THE UNIVERSITY *of* EDINBURGH

Simulating Fracture in Brittle Objects and Thin Shell Objects for Visual Effects

Linxu Fan

Doctor of Philosophy

Institute of Perception, Action and Behaviour

School of Informatics

University of Edinburgh

2024

Abstract

This doctoral thesis presents a series of studies focused on simulating fractures within various objects. In the context of fracture simulation, it is crucial to address collision detection and contact handling, including self-contact, particularly when employing mesh-based methods. However, these methods necessitate the discretization of the domain using volumetric meshes, as opposed to surface meshes commonly used in computer graphics applications. For the sake of simplicity and efficiency, material point method (MPM) is employed for more straightforward discretization. MPM also facilitates automatic contact due to its hybrid Lagrangian/Eulerian representation. Nevertheless, the continuum assumption of MPM prevents the representation of cracks as discontinuities. Additionally, the intricate behaviors of real-world fractures, such as dynamic bifurcation and merging, require sophisticated explicit geometry processing methods.

To address these challenges, a novel approach is proposed to implicitly simulate arbitrary non-manifold cracks by monitoring the evolution of a scalar phase field. The simulation encompasses two categories of commonly encountered objects: brittle objects and codimensional thin shell objects.

For brittle objects, a combination of elastodynamic continuum mechanical models and rigid-body methods is utilized. The global dynamic motion of fragments is captured through a rigid-body solver. The impacts computed by this solver, including contact forces and durations, are then integrated into the MPM framework to update the phase field. This approach enables the extraction of a non-manifold crack surface from the phase field, facilitating accurate and robust modeling of material fragment volumes to enhance fast and rigid shatter effects. To faithfully replicate the natural patterns of brittle fracture surfaces, fracture details are introduced, incorporating particle damage-time to guide localized perturbations of the crack surface with artistic control.

Regarding thin shell objects, a hybrid Lagrangian/Eulerian continuum shell formulation is introduced to facilitate arbitrary fracturing. The geometry of thin shell objects is described using Non-Uniform Rational Basis-Spline (NURBS) surfaces. The kinematics is simplified using a Kirchhoff-Love shell model, followed by compression and shearing in the normal direction. A coherent crack surface is extracted from the evolving phase field within the co-dimensional manifold. To capture the pronounced discontinuities of interpolated field quantities near the crack, a moving least squares (MLS) approximation is applied. To address complex surface geometries, a novel NURBS

patch coupling method is presented, ensuring arbitrary order continuity across the interface.

In summary, the dissertation addresses the intricate challenges of fracture simulation. It thoroughly investigates brittle object fracture and thin-shell object fracture to achieve generality. The study proposes several innovative approaches to tackle the issues encountered during the simulation process.

Acknowledgements

I would like to thank my supervisor, Taku Komura, for his unwavering support, guidance, and supervision. Additionally, I deeply appreciate the help from my second supervisor and close friend, Floyd Chitalu, who engaged in patient discussions and provided words of encouragement. My family, particularly my parents, offered me the strongest emotional support throughout my student life, and words cannot adequately convey my appreciation. Finally, I am immensely grateful to my wife, Mengying Zhang, who not only prepared delicious meals but also accompanied me during the endless dark Scotland winters.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Linxu Fan*)

