be calculated at every point at the crack front. Also, it requires additional treatment after propagating the crack front to build a new crack front by adding a point or merging two points.

3 Technical Background

3.1 Numerical Method

The following method extends the traditional MPM based on the paper by Stomakhin et al. (2013) to handle cracks (Nairn 2003). Also, this method uses J-integral to calculate the crack propagation.

The following is an overview of the method used for this project.

1. Particles to Grid
2. Update Grid Velocity
3. Crack Surface Contact Handling
4. Grid-Based Collision Handling
5. Crack Surface Contact Handling
6. Update Deformation Gradient
7. Particle Advection
8. Crack Particle Advection
9. Calculate J-Integral
10. Propagate Crack

3.1.1 Particles to Grid

The first step in MPM is to transfer mass and velocity from particles to the grid. The traditional MPM uses only one velocity field but for this project, three types of velocities fields are used to handle cracks. One of the fields is used for particles on the same side of all cracks as the node, one for particles above a crack relative to the node, and one for particles below a crack relative to the node. Nairn (2003) introduced a line-crossing algorithm to determine the appropriate field for the combination of particle and node efficiently and precisely.

The line-crossing algorithm checks if the line drawn from particle to node intersects with the crack. It uses the signed area of the triangle of three vertices $\boldsymbol{x_1}$, $\boldsymbol{x_2}$ and $\boldsymbol{x_3}$ calculated as follows:

$$Area = \boldsymbol{x_1}(\boldsymbol{y_2} - \boldsymbol{y_3}) + \boldsymbol{x_2}(\boldsymbol{y_3} - \boldsymbol{y_1}) + \boldsymbol{x_3}(\boldsymbol{y_1} - \boldsymbol{y_2}) \tag{1}$$

If the area is positive, the path from $\boldsymbol{x_1}$ to $\boldsymbol{x_3}$ is counterclockwise, if negative, it is clockwise, and if zero, the points are collinear.

First, check if the crack intersects the rectangle defined by the particle and the node. If it does, calculate the sign of the areas of triangles $(\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3})$, $(\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_4})$, $(\boldsymbol{x_3}, \boldsymbol{x_4}, \boldsymbol{x_1})$ and $(\boldsymbol{x_3}, \boldsymbol{x_4}, \boldsymbol{x_2})$ where $\boldsymbol{x_1}$ is the particle position, $\boldsymbol{x_2}$ is the node position, $\boldsymbol{x_3}$ and $\boldsymbol{x_4}$ are the endpoints of a crack segment.

If the results are $(-++-)$, $(-++0)$, $(0++0)$, or $(-0+0)$, the particle is above the crack, if $(+--+)$, $(+--0)$, $(0--0)$, or $(+0-0)$, the particle is below the crack, otherwise, the line does not cross the segment.

If the line crosses a crack segment, the normal to the crossed segment is saved to the node. The average of the saved normal vectors at the node is used as a crack surface normal at the node, which will be used in the later steps.

After the appropriate fields are determined, masses and velocities can be rasterized as follows:

$$\begin{aligned} m_{i,j}^n &= \sum_p m_p w_{ip}^n \delta_{j,\nu(p,i)} \\ \boldsymbol{v}_{i,j}^n &= \frac{\sum_p \boldsymbol{v}_p^n m_p w_{ip}^n \delta_{j,\nu(p,i)}}{m_i^n} \end{aligned} \tag{2}$$

where $m_{i,j}^n$ and $\boldsymbol{v}_{,j} i^n$ are the rasterized mass and velocity on the velocity field j at node i, m_p and $\boldsymbol{v}_p^n$ are the particle mass and velocity, w_{ip}^n is a weighting function, $\nu(p,i)$ is the determined velocity field and $\delta_{j,\nu(p,i)}$ is the Kronecker delta function.

3.1.2 Update Velocity

After the particle masses and velocities are transferred to fields, stress-based forces are computed on the field. First, the particle stress is calculated as:

$$\boldsymbol{\sigma}_p = \frac{1}{J_p^n} \frac{\partial \Psi}{\partial \boldsymbol{F}_E} (\boldsymbol{F}_{E_p}^n, \boldsymbol{F}_{P_p}^n)(\boldsymbol{F}_{E_p}^n)^T \tag{3}$$

$$\frac{\partial \Psi}{\partial \boldsymbol{F}_E}(\boldsymbol{F}_{E_p}^n, \boldsymbol{F}_{P_p}^n) = 2\mu_0 e^{\xi(1-J_{P_p}^n)}(\boldsymbol{F}_{E_p}^n - \boldsymbol{R}_{E_p}^n) + \lambda_0 e^{\xi(1-J_{P_p}^n)}(J_{E_p}^n - 1)J_{E_p}^n(\boldsymbol{F}_{E_p}^n)^{-T} \tag{4}$$

where $\boldsymbol{F}_E$ and $\boldsymbol{F}_P$ are elastic and plastic part of deformation gradient, J is the determinant of $\boldsymbol{F}$, μ_0 and λ_0 are the initial Lame coefficients, ξ is a dimensionless plastic hardening parameter 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 $\boldsymbol{F} = \boldsymbol{U\Sigma V}^T$.

Then, the force is calculated as:

$$\boldsymbol{f}_i = -\sum_p V_p^n \boldsymbol{\sigma}_p \boldsymbol{\nabla} \boldsymbol{w}_{ip}^n \tag{5}$$

where V_p^n is particle volume and $\boldsymbol{\nabla} \boldsymbol{w}_{ip}^n$ is a gradient of weighting function.

After adding external forces to the computed force, update the velocities on the fields as:

$$\boldsymbol{v}_{i,j}^* = \boldsymbol{v}_{i,j}^n + \Delta t m_i^{-1} \boldsymbol{f}_i^n \tag{6}$$

3.1.3 Crack Surface Contact Handling

The grid velocities were updated in the previous step, which can cause the intersection between the crack surfaces. It is important to handle the collision between the crack surfaces every time the grid velocities are updated to prevent non-physical changes.

Nairn (2003) introduced the identification of crack surface contact using nodal volume. Later, Guo and Nairn (2006) updated the method using nodal displacement instead of volume, which is simpler and more robust.

First, check if the crack surfaces are moving towards each other, i.e.,

$$(\boldsymbol{v}_{i,a} - \boldsymbol{v}_{i,b}) \cdot \hat{\boldsymbol{n}}_i < 0 \tag{7}$$

where $\boldsymbol{v}_{i,a}$ and $\boldsymbol{v}_{i,b}$ are nodal velocities for the particles above and below the crack and $\hat{\boldsymbol{n}}_i$ is the crack surface normal calculated during the line-crossing algorithm.

If the crack surfaces are moving towards each other, check if the surfaces are in contact using nodal displacements:

$$(\boldsymbol{u}_{i,a} - \boldsymbol{u}_{i,b}) \cdot \hat{\boldsymbol{n}}_i < 0 \tag{8}$$

If contact between the two surfaces are detected, update the velocities as follows:

$$\boldsymbol{v}^*_{i,j} = \boldsymbol{v}^*_{i,j} - \hat{\boldsymbol{n}}_i((\boldsymbol{v}^*_{i,j} - \boldsymbol{v}_{i,c}) \cdot \hat{\boldsymbol{n}}_i) \tag{9}$$

where $\boldsymbol{v}_{i,c}$ is the center-of-mass velocity of the two velocity fields, computed as:

$$\boldsymbol{v}_{i,c} = \frac{m_{i,a}\boldsymbol{v}_{i,a} + m_{i,b}\boldsymbol{v}_{i,b}}{m_{i,a} + m_{i,b}} \tag{10}$$

3.1.4 Grid-Based Collision Handling

The final step of the computations on the grid is to handle collisions with other objects based on the velocities on the grid.

When a collision is detected, compute the local normal $\boldsymbol{n}$. If the collided objects are moving towards each other, i.e., $v_n = \boldsymbol{v}^*_{i,j} \cdot \boldsymbol{n} \leq 0$, the velocity must be updated as follows:

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

Here, the contact between the crack surfaces must be handled again following the method described in Section 3.1.3 as the grid velocities were updated in this step.

3.1.5 Update Deformation Gradient

After the calculations on the grid are completed, update the deformation gradients of the particles.

First, update the elastic part of the deformation gradient temporarily as follows:

$$\boldsymbol{\nabla}\boldsymbol{v}^{n+1}_p = \sum_i \boldsymbol{v}^{n+1}_{i,\nu(p,i)}(\boldsymbol{\nabla}\boldsymbol{w}^n_{ip})^T \tag{12}$$

$$\boldsymbol{F}^*_{Ep} = (\boldsymbol{I} + \Delta t \boldsymbol{\nabla}\boldsymbol{v}^{n+1}_p)\boldsymbol{F}^n_{Ep} \tag{13}$$

Next, take the part of $\boldsymbol{F}^*_{Ep}$ that exceeds the critical deformation threshold and push it into $\boldsymbol{F}^*_{Pp}$. This can be done by computing the singular value decomposition of $\boldsymbol{F}^*_{Ep}$ and clamping the singular values to the permitted range:

$$\boldsymbol{F}^*_{Ep} = \boldsymbol{U}_p \boldsymbol{\Sigma}^*_p \boldsymbol{V}^T_p \tag{14}$$

$$\boldsymbol{\Sigma}^{n+1}_p = clamp(\boldsymbol{\Sigma}^*_p, [1-\theta_c, 1+\theta_s]) \tag{15}$$

where θ_c and θ_s are critical compression and stretch, respectively.

Finally, update the deformation gradients as:

$$\boldsymbol{F}^{n+1}_{Ep} = \boldsymbol{U}_p \boldsymbol{\Sigma}^{n+1}_p \boldsymbol{V}^T_p \tag{16}$$

$$\boldsymbol{F}^{n+1}_{Pp} = \boldsymbol{V}^T_p {\boldsymbol{\Sigma}^{n+1}_p}^{-1} \boldsymbol{U}^T_p \boldsymbol{F}^*_{Ep} \boldsymbol{F}^n_{Pp} \tag{17}$$

3.1.6 Particle Advection

The traditional MPM by Stomakhin et al. (2013) uses the combination of PIC and FLIP. However, this project uses multiple velocity fields to handle cracks and the appropriate field for the combination of particle and node can be changed in each step. Therefore, the velocities from the previous step cannot be used for the next step. In this project, the grid velocities are simply transferred to particles using PIC only as:

$$\boldsymbol{v}^{n+1}_p = \sum_i \boldsymbol{v}^{n+1}_{i,\nu(p,i)} w^n_{ip} \tag{18}$$

After transferring the velocities, collisions are handled based on particle velocities following the method described in Section 3.1.4.

Finally, update particle positions by:

$$\boldsymbol{x}^{n+1}_p = \boldsymbol{x}^n_p + \Delta t \boldsymbol{v}^{n+1}_p \tag{19}$$

3.1.7 Crack Particle Advection

As the object has been moved by updating the particle positions, the crack must be moved along with the object. As the crack is represented by massless particles, the update of crack position can be done by advecting the crack particles similar to the previous step. Instead of using a nodal velocity from one of the fields, use the centre of mass velocity of each node, which can be calculated as follows:

$$\boldsymbol{v}_{i,c}^{n+1} = \frac{\sum_j m_{i,j}^{n+1} \boldsymbol{v}_{i,j}^{n+1} \varphi_{i,j}}{\sum_j m_{i,j}^{n+1} \varphi_{i,j}} \tag{20}$$

where $\varphi_{i,j}$ is 1 if velocity field j exists or 0 if not.

Then, the computed centre of mass velocities is transferred to the crack particles and update the particle position as:

$$\boldsymbol{v}_{cp}^{n+1} = \sum_i \boldsymbol{v}_{i,c}^{n+1} w_{ip}^{n} \tag{21}$$

$$\boldsymbol{x}_{cp}^{n+1} = \boldsymbol{x}_{cp}^{n} + \Delta t \boldsymbol{v}_{cp}^{n+1} \tag{22}$$

where $\boldsymbol{v}_{cp}^{n+1}$ and $\boldsymbol{x}_{cp}^{n+1}$ are the velocity and position of the crack particles.

The crack particles track the crack position as well as the displacements of crack surfaces. The displacements can be updated similar to the above:

$$\boldsymbol{x}_{a}^{n+1} = \boldsymbol{x}_{a}^{n} + \Delta t \sum_i \boldsymbol{v}_{i,a}^{n+1} w_{ip}^{n} \tag{23}$$

$$\boldsymbol{x}_{b}^{n+1} = \boldsymbol{x}_{b}^{n} + \Delta t \sum_i \boldsymbol{v}_{i,b}^{n+1} w_{ip}^{n} \tag{24}$$

where $\boldsymbol{x}_{a}^{n+1}$ and $\boldsymbol{x}_{b}^{n+1}$ are the displacements of the crack surfaces above and below the crack.

The algorithm until here can be used for MPM deformation simulation with crack handling.

3.1.8 J-integral

The next step is to calculate J-integral at the crack tip to simulate crack propagation. The following method is based on the papers by Guo and Nairn (2004, 2006).

There are mainly three steps to calculate J-integral:

1. Generate a contour

2. Transfer particle values to the contour

3. Calculate J-integral

The first step is to generate contour around the crack tip. As you can see in Figure 1, the contour can be a circular path or a rectangular path with n segments. A circular path with a radius of 2 times of cell size with 16 segments is used for this project as recommended by Guo and Nairn (2006).

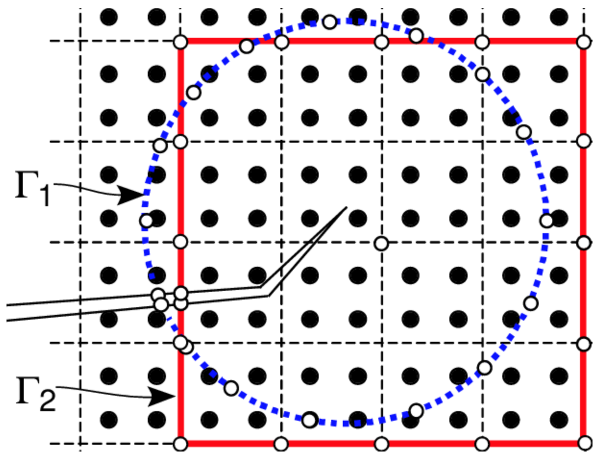


Figure 1: Two possible J-integral contours (Guo and Nairn 2004)

This contour can be generated by creating a circle around the crack tip, finding the intersection between the circle and the crack surfaces and dividing the path from the intersection with the surface below the crack to the intersection with the surface above into 16 segments. The contour ends up with 17 points along the circle.

The next step is to transfer the values required for the J-integral calculation from the particles to the points on the contour using:

$$q_k = \frac{\sum_p m_p q_p w_{kp} \delta_{j,\nu(p,k)}}{\sum_p m_p w_{kp} \delta_{j,\nu(p,k)}} \tag{25}$$

where q_k is the transferred value at the point k on the contour. To transfer values from the particles on the same side of the crack, $j = 0$ or 2 is used for the first half of the points and $j = 0$ or 1 is used for the second half of the points. For this project, mass, velocity, displacement, elastic and plastic parts of deformation gradient, and stress are transferred.

Finally, calculate the J-integral using the transferred values as follows:

$$J = J_x \cos\theta + J_y \sin\theta \tag{26}$$

where J_x and J_y are components of the J-integral in x and y axis and θ is the crack propagation angle measured from x axis.

The components of J integral are calculated as:

$$J_m = \sum_{k}^{n-1} (F_m^k + F_m^{k+1}) \frac{\Delta_k}{2} (m = x, y) \tag{27}$$

$$F_m^k = (W_k + K_k) n_m - \boldsymbol{\sigma}_k \hat{\boldsymbol{n}} \cdot \frac{\partial \boldsymbol{u}_k}{\partial x_m} \tag{28}$$

where Δ_k is the length of segment k, W and K are strain and kinetic energy densities, n_m is a component of the unit normal vector to the J-Integral contour and $\boldsymbol{u}_k$ is displacement. As this project uses the elasto-plastic constitutive model introduced by Stomakhin et al. (2013), elasto-plastic energy density is used for the strain energy density W instead of the stress-work density used in the papers (Guo and Nairn 2004, 2006). The energy densities can be computed as follows:

$$W = \mu_0 e^{\xi(1-J_P)} \|\boldsymbol{F}_E - \boldsymbol{R}_E\|_F^2 + \frac{\lambda_0 e^{\xi(1-J_P)}}{2} (J_E - 1)^2 \tag{29}$$

$$K = \frac{1}{2} \rho \dot{u}_i \dot{u}_i \tag{30}$$

The computed J-integral J can be converted into mode I and mode II stress intensity factors K_I and K_{II}.

For slow crack growth, the stress intensity factors can be calculated as:

$$K_I = \frac{\delta_I}{\delta} \sqrt{JE}, K_{II} = \frac{\delta_{II}}{\delta} \sqrt{JE} \tag{31}$$

for plane stress, and

$$K_I = \frac{\delta_I}{\delta} \sqrt{\frac{JE}{1-\nu^2}}, K_{II} = \frac{\delta_{II}}{\delta} \sqrt{\frac{JE}{1-\nu^2}} \tag{32}$$

for plane strain where δ_I and δ_{II} are crack opening and shearing displacements near the crack tip, δ is the magnitude of the crack opening displacement and E and ν are Young's modulus and Poisson's ratio.

3.1.9 Propagation

The final step is to propagate the crack. To simulate the propagation, the criteria and direction of the propagation must be predicted. For this project, the maximum hoop stress criterion used in the paper by Guo and Nairn (2017) is used. This criterion states that the direction of the propagation is in the direction normal to the maximum principal stress:

$$\theta_c = 2\tan^{-1}(\frac{\alpha \pm \sqrt{\alpha^2+8}}{4}) \tag{33}$$

where $\alpha = \frac{K_I}{K_{II}}$ is a mixed-mode ratio.

The crack will propagate when the equivalent stress intensity factor reaches a critical value, K_c, i.e.,

$$K_I \cos^3\frac{\theta_c}{2} + 3K_{II}\cos\frac{\theta_c}{2}\sin\frac{\theta_c}{2} \geq K_c \tag{34}$$

When the above condition is met, add a new crack particle in the direction θ_c at a distance of half a cell size. This new particle represents a new crack tip.

3.2 Houdini

The software used for this project is Houdini from SideFX (SideFX 2022b), which is one of the popular software used in the VFX industry. It was first released in 1996 and it's been used for creating visual effects in TV, films and games (SideFX 2022a). Houdini is a node-based software that can create procedural content. There are many tools to construct the nodes and networks automatically which allows artists to work more easily. This software can be used for various areas of VFX and animation productions, such as modelling, animation, effects, simulation, rendering, and compositing.

This software is advanced for creating dynamic simulations. Dynamics networks in Houdini enable users to create different types of simulations, such as cloth, rigid body, pyro and fluid. Currently, there is no MPM solver or any solvers that handle physically based fractures in Houdini. Fracture simulation is currently created using a rigid body dynamics solver with pre-fractured geometry with constraints. Therefore, a custom solver to simulate crack propagation using MPM and J-integral must be developed from scratch in this project.

In the DOP network in Houdini, there are many gas microsolvers that can be used to develop a custom solver. Since MPM is an extension of FLIP, some of the microsolvers used in the default FLIP solver in Houdini can be used for developing an MPM solver.

Other processes needed for the algorithms are implemented using Geometry Wrangle node and Gas Field Wrangle nodes with VEX. VEX is a high-performance expression language based on C, C++ and RenderMan shading languages (SideFX 2022e). It is used in several places in Houdini for writing shaders and custom nodes. The VEX codes run on each element of input geometry, like point, primitive and voxel. It can read, modify and add attributes of the elements. VEX ch functions create custom channels to be evaluated and return the values. The wrangle nodes accept four virtual inputs which can be accessed with @OpInput1-4 string parameters with VEX. There are various VEX functions to access simulation fields as volume, such as volumesample, volumeindex and volumeres. When accessing simulation particles from Gas Field Wrangle nodes, VEX pcopen function can be used, which imports the particles as a point cloud. The imported point cloud can be iterated with pciterate function and attributes of each point can be imported with pcimport function. The implementation details using these functions can be found in Section 4.

Houdini lets users convert their networks into a reusable custom node called Houdini Digital Asset (HDA) (SideFX 2022f). A user interface for the digital asset can be created by promoting parameters from the contained nodes to the asset node. Users can document the digital assets formatted with wiki markup (SideFX 2022d). This can be accessed from the Help button on the toolbar in the parameter editor of the node.

4 Implementation

To ensure that every step works, the implementation was done step by step. First, a custom solver for MPM is developed and it is integrated to handle cracks. Next, the calculation of J-integral is added to compute crack propagation. Finally, the solver is wrapped into Houdini Digital Asset to allow more user controls. Any information about nodes is found in the Houdini documentation (SideFX 2022c).

4.1 Custom Solver

The custom solver was developed with a variety of microsolvers and wrangle nodes inside a DOP network. These nodes are merged and connected to a Multiple Solver which handles a simulation by more than one solver in order at each time step.

4.1.1 MPM

The first step of the implementation is to prepare the simulation geometry and fields.

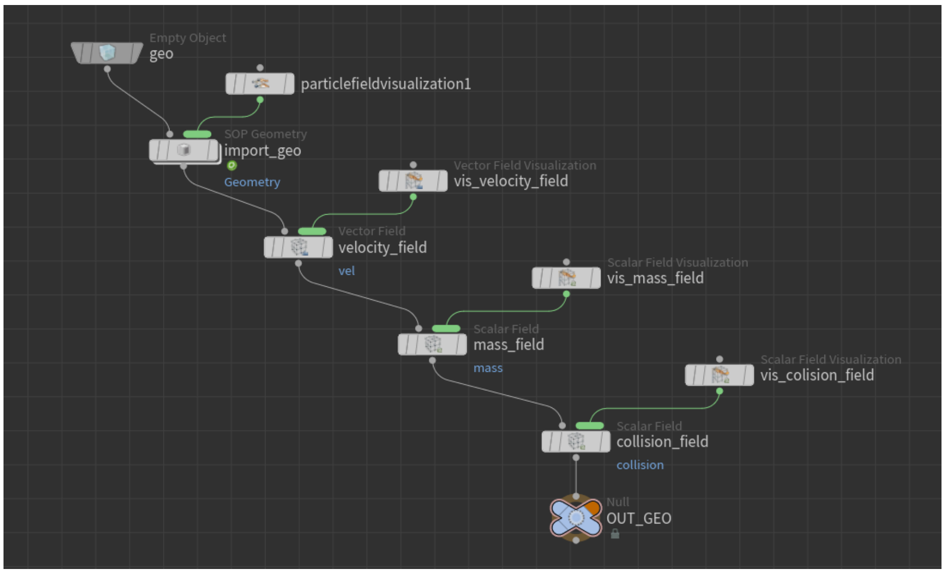


Figure 2: Import simulation geometry and initialise fields.