To my grandmother

Table of Contents

1 Introduction **1**

1.1 Related work . . . 3

1.1.1 Non-physical fracture patterns . . . 3

1.1.2 Brittle object . . . 5

1.1.3 Thin shell object . . . 12

1.2 Problem statement . . . 19

1.3 Contributions . . . 21

1.4 Thesis structure . . . 22

2 Theoretical background **23**

2.1 Kinematics . . . 23

2.1.1 Deformation gradient . . . 23

2.1.2 Constitutive model . . . 24

2.1.3 Stress tensor . . . 25

2.1.4 Governing equations . . . 26

2.2 Material point method . . . 26

2.2.1 Weight function . . . 27

2.2.2 Full scheme . . . 28

2.2.3 Moving least-squares material point method . . . 30

2.3 Phase field method . . . 31

2.3.1 Updating particle damage . . . 32

2.3.2 Effective Stress weakening . . . 33

2.4 Basics of NURBS . . . 34

3 Brittle object fracture **37**

3.1 Introduction . . . 38

3.2 Method overview . . . 39

3.3 Extract non-manifold crack surface from damage 41
3.3.1 Rasterizing particles onto the grid 43
3.3.2 Computing the Voronoi diagram 43
3.3.3 Identifying interior crack faces 44
3.3.4 Identifying extending crack faces 45
3.4 Enforce material discontinuity by using two velocity fields 47
3.4.1 Discontinuous velocity fields 48
3.5 Extract fragments 50
3.5.1 Identifying Voronoi-tessellation regions 51
3.5.2 Cut fragments in the material space 52
3.6 Rigid-body simulation 53
3.6.1 Decimate mesh 53
3.6.2 Contact impact 53
3.6.3 Rigid-body solver 55
3.7 Add realistic brittle fracture details 56
3.8 Results 61
3.8.1 Method validation 61
3.8.2 Dynamic rigid body fracturing 69
3.9 Conclusions 75

4 Thin shell object fracture **81**
4.1 Introduction 82
4.2 Shell kinematics 84
4.2.1 Shell motion 85
4.2.2 Elastic potential, yield condition and return mapping 86
4.3 Discretization using MPM 86
4.3.1 NURBS-based Lagrangian representation 87
4.3.2 MPM particle classification and simulation 89
4.4 Co-dimensional fracture 91
4.4.1 Calculating stress and damage 91
4.4.2 Consistent crack extraction 93
4.5 In-plane stress 96
4.6 Resolving discontinuities in material space 97
4.7 Resolving discontinuities in world space 100
4.7.1 Grid-based partitioning 101

4.7.2 Guaranteeing separation by contact forces 102
4.8 Coupling NURBS patches 104
4.8.1 C^0 continuity 105
4.8.2 C^1 continuity 107
4.9 Get physical rendering mesh 107
4.9.1 Overview 108
4.9.2 Find elements separated by cracks 108
4.9.3 Calculate polygons vertices' world-space position 109
4.10 Results 111
4.10.1 Mode tests 111
4.10.2 Elasto-plasticity 112
4.10.3 Cohesion-free contacts and prescribed cut paths 113
4.10.4 Consistent crack paths with branching 113
4.10.5 Influencing fracture with material parameters 114
4.10.6 Stitching NURBS patches 114
4.10.7 Couple with a rigid-body solver 117
4.11 Conclusion and Discussion 117

5 Conclusions **123**

A Weak decoupling of compatible particle in cell (CPIC) **127**

B Gradient of damage time **131**

C Fracture energy and strain energy **133**

D Calculation of patch coupling matrix **135**
D.1 Insert one knot to B-spline 135
D.2 Insert multiple knots to NURBS 136

Bibliography **139**

List of Figures

2.1 Deformation mapping. 24
2.2 An overview of the full MPM algorithm. 28
2.3 Crack is represented as continuous phase field. 31
2.4 2D illustration of the Rankine damage model. 32
2.5 Linear softening and exponential softening. 33
2.6 NURBS curve and basis functions. 36
2.7 A 2D NURBS surface. 36

3.1 Brittle fracture teaser figure. 37
3.2 Brittle fracture simulation method overview. 39
3.3 A visual summary of the stages to fracture an object. 40
3.4 An overview of how we extract a crack surface. 42
3.5 Damage gradient vanishes in excessive thick crack region 48
3.6 Estimating crack orientation. 49
3.7 Boolean intersection between the domain and the boundary surface. . 50
3.8 An illustration of extracting fragments from Voronoi tessellation regions. 51
3.9 Different fragment cutting methods. 52
3.10 Simulation mesh and rendering mesh. 54
3.11 Gradient vector field visualisation. 56
3.12 Crack damage time visualization. 57
3.13 A visual depiction of chevron lines. 58
3.14 Mode tests results. 62
3.15 Pre-notched rectangular plate test. 63
3.16 Breakage result at different resolutions. 64
3.17 Interpolation effect on smeared crack path thickness. 65
3.18 Comparison between a single velocity field *vs.*multiple velocity fields. 66
3.19 Fracture energy plot. 67