The Empty Object node is the first node for setting up the simulation object. This node is a base of any objects in DOPs that can attach various types of data. Next, a geometry to be simulated is imported using SOP Geometry node. Then, velocity, mass and collision field are initialised and attached to the object. Particle Fluid Visualization node is connected to SOP Geometry node to visualise values stored in the particles and Field Visualization nodes are attached to Field nodes to visualise each field. This initialised object is connected to the first input of Multiple Solver node.

The first step in MPM is to transfer particle properties to the fields.

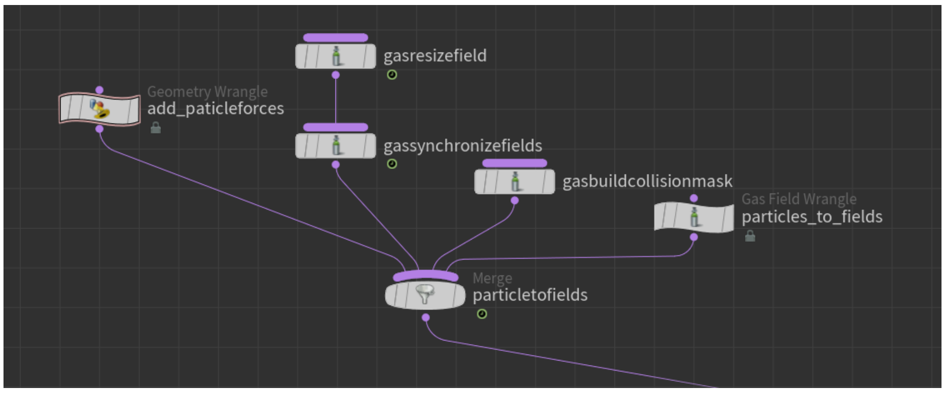


Figure 3: Particles to fields.

Before transferring the particle properties, the particle velocities are updated with pre-defined forces using Geometry Wrangle. Also, the existing fields are resized based on the simulation object with Gas Resize Field node and the positions and sizes are matched with Gas Synchronize Fields node based on the velocity field. This additional synchronising after resizing is important as the fields can drift away from each other during the simulation. Here, Gas Build Collision Mask node is used to generate a signed distance field which is negative inside and positive outside colliding objects. This collision field will be used for grid-based collision handling later. Finally, the particle properties are transferred to fields using Gas Field Wrangle node. This node computes Equation 2 using the product of one-dimensional cubic B-splines as the weighting function. The particle attributes are accessed as a point cloud.

Next, the velocity field is updated.

To avoid the complexity of importing many particle attributes in Gas Field Wrangle node, stress is calculated in Geometry Wrangle and stored as a matrix attribute into particles based on Equation 3. The stress attribute is transferred similar to the particle velocity and mass and used for calculating force and velocity on the field according to Equations 5 and 6.

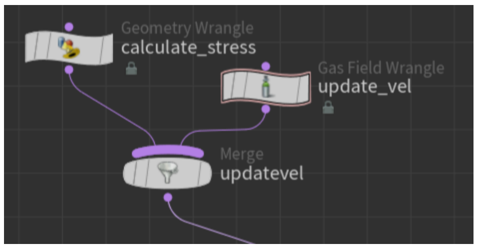


Figure 4: Field velocity update.

After the velocity field update, the collision is handled based on the updated velocities on the field. This is done by multiple nodes as shown in Figure 5.

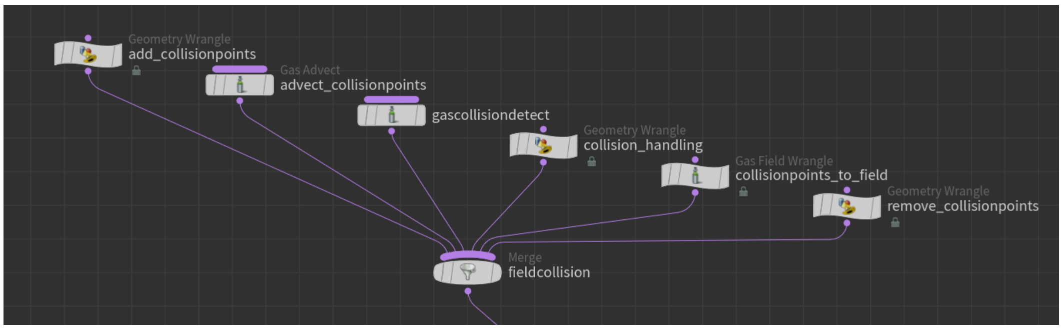


Figure 5: Grid-based collision handling.

There is a microsolver called Gas Collision Detect. This node detects collisions between particles and geometry and adds attributes to the particles that store useful data, such as the number of times the particle has collided and the normal of geometry at the time of the collision. These attributes are useful to handle collisions based on the method described in Section 3.1.4. To use this microsolver, the nodes on the fields must be represented as particles. In the first Geometry Wrangle node, points are created at the position of the nodes and velocities on the nodes are transferred to the points. The indices of the nodes are also stored on the points so that the updated velocities can be transferred back to the correct nodes. For efficiency, the collision field that was created earlier is used to determine which nodes collided with the collision objects, and points are created only at the collided nodes. Next, the created points are advected by the velocities transferred from the field using Gas Advect microsolver and advected points are used for Gas Collision Detect microsolver. After that, the collision is handled with Equation 11 using the normal attribute provided by the Gas Collision Detect microsolver. The updated velocities on the points are transferred to the nodes by comparing the index stored in the points and the index of the nodes. Finally, remove the points created for the collision handling.

Deformation gradients are updated in a Geometry Wrangle node following the equations in Section 3.1.5. The following code snippet is used when calculating the particle velocity gradient (Equation 12) to loop through only the closest nodes for efficiency.

```
index = volumepostoindex("op:../:simgeo/mass","mass",@P);
for (float i=index.x-2; i<=index.x+2; i++)
{
    for (float j=index.y-2; j<=index.y+2; j++)
    {
        pos = volumeindextopos("op:../:simgeo/mass","mass",set(i,j,0));
        vel = volumeindexv("op:../:simgeo/vel","vel",set(i,j,0));
    }
}
```

First, volumepostoindex function is used to get the index of the closest node from the particle. Then, loop through 5x5 nodes around the particle. The position of each node is converted from the index using volumeindextopos function and the velocity stored at the node is retrieved using volumeindexv function.

The next step is to transfer the updated field velocities to particles and move the particles. This step can be done very simply with a Gas Advect microsolver as this node can update particle velocities and positions based on a velocity field.

The final step is to handle collisions based on particle velocities. This step was implemented the same as the grid-based collision handling using the geometry points as collision particles.

4.1.2 Crack Handling

The first step to expand the above implementation for MPM to handle crack is to add multiple velocity fields.

First, additional velocity and mass fields for above and below a crack are initialised similarly to the original ones as shown in Figure 6. Here, an initial crack is imported as a SOP Geometry. The additional fields are resized and synchronised with the other fields.

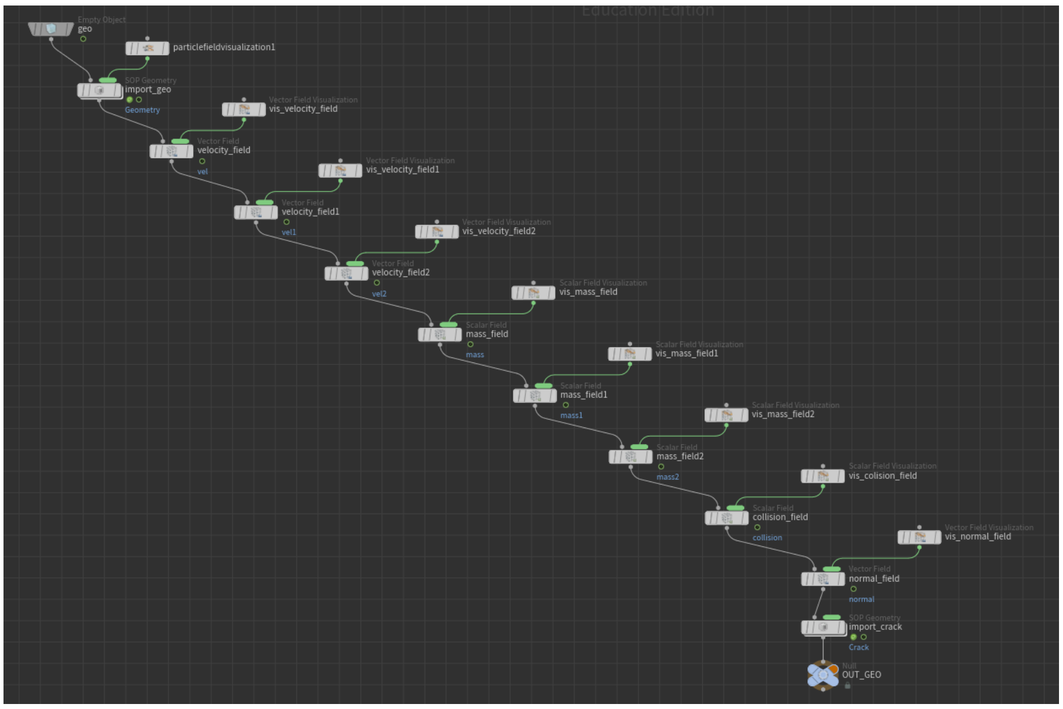


Figure 6: Updated simulation object setup for crack handling.

When transferring particle attributes to the fields, the appropriate field for each combination of particle and node must be determined using the line-crossing algorithm. The following is a part of the code added to the Gas Field Wrangle node in Figure 3.

```
int handle = pcopen(0, "P", v@P, (3 * divsize), 1000000);
while (pciterate(handle)) {
    pcimport(handle, "P", pos);
    if(checkIntersect(@OpInput2,v@P, pos))
    {
        int crackpoints[] = primpoints(@OpInput2,0);
        for(int i=0; i<len(crackpoints)-1; i++)
        {
            pos1 = point(@OpInput2,"P",crackpoints[i]);
            pos2 = point(@OpInput2,"P",crackpoints[i+1]);
            field = getField(pos,v@P,pos1,pos2);
            if(field != 0)
            {
                break;
            }
        }
    }
}
```

The first step in the algorithm is to check if the rectangle defined by the particle and the node intersects with any crack segments. A custom function called checkIntersect was implemented to find an intersection between the edges of the rectangle and the crack geometry using VEX intersect function. If an intersect is found, loop through crack segments and determine the field using getField custom function. This function calculates areas of triangles as Equation 1 and returns the field number according to the results. After determining the appropriate field, particle velocity and mass are transferred to the field. The same functions are used for calculating the particle velocity gradients when updating deformation gradients (Equation 12).

Another update for crack handling is to add Crack Surface Contact Handling. Gas Field node that implements the method described in Section 3.1.3 is added before and after the grid-based collision handling nodes.

As a result of having multiple velocity fields, the particles cannot be simply advected by a velocity field using Gas Advect node. Therefore, Geometry Wrangle node is added before advecting particles to manually transfer field velocities to particles using the line-crossing algorithm as implemented for transferring particle velocities to fields. Here, another Geometry Wrangle is added to handle fixed particles. After transferring velocities, the Gas Advect node is used as before. The Gas Advect node allows advecting particles based on particle velocities instead of a velocity field.

Finally, crack particles are advected similar to the simulation object particles.

The crack particles store crack surface displacements. The displacements are computed using Gas Advect wrangles. Instead of advecting the particle positions, displacement attributes `xa` for the surface above the crack and `xb` for the surface below the crack are advected by the appropriate velocity fields. For a visualisation purpose, points are

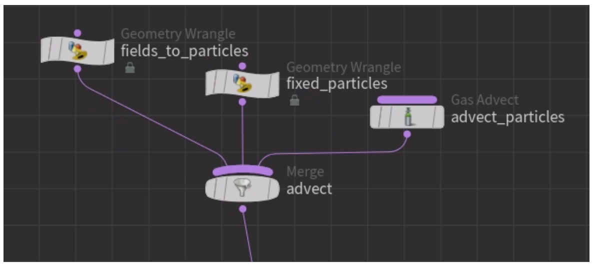


Figure 7: Updated particle advection.

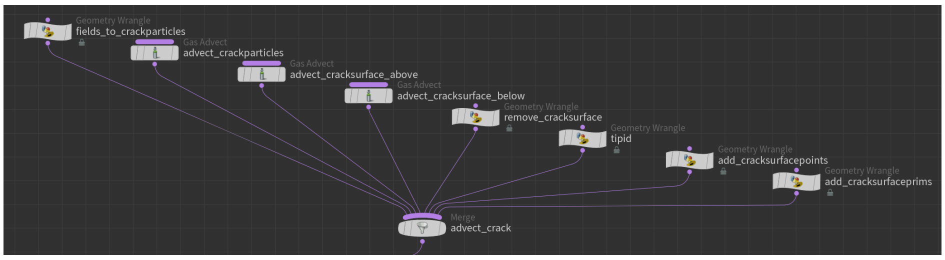


Figure 8: Crack particle advection.

added at `xa` and `xb` and connected as polylines. The crack surface polylines from the previous timestep are removed before creating new ones.

4.1.3 Crack Propagation

To help generate a J-integral contour around the crack tip, circle geometry is created outside the DOP network and imported using SOP Geometry similar to the initial crack geometry.

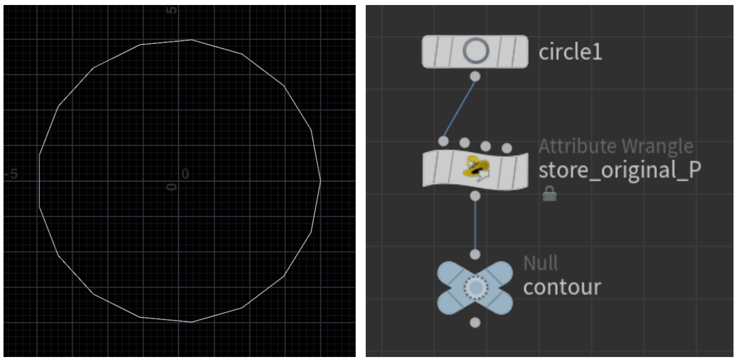


Figure 9: Circle Geometry for generating a contour and its setup.

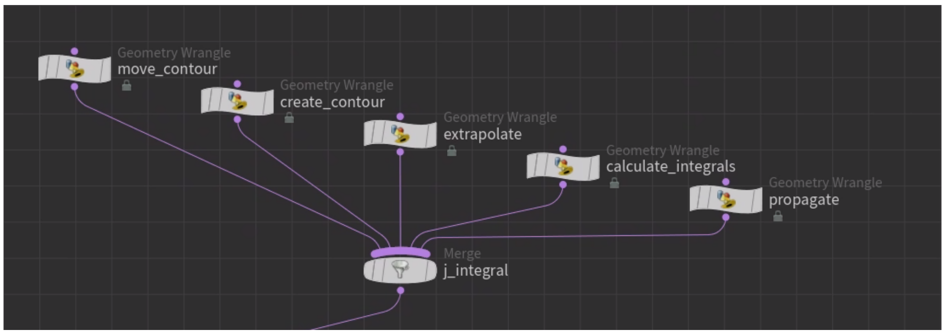


Figure 10: Crack Propagation.

The first Geometry Wrangle node in Figure 10 moves the imported circle geometry to the crack tip position and the next node generates a J-integral contour as described in Section 3.1.8. Initially, the VEX intersect function was used to find the intersection between the circle geometry and the crack surface polylines but returned intersection points were not accurate enough for the simulation. Therefore, intersections are found by looping through segments of crack surface polylines and circle primitive and using a custom function that returns an intersection point between two lines. After finding the intersections, a J-integral contour is generated with the following codes:

```
float angle = (2*$PI -
↪   acos(dot(normalize(posa-tippos),normalize(posb-tippos))))/16;
float radius = divsize*2;
float currentangle = acos(dot(normalize(posb-tippos),set(1,0,0)));
if(cross(set(1,0,0),posb-tippos).z<0)
{
    currentangle *= -1;
}
int prim_num = addprim(0,"polyline");
for (int i=0; i<17; i++)
{
    float x = tippos.x+radius*cos(currentangle);
    float y = tippos.y+radius*sin(currentangle);
    int point_num = addpoint(0, set(x,y,0));
    setpointattrib(0,"normal",point_num,normalize(set(x,y,0)-tippos));
    setpointgroup(0,"contour",point_num,1);
    addvertex(0,prim_num,point_num);
    currentangle += angle;
}
```

where posa and posb is the intersection points for surface above and below and tippos is the position of the crack tip. This code first computes the angle from the intersection point with the surface below to the intersection point with the surface above and divides by 16 to get the angle of 16 segments. Then, add 17 points at the positions calculated with the computed angle. The points are grouped and connected as a polyline.

The next Geometry Wrangle node transfers nearby particle attributes to the J-integral contour points. This is done similar to when transferring particle attributes to fields.

Using the transferred values, the components of the integrand at each point of the contour are computed with Equation 28 and stored as attributes on the point in the next Geometry Wrangle node.

The final Geometry Wrangle calculates J-integral and propagates the crack. It loops through the contour points and calculates the components of J-integral with Equation 27. After calculating J-integral from the components, it is converted into stress intensity factors with Equation 31 or 32. VEX distance function is used for calculating the length of each contour segment and the magnitude of the crack opening displacement. Then, the angle of the crack propagation is calculated with Equation 33 and checks if it meets the criterion with Equation 34. If the criterion is met, add a point to the crack geometry half a cell size away from the crack tip in the computed angle.

When testing the implementation, the crack direction changes unexpectedly in some situations. It was found that the problem is caused by a truncation error in MPM calculation making a small amount of shearing displacements. This results in a large mixed-mode ratio. The shearing displacement is rounded to 4 decimal numbers to avoid this problem at first. However, it was discovered that cracks propagate differently depending on how many decimal numbers to keep after several testings. Therefore, the number of decimal numbers to keep is promoted as a parameter for users to control freely later.

Later during the development, a function to follow a pre-defined crack path was added to allow users to design a crack pattern manually. This is handled as follows. First, the pre-defined path is imported as SOP Geometry during the simulation object setup. Instead of computing the propagation angle with Equation 33, the angle from the current crack tip to the first point in the pre-defined path is used as θ_c. The angle is used to check the criterion as before and a point is added at the first point in the pre-defined path. Finally, the first point in the pre-defined path is removed so the next point becomes the first one, which will be used for the next timestep.

4.2 Houdini Digital Asset

The implemented solver is wrapped into a digital asset so that it can be reused for different simulations. The digital asset includes nodes for geometry setup as can be seen in Figure 11.

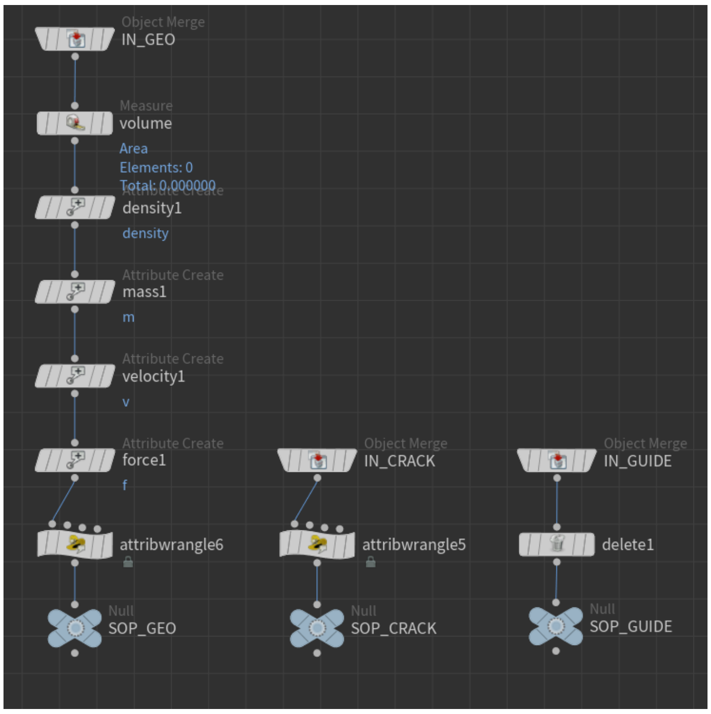


Figure 11: Geometry setup.

The attributes of the simulation geometry are initialised with Attribute Create node and Attribute Wrangle nodes. The existing attributes are used if these are already set by a user. As the crack particles store crack surface displacements, point attributes to store the displacements and velocities to advect the displacements are initialised with the Attribute Wrangle node. Pre-defined crack path (Guide Geometry) is expected to include the initial crack so the first point is removed before being imported into the

DOP network. Also, the path is resampled with a length of half a cell size to propagate half a cell size at a time.

The digital asset also includes nodes for post-processing the simulation as shown in Figure 12.