3.20 Various fracture detail patterns. 68
3.21 Using damage thresholds as a fracture stopping criteria. 69
3.22 Formation of interior voids due to crack bifurcation and merging. . . . 70
3.23 A shattering glass. 70
3.24 A bunny is hit by a bullet. 71
3.25 Controlling the number of brittle fracture fragments 71
3.26 Crack-bifurcation comparison. 72
3.27 Fragment generation and distribution. 73
3.28 Animation comparison between MPM and XFEM. 73
3.29 Four simple shape objects drop to the ground and shatter. 74
3.30 Two armadillos collision. 74
3.31 Comparison of fragment volume distribution with Mott's Formula. . . 75
3.32 A failed cut. 77

4.1 Thin shell fracture teaser figure. 81
4.2 Thin shell mapping. 84
4.3 System overview of codimensional object crack simulation. 87
4.4 Schematic illustration of thin-shell as a NURBS surface. 88
4.5 The evolution of phase field. 92
4.6 Extreme over-stretching occurs without material healing. 93
4.7 Overview of the 2D crack extraction algorithm. 94
4.8 Extract a consistent crack. 95
4.9 Comparison of extracted cracks with previous method. 96
4.10 Comparison of in-plane stress models. 97
4.11 Remove normal component to get the plane-stress tensor. 98
4.12 Approximating field quantities of a quadrature point near the crack. . 99
4.13 Smooth transition of our MLS formulation. 101
4.14 Clean separation of codimensional fragments. 104
4.15 Non-conforming NURBS patch coupling. 104
4.16 Get physical rendering mesh. 109
4.17 Deviate shared vertices by a small magnitude. 110
4.18 Cracked NURBS patch and physical rendering mesh. 110
4.19 Benchmark mode test results. 112
4.20 Bullet hits a thin metal sheet. 112
4.21 A sheet is pre-cut with an Archimedean Spiral curve. 113

4.22 Tear a thin rubber. 114
4.23 Crack propagation in weakened material regions. 115
4.24 Cloth hanging test. 116
4.25 Cylinder composed of two patches drops to the ground. 116
4.26 A garment is torn. 117
4.27 Inflatable mannequin 118
4.28 Example breakdown of total simulation time. 119
4.29 Breaking a thin glass. 120

A.1 Weak decoupling of CPIC 128

List of Tables

3.1 A summary of the set notation that is used to classify grid nodes. . . . 43
3.2 Fracture detail control parameters. 59
3.3 Brittle fracture simulation performance summary 76

4.1 Notation and definitions of continuum thin shell. 85
4.2 A summary of the set notation that is used to extract consistent cracks. 94
4.3 Thin shell fracture simulation performance summary. 118

Acronyms

MPM	material point method	iii
NURBS	Non-Uniform Rational Basis-Spline	iii
CPIC	compatible particle in cell	xi
FEM	finite element method	1
XFEM	extended finite element method	1
BEM	boundary element method	1
DOFs	degree of freedoms	1
BVH	bounding volume hierarchies	1
CD	continuum mechanics	2
CDM	continuum damage mechanics	2
PFM	phase field method	2
SVO	sparse voxel octree	4
SPH	smoothed particle hydrodynamics	5
XMPM	extended material point method	11
EFGM	element-free Galerkin method	17
XIGA	extended isogeometric analysis	17
MLS	moving least square	18
MLS-MPM	moving least-squares material point method	26
DFG	damage field gradient	47

Chapter 1

Introduction

The advancement of the game and film industry has made physical fracture simulation an integral component of visual effects. Fast and precise simulation of real-world fractures is a necessity in these sectors, encompassing the destruction of buildings, shattering glass, and car crashes, among others. The speed and accuracy of these simulations significantly enhance the realism of visual effects and grant artists the freedom to bring their creative visions to life.

In the realm of physical fracture simulation, the initial step involves discretizing the domain. Traditional mesh-based approaches like finite element method (FEM) and extended finite element method (XFEM) employ volumetric mesh representations, necessitating complex remeshing preprocessing to transition from a surface mesh, which is more common in computer graphics applications. In contrast, boundary element method (BEM) (Hahn and Wojtan, 2016) avoids volumetric representation by utilizing only degree of freedoms (DOFs) on the surface mesh. However, it sacrifices the efficient handling of spatially varying material parameters. Conversely, particle-based methods like MPM discretize the domain using material points, a process that can be readily achieved by sampling the surface mesh.