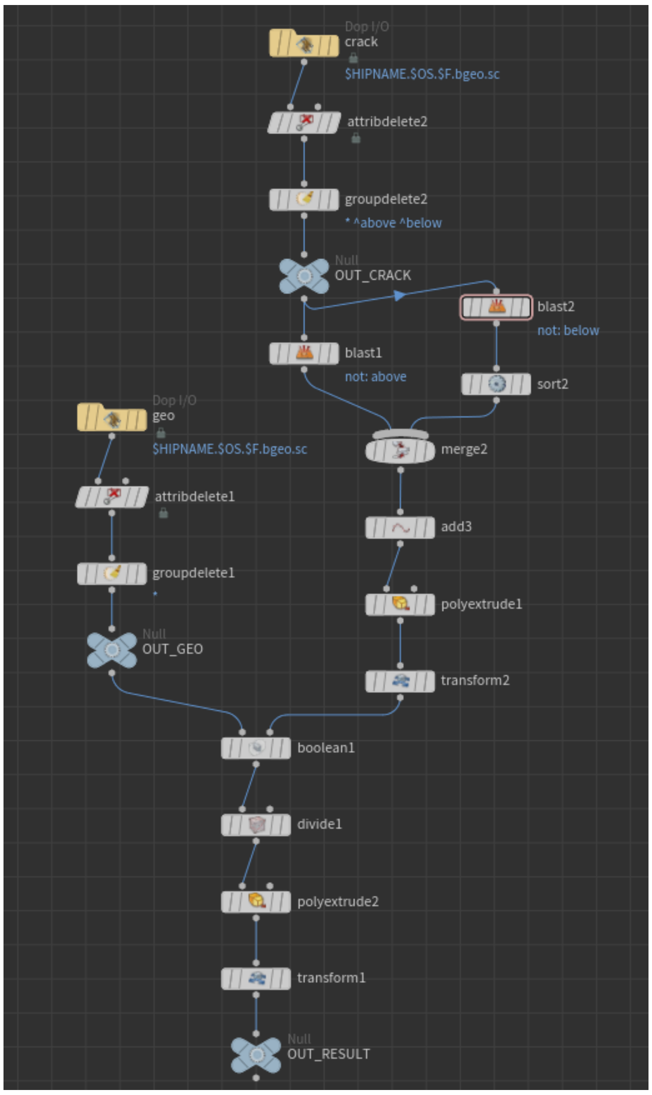


Figure 12: Post-process.

The post-processing import simulated object and crack separately. The crack surface above and below are retrieved separately and connected as a polygon with the space between the surfaces filled. Then, Boolean node subtracts the connected crack polygon from the simulated object to create a cracked object. Poly Extrude node is added at the end to allow users to control the thickness of the object.

The post-processed output as well as the original simulated crack, geometry and collision objects are merged with Object Merge nodes and connected to Switch node to allow a user to decide what to display as can be seen in Figure 13.

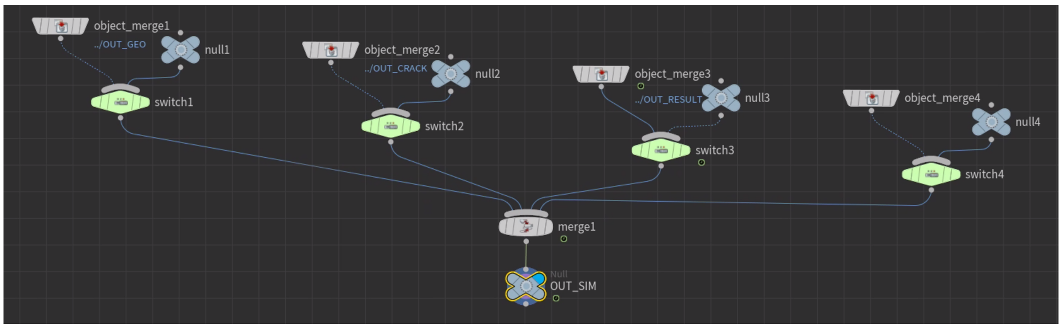
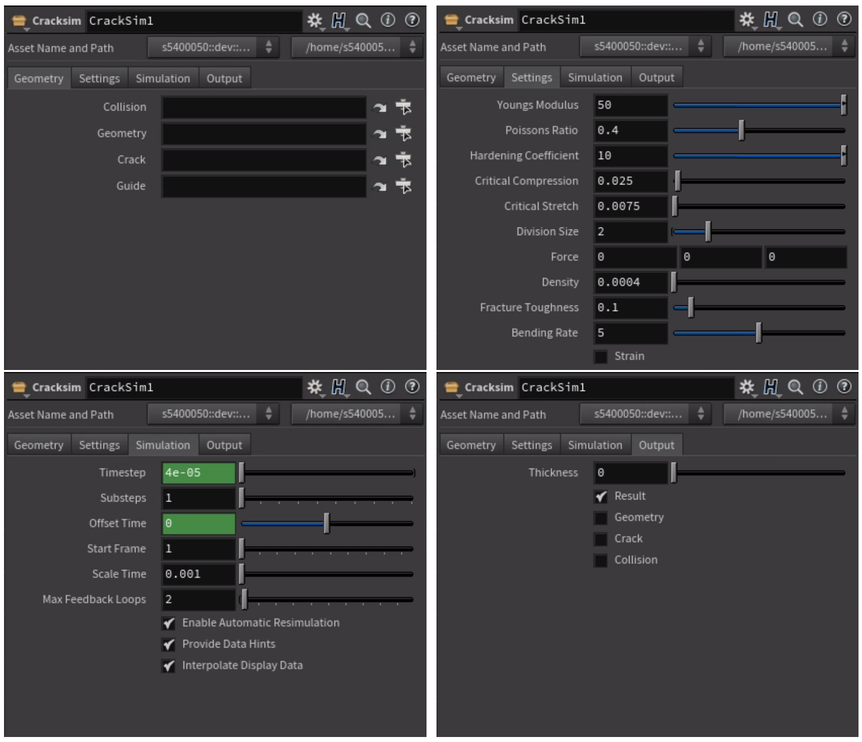


Figure 13: Final output node.

A user interface for the digital asset is created to allow more user controls as shown in Figure 14.



Collision	
Geometry	
Crack	
Guide	

Youngs Modulus	50		
Poissons Ratio	0.4		
Hardening Coefficient	10		
Critical Compression	0.025		
Critical Stretch	0.0075		
Division Size	2		
Force	0	0	0
Density	0.0004		
Fracture Toughness	0.1		
Bending Rate	5		

☐ Strain

Timestep	4e-05
Substeps	1
Offset Time	0
Start Frame	1
Scale Time	0.001
Max Feedback Loops	2

✓ Enable Automatic Resimulation
✓ Provide Data Hints
✓ Interpolate Display Data

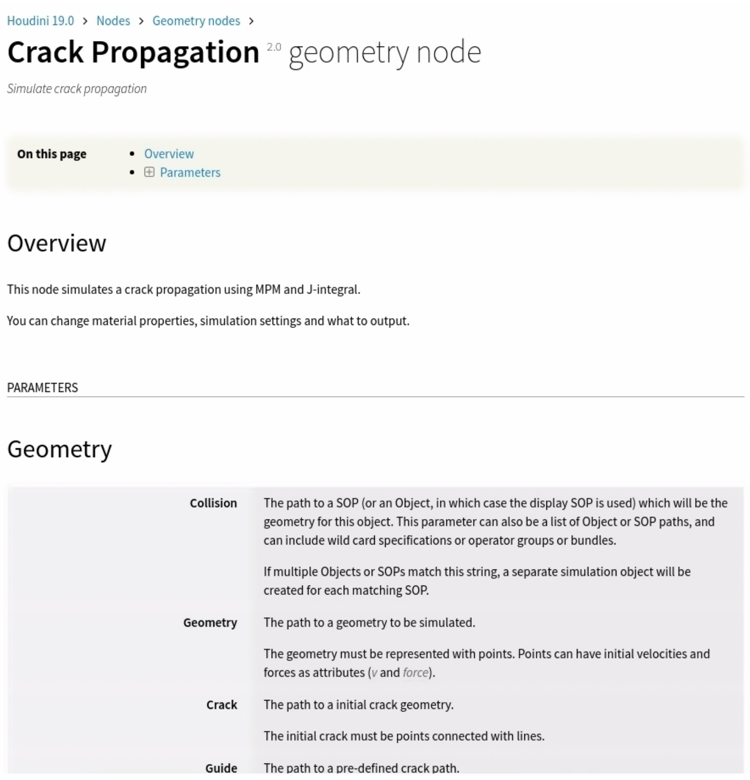
Thickness	0

✓ Result
☐ Geometry
☐ Crack
☐ Collision

Figure 14: User interface.

The Geometry tab is to import geometry to be simulated, collision object, initial crack and pre-defined path. The digital asset can be used as a simple MPM deformation solver if the crack field is empty. The Settings tab allows users to change parameters used for the simulation such as properties of the simulation object material, global force applied to the simulation and bending rate to change how easy the crack direction changes. The Simulation tab has parameters for simulation settings. It is the same as the Simulation tab on DOP network. The Output tab lets users choose what type of output to be displayed. The user can change the thickness of the simulated object here.

After the completion of the asset, a help card is added to guide users on how to use this asset as shown in Figure 15. The documentation can be found from the Help button at the top right corner of the parameter editor of the asset node.



Houdini 19.0 > Nodes > Geometry nodes >

Crack Propagation 2.0 geometry node

Simulate crack propagation

On this page
- Overview
- ⊞ Parameters

Overview

This node simulates a crack propagation using MPM and J-integral.

You can change material properties, simulation settings and what to output.

PARAMETERS

Geometry

Collision	The path to a SOP (or an Object, in which case the display SOP is used) which will be the geometry for this object. This parameter can also be a list of Object or SOP paths, and can include wild card specifications or operator groups or bundles. If multiple Objects or SOPs match this string, a separate simulation object will be created for each matching SOP.
Geometry	The path to a geometry to be simulated. The geometry must be represented with points. Points can have initial velocities and forces as attributes (*v* and *force*).
Crack	The path to a initial crack geometry. The initial crack must be points connected with lines.
Guide	The path to a pre-defined crack path.

Figure 15: Preview of the asset help card.

4.3 Testing

Throughout the development, the solver was tested many times with various approaches. After each step was implemented, the simulation was run with small numbers of particles and nodes to check if the implementation works. When testing the simulation, the particles and fields are visualised with the visualisation nodes to check if the simulation is working correctly as can be seen in Figure 16.

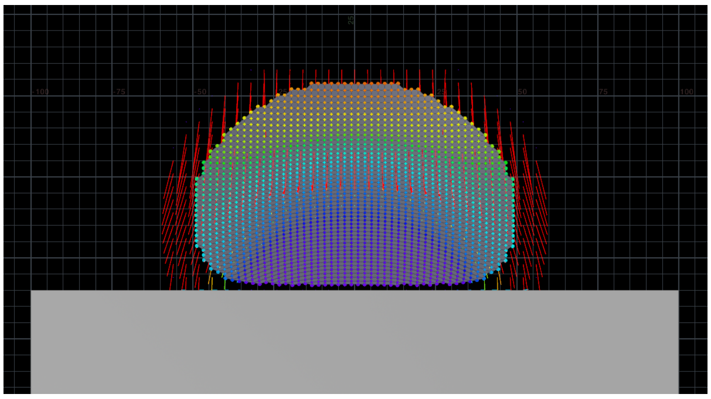


Figure 16: Visualising particle velocities with spheres and velocity fields with vectors in infra-red mode.

When testing the J-integral contour, the contour and integrands at each point were visualised as vectors as shown in Figure 17 to see if the calculation is correct.

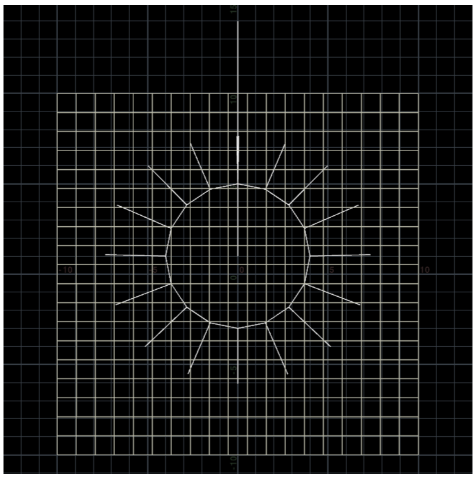


Figure 17: Visualisation of J-integral contour.

A snippet of VEX codes in wrangle nodes was tested with VEX printf function, which prints out values on terminals to see if the outputs are as expected. Another way to check the values used for the simulation is to store these as variables and check these on Geometry Spreadsheet.

After each step of the implementation, the solver was tested with the same settings. This test drops a circle object with a radius of 50m from 5m above the collision surface. The snow material shown in Table 1 was used for the tests. The particle separation and grid cell size were set to 1m and 2m respectively and the timestep was set to 0.004s. The screenshots of the test results can be found in Appendix A.

5 Result

The completed digital asset was tested in two setups, one is a pull test and another one is a drop test. The material parameters used for the tests can be found in Table 1.

Material	Young's modulus E (Pa)	Poisson's ratio ν	Density ρ (kg/m^3)	Fracture Toughness K_c $(MPa\sqrt{m})$
Snow	$140K$	0.2	400	10
PMMA	$2.94G$	0.3	1190	$1.2M$
Glass	$70G$	0.22	2500	$0.75M$
Jello	50	0.4	2	10

Table 1: Material parameters used for the tests. Snow material is based on the paper by Stomakhin et al. (2013), PMMA is based on the example provided by Guo and Nairn (2017) and Jello is based on the paper by Wolper et al. (2019).

The material of the simulation object is set as PMMA and the particle separation of the object is set to 1m unless mentioned otherwise. The grid cell size is set to double the particle separation. The simulation was tested on a large scale to avoid truncation error as Houdini cannot handle large floating point numbers.

5.1 Pull Test

This test uses a 20mx20m square object with forces applied to the particles on right and left sides to pull the object. The total force applied on both sides is $F = 40MN$.

First, simulations with different initial cracks and a guide path were tested as shown in Figure 18.

The crack propagates differently depending on the initial crack as can be seen by comparing the straight initial crack on the left and the rotated initial crack in the middle of Figure 18. When a guide path is applied to the simulation, the crack propagation successfully followed the path as can be seen on the right in Figure 18.

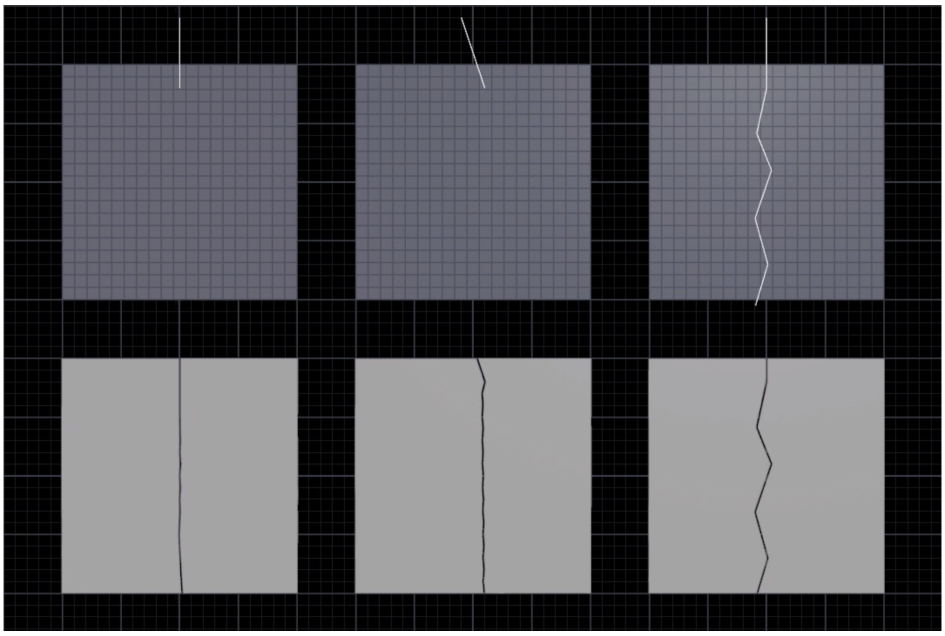


Figure 18: Initial crack on the top and pull test result on the bottom.

Next, the simulation is tested with varying parameters. As shown in the figures below, the crack propagated differently depending on the settings.

First, the simulation was tested with different timesteps.

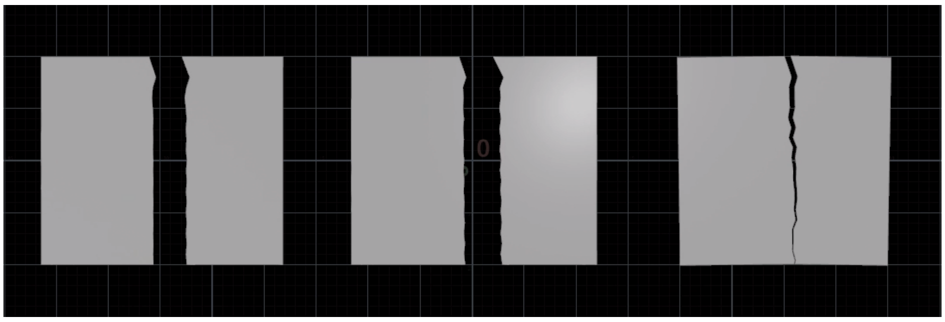


Figure 19: Pull test result with different timesteps (0.00004s on the left, 0.0001s in the middle and 0.0004s on the right).

As can be seen in Figure 19, the crack propagated more smoothly with smaller timesteps. It can be said that there are more errors when the timestep is large so the crack direction changes more.

Next, crack propagation was tested with various bending rates.

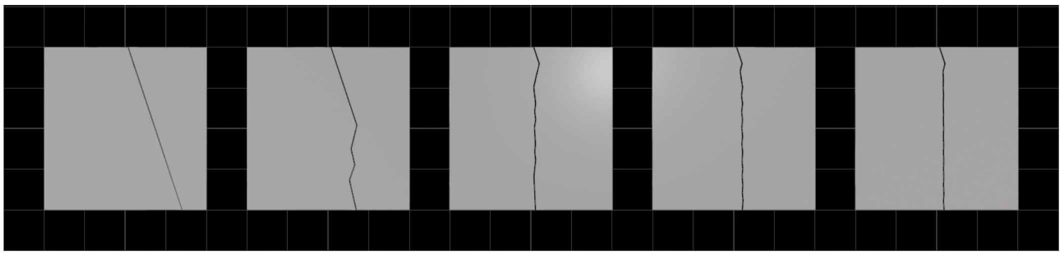


Figure 20: Pull test result with different bending rate (2,3,4,5,6 from left to right).

Figure 20 shows that the direction of the crack propagation changes more with a higher bending rate. It can be useful for artists to control crack patterns more freely.

The simulation was also tested with different resolutions.

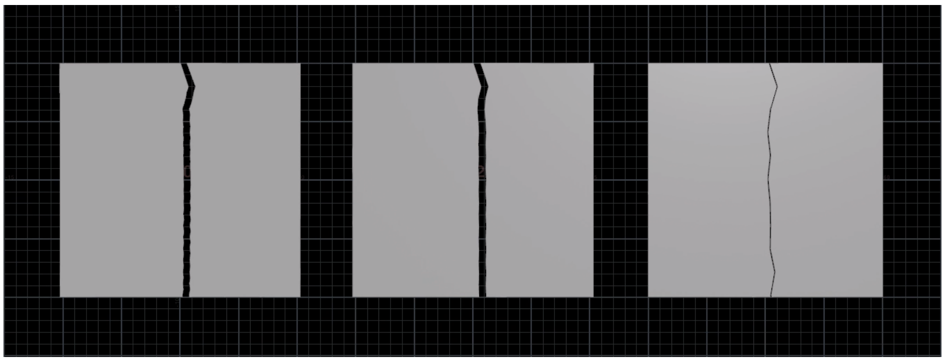


Figure 21: Pull test result with different particle separation (0.5m on the left, 1m in the middle and 2m on the right).

Higher resolution gives a more detailed crack pattern as shown in Figure 21. However, a higher resolution requires more particles and nodes to be computed and a smaller timestep must be used so it takes longer to simulate.

Finally, different materials were simulated as shown in Figure 22. Glass material is used for this test.

As shown in Figure 22, the crack pattern changes depending on the materials. Also, the crack on PMMA propagated faster than the crack on the glass. It can be said that the simulation reflects the real life as softer materials fracture easier.

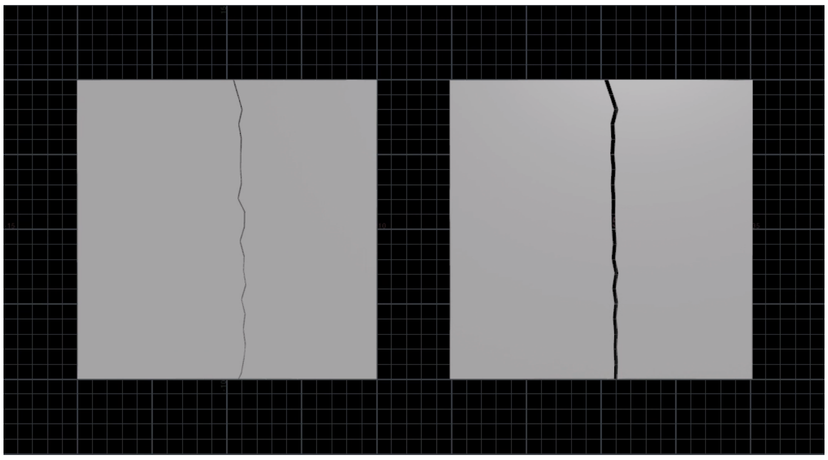


Figure 22: Pull test result with different materials (glass on the left and PMMA on the right).

5.2 Drop Test

This test uses a circle object with a radius of 50m and drops the object from 5m above the surface similar to the test for each implementation step described in Section 4.3.

First, different materials were tested.

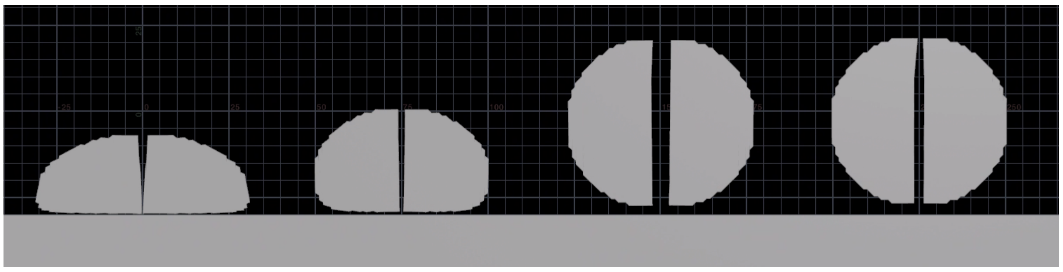


Figure 23: Drop test result with different materials (jello, snow, PMMA and glass from left to right).

The crack propagated when the object collides with the collision object surface. As can be seen in Figure 23, softer materials deform more. Also, the harder the material is, the smaller timestep must be used as the velocities get large when the simulation object collides with the collision object. The timestep of 0.004s was used for the jello material and 0.00004s was used for the glass material.

Next, different resolutions were tested.

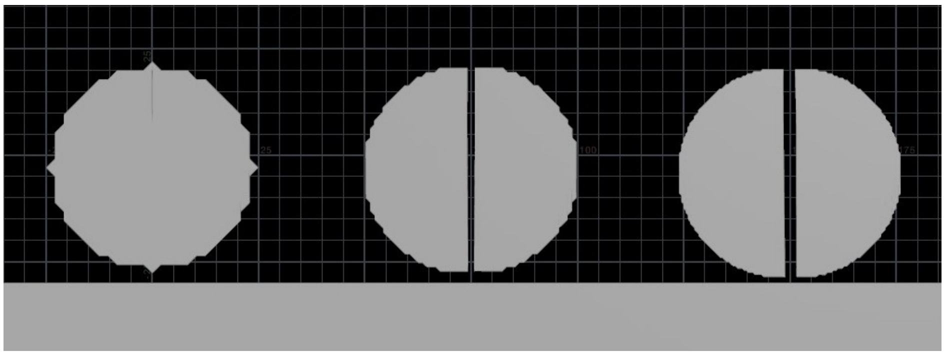


Figure 24: Drop test result with different particle separation (2m on the left, 1m in the middle and 0.5m on the right).

As shown in Figure 24, the simulation object with particle separation of 2m did not propagate. It seems the stress generated by the collision with the collision object surface was spread throughout the object before the stress at the crack tip reached the critical value as the timestep is too large. Similarly to the pull test, the higher the resolution, the smaller timestep must be used. It took almost 3 hours to simulate the drop test with a particle separation of 0.5m for 45 frames with substeps of 800.

6 Conclusions

The pull test with glass material explained in Section 5 was rendered as shown in Figure 25.

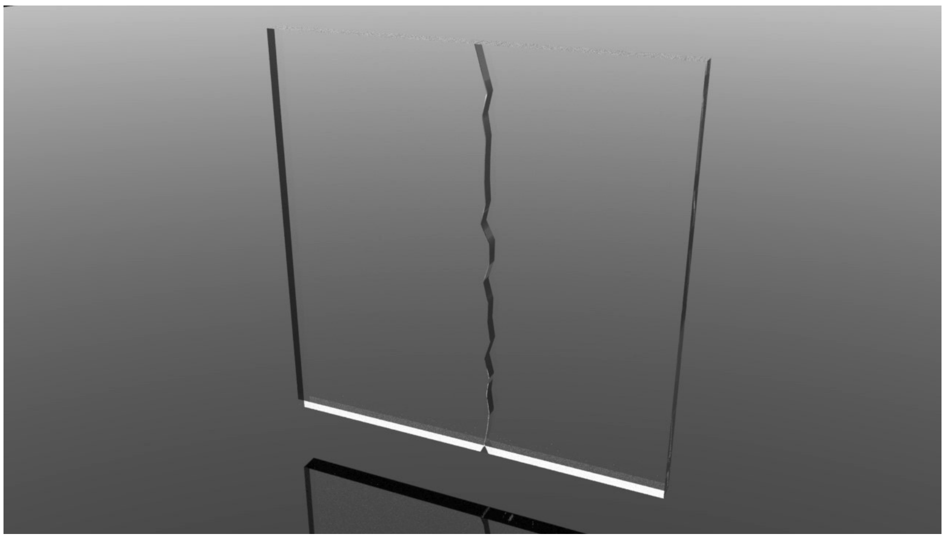


Figure 25: Rendered crack propagation result.

Overall, the project can be considered successful due to the fact that the initial aim to develop a digital asset for crack propagation simulation with various user controls is achieved. The developed asset can simulate deformation with or without crack successfully. The simulation using the asset generates various patterns with different settings as described in Section 5. This asset allows to simulate different types of materials by changing the parameters. There are other properties that the user can control such as timestep, division size, external force and bending rate. Also, initial forces and velocities can be added to the selected points on the geometry before importing the geometry into the digital asset. This asset let the user use a guide crack path, which gives the user more control over the crack patterns.

Although the overall development was successful, there are some limitations with the asset. First of all, this asset cannot be used for very small scale simulations as Houdini cannot handle large floating numbers. Small numerical differences make large errors in some computations in the method used for this project. For example, the intersection point cannot be very accurate in a small scale simulation so the J-integral contour cannot be created as expected. Also, this asset does not automatically add substeps so it is important to use appropriate timestep. If the timestep is too big, the particles move unexpectedly or Houdini can crash in the worst case as the values in the computation become too large. Another limitation is that the asset only handles a single crack and it can only propagate in one direction.

Many improvements can be made to the developed asset. One of the improvements is to reduce the limitations stated above. At each timestep, the solver should check the velocities and add substeps if the velocities are too big to avoid the particles to move unexpectedly. Also, the crack should be able to propagate on both sides by setting both ends of the crack line as crack tips and computing the crack tip properties at both tips. Handling multiple materials in a single object would allow a user to simulate more types of objects, such as an object that has different materials for the inner part and the outer part. Adding a paint tool to design the strength of the object would also be a good improvement. Another improvement can be extending the method to handle multiple cracks as only one crack in a simulation is not very useful for fracture animations. Enabling branching would also be an interesting improvement. Also, this project was worked in 2D but the deformation simulation with MPM can be extended to 3D with the same algorithm and the same crack propagation algorithm can be applied for simulating surface cracks on 3D objects. This can be further extended to have 3D crack propagation using a surface as a crack instead of a line.

References

Bao, Z., Hong, J.-M., Teran, J. and Fedkiw, R. 2007. Fracturing rigid materials, *IEEE Transactions on Visualization and Computer Graphics* **13**(2): 370–378.

Belytschko, T. and Black, T. 1999. Elastic crack growth in finite elements with minimal remeshing, *International journal for numerical methods in engineering* **45**(5): 601–620.

Chitalu, F. M., Miao, Q., Subr, K. and Komura, T. 2020. Displacement-correlated xfem for simulating brittle fracture, *Computer Graphics Forum*, Vol. 39, Wiley Online Library, pp. 569–583.

Daux, C., Moës, N., Dolbow, J., Sukumar, N. and Belytschko, T. 2000. Arbitrary branched and intersecting cracks with the extended finite element method, *International journal for numerical methods in engineering* **48**(12): 1741–1760.

Daviet, G. and Bertails-Descoubes, F. 2016. A semi-implicit material point method for the continuum simulation of granular materials, *ACM Transactions on Graphics (TOG)* **35**(4): 1–13.

Desbenoit, B., Galin, E. and Akkouche, S. 2005. Modeling cracks and fractures, *The Visual Computer* **21**(8): 717–726.

Fan, L., Chitalu, F. M. and Komura, T. 2022. Simulating brittle fracture with material points, *ACM Transactions on Graphics (TOG)* **41**(5): 1–20.

Glondu, L., Marchal, M. and Dumont, G. 2012. Real-time simulation of brittle fracture using modal analysis, *IEEE Transactions on Visualization and Computer Graphics* **19**(2): 201–209.

Glondu, L., Muguercia, L., Marchal, M., Bosch, C., Rushmeier, H., Dumont, G. and Drettakis, G. 2012. Example-based fractured appearance, *Computer Graphics Forum*, Wiley Online Library, pp. 1547–1556.

Guo, Y. and Nairn, J. 2006. Three-dimensional dynamic fracture analysis using the material point method, *Computer Modeling in Engineering and Sciences* **16**(3): 141.

Guo, Y. and Nairn, J. 2017. Simulation of dynamic 3d crack propagation within the material point method, *Comput Model Eng Sci* **113**(4): 389–410.

Guo, Y. and Nairn, J. A. 2004. Calculation of j-integral and stress intensity factors using the material point method, *Computer Modeling in Engineering and Sciences* **6**: 295–308.

Hahn, D. and Wojtan, C. 2015. High-resolution brittle fracture simulation with boundary elements, *ACM Transactions on Graphics (TOG)* **34**(4): 1–12.

Hahn, D. and Wojtan, C. 2016. Fast approximations for boundary element based brittle fracture simulation, *ACM Transactions on Graphics (TOG)* **35**(4): 1–11.

Hellrung, J., Selle, A., Shek, A., Sifakis, E. and Teran, J. 2009. Geometric fracture modeling in bolt, *SIGGRAPH 2009: Talks*, pp. 1–1.

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): 1–14.

Iben, H. N. and O'brien, J. F. 2009. Generating surface crack patterns, *Graphical Models* **71**(6): 198–208.

Jiang, C., Gast, T. and Teran, J. 2017. Anisotropic elastoplasticity for cloth, knit and hair frictional contact, *ACM Transactions on Graphics (TOG)* **36**(4): 1–14.

Klár, G., Gast, T., Pradhana, A., Fu, C., Schroeder, C., Jiang, C. and Teran, J. 2016. Drucker-prager elastoplasticity for sand animation, *ACM Transactions on Graphics (TOG)* **35**(4): 1–12.

Koschier, D., Lipponer, S. and Bender, J. 2014. Adaptive tetrahedral meshes for brittle fracture simulation., *Symposium on Computer Animation*, pp. 57–66.

Martinet, A., Galin, E., Desbenoit, B. and Akkouche, S. 2004. Procedural modeling of cracks and fractures, *Proceedings Shape Modeling Applications, 2004.*, IEEE, pp. 346–349.

Moës, N., Dolbow, J. and Belytschko, T. 1999. A finite element method for crack growth without remeshing, *International journal for numerical methods in engineering* **46**(1): 131–150.

Muguercia, L., Bosch, C. and Patow, G. 2014. Fracture modeling in computer graphics, *Computers & graphics* **45**: 86–100.

Müller, M., Chentanez, N. and Kim, T.-Y. 2013. Real time dynamic fracture with volumetric approximate convex decompositions, *ACM Transactions on Graphics (TOG)* **32**(4): 1–10.

Müller, M. and Gross, M. H. 2004. Interactive virtual materials., *Graphics interface*, Vol. 2004, pp. 239–246.

Müller, M., McMillan, L., Dorsey, J. and Jagnow, R. 2001. Real-time simulation of deformation and fracture of stiff materials, *Computer Animation and Simulation 2001*, Springer, pp. 113–124.

Nairn, J. A. 2003. Material point method calculations with explicit cracks, *Computer Modeling in Engineering and Sciences* **4**(6): 649–664.

O'Brien, J. F. and Hodgins, J. K. 1999. Graphical modeling and animation of brittle fracture, *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 137–146.

Parker, E. G. and O'Brien, J. F. 2009. Real-time deformation and fracture in a game environment, *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 165–175.

Raghavachary, S. 2002. Fracture generation on polygonal meshes using voronoi polygons, *ACM SIGGRAPH 2002 conference abstracts and applications*, pp. 187–187.

Ram, D., Gast, T., Jiang, C., Schroeder, C., Stomakhin, A., Teran, J. and Kavehpour, P. 2015. A material point method for viscoelastic fluids, foams and sponges, *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 157–163.

SideFX 2022a. Community news feed | sidefx [online]. Available from: `https://www.sidefx.com/community/section/stories/` [Accessed: 14 September 2022].

SideFX 2022b. Houdini - 3d modeling, animation, vfx, look development, lighting and rendering | sidefx [online]. Available from: `https://www.sidefx.com/` [Accessed: 08 September 2022].

SideFX 2022c. Houdini 19.5 [online]. Available from: `https://www.sidefx.com/docs/houdini/` [Accessed: 13 September 2022].

SideFX 2022d. How to use the help [online]. Available from: `https://www.sidefx.com/docs/houdini/help/` [Accessed: 14 September 2022].

SideFX 2022e. Vex [online]. Available from: `https://www.sidefx.com/docs/houdini/vex/` [Accessed: 14 September 2022].

SideFX 2022f. Vex [online]. Available from: `https://www.sidefx.com/docs/houdini/assets/` [Accessed: 14 September 2022].

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): 1–10.

Su, J., Schroeder, C. and Fedkiw, R. 2009. Energy stability and fracture for frame rate rigid body simulations, *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 155–164.

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): 236–252.

Terzopoulos, D. and Fleischer, K. 1988. Modeling inelastic deformation: viscolelasticity, plasticity, fracture, *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pp. 269–278.

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).

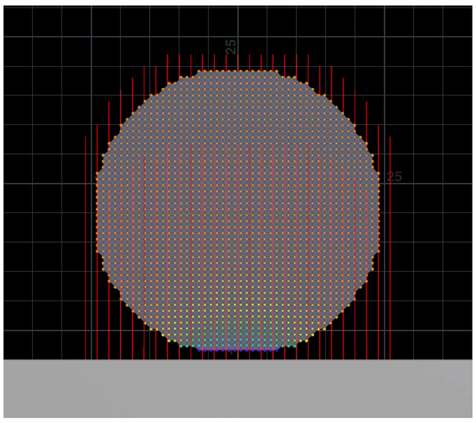
Wolper, J., Fang, Y., Li, M., Lu, J., Gao, M. and Jiang, C. 2019. Cd-mpm: continuum damage material point methods for dynamic fracture animation, *ACM Transactions on Graphics (TOG)* **38**(4): 1–15.

Wretborn, J., Armiento, R. and Museth, K. 2017. Animation of crack propagation by means of an extended multi-body solver for the material point method, *Computers & Graphics* **69**: 131–139.

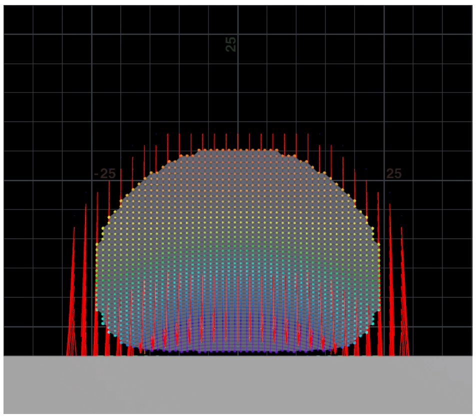
Yue, Y., Smith, B., Batty, C., Zheng, C. and Grinspun, E. 2015. Continuum foam: A material point method for shear-dependent flows, *ACM Transactions on Graphics (TOG)* **34**(5): 1–20.

Zhu, Y., Bridson, R. and Greif, C. 2015. Simulating rigid body fracture with surface meshes., *ACM Trans. Graph.* **34**(4): 150–1.

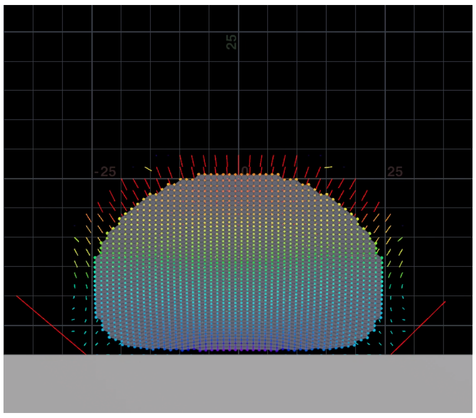
A Test Results



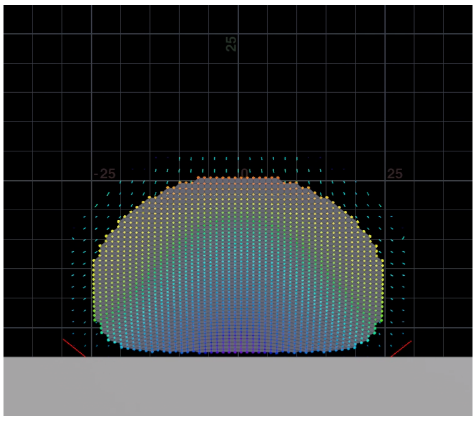
(a) After 1s



(b) After 2s

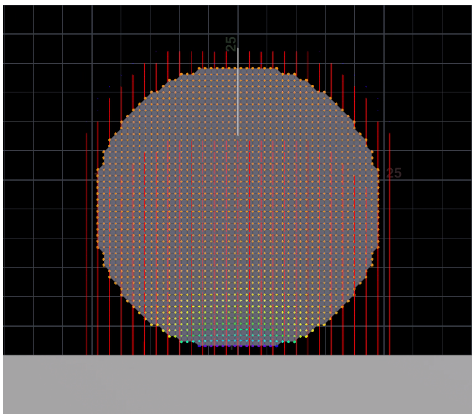


(c) After 3s

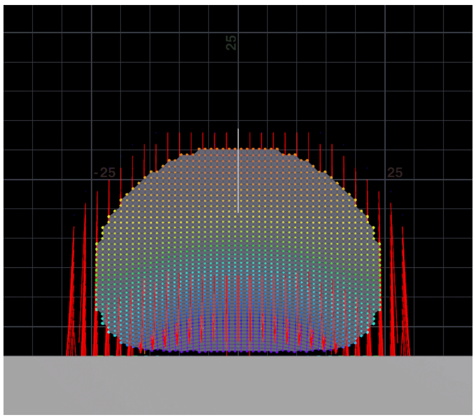


(d) After 4s

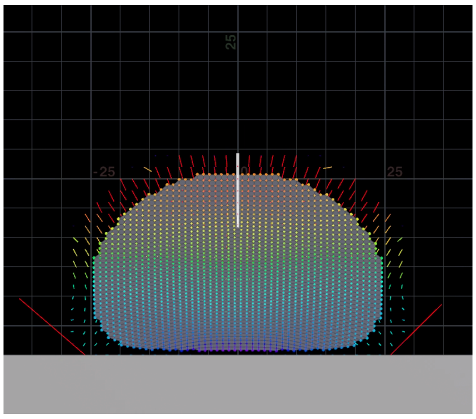
Figure 26: Test result after the implementation for MPM solver.



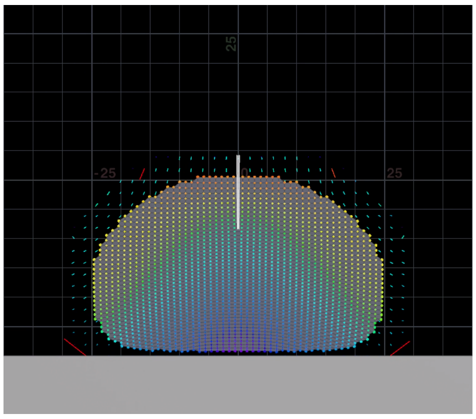
(a) After 1s



(b) After 2s

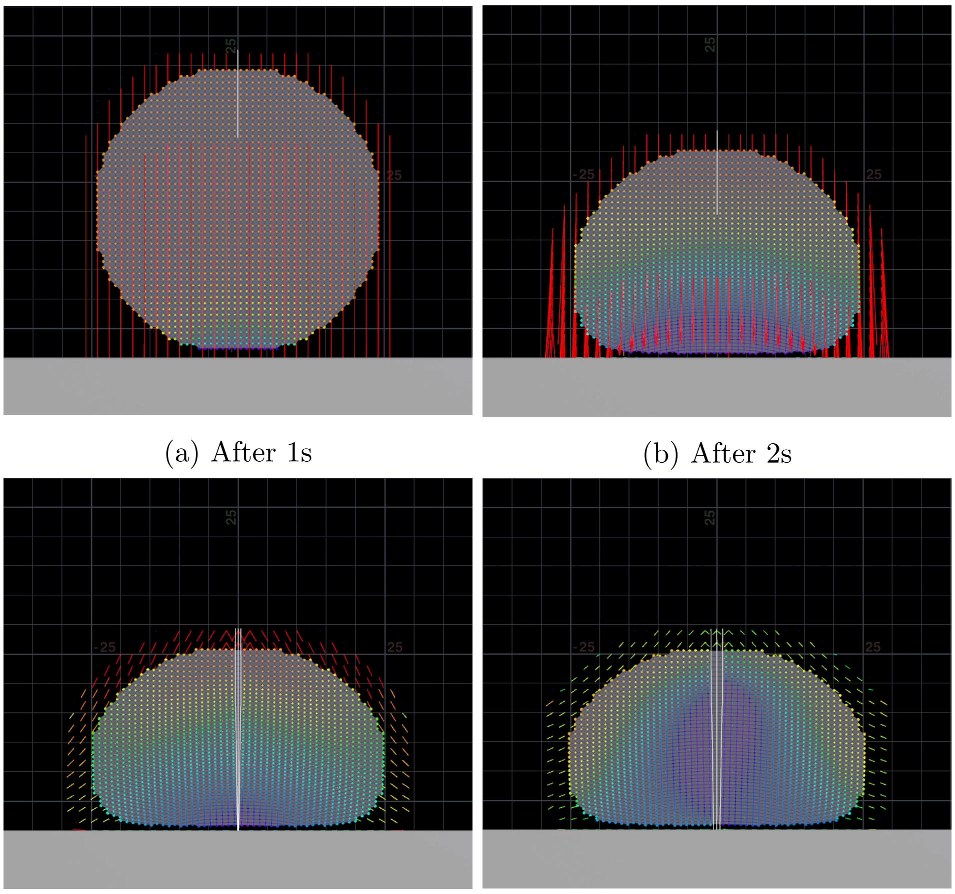


(c) After 3s



(d) After 4s

Figure 27: Test result after extending the solver to handle a crack.



(a) After 1s

(b) After 2s

(c) After 3s

(d) After 4s

Figure 28: Test result after the implementation for crack propagation.