Following discretization, the physical fracture simulation can be viewed as two distinct problems: fragment dynamic motion and crack initiation and propagation. The former deals with interactions between fragments, encompassing collision detection and contact treatment, which impacts the object's deformation. This deformation, in turn, initiates and propagates cracks within the material.

Traditional collision detection methods assign bounding volumes to objects, and they determine whether two pairs of object bounding volumes intersect. However, constructing bounding volume hierarchies (BVH) necessitates additional memory stor-

age and can become computationally expensive as the number of fragments increases, leading to suboptimal scaling. In the simplest scenario, the computational complexity is O(N^2) where N is the number of fragments (Chitalu et al., 2018).

Once the collision between two objects is detected, the contact force is computed based on a specific model and applied as a Neumann boundary condition. This contact force contributes to the dynamic motion of the entire object in the world space and induces internal deformation within the material space. When the deformation reaches a critical point, material failure occurs, initializing a crack. Two methods for crack propagation are available: the explicit geometry method and the implicit method.

The geometry method gradually extends the crack using an explicit surface, involving the calculation of the stress intensity factor to determine the crack's direction and speed (Hahn and Wojtan, 2016). Complex geometry operations are required to connect these explicit crack surfaces, as real-world fractures often exhibit dynamic merging and bifurcation behaviors, giving rise to stability and robustness issues. The continuum assumption of continuum mechanics (CD) precludes the existence of a crack as a strong discontinuity. FEM (Pfaff et al., 2014) refines mesh geometry to allow crack alignment with element edges and further separation, but it increases the system's DOFs and can lead to nearly degenerated elements. XFEM (Chitalu et al., 2020) eliminates the need for remeshing by adopting an enrichment function to represent the crack, but it faces difficulties with branching cracks, especially in three-dimensional scenarios.

In contrast, continuum damage mechanics (CDM) initializes and propagates cracks implicitly. Methods like phase field method (PFM) consider the cracked material as a continuum with a scalar quantity called "phase" weakening the material properties in response to evolving stress and strain, thereby accounting for cracks without requiring explicit crack surface representation. PFM excels in handling complex crack merging and bifurcation phenomena.

Despite handling continuity and discontinuities in an integrated fashion and the ability to model complex branching effects, PFM faces several challenges for simulating brittle fracture. These challenges include sticky artefacts where material fragments fail to break off; a pronounced difficulty in simulating partial cracks due to slight material softening; and inaccuracies in the simulation due to the lack of an explicit crack surface representation which then results in excessive thickening of the damaged regions, unintended branching, and exacerbated energies. In addition, simulating PFM can be costly, due to the large amount of computation in each step when the object has little-to-no impulsive contact. PFM is also limited to produce realistic fracture patterns

on the crack surface due to the spatially varying material parameters.

For codimensional thin shell objects, PFM can cause excessive damage regions, resulting in over-stretching artifacts. Due to the high surface-to-volume ratio of thin shell objects, they are often simplified using shell models like Kirchhoff-Love, which necessitate the representation of objects as a continuum surface using methods such as NURBS and subdivision surfaces. In the cracked region, the material loses its ability to resist tensile loading, leading to over-stretching artefacts. Moreover, shell fragments cannot be cut without an explicit crack surface, as field quantities like position and velocity remain continuous throughout the shell.

The primary objective of this thesis is to develop an integrated approach that combines MPM with PFM to simulate crack initiation and propagation while automatically handling collision detection and contact treatment. Our goal is to address challenges during simulation such as slow performance, over-stretching issues, and the lack of sharp features.

1.1 Related work

In this section, we review related work about fracture simulation of both brittle objects and thin shell objects, focusing on literature within computer graphics.

1.1.1 Non-physical fracture patterns

Geometry-based methods have gained popularity due to their simplicity and speed, making them ideal for real-time applications. Within this class of methods, some have successfully employed pre-defined crack patterns like Voronoi cells (Raghavachary, 2002) to compute fragments. These methods are known for their high performance in the gaming industry.

One approach involves pre-computing fractures during the pre-processing step (Hellrung et al., 2009), applying cracks directly to the object before simulation. While this approach can achieve nearly real-time performance, a significant challenge is defining the cracks. During the modelling stage, artists must meticulously select crack patterns that align with the location and intensity of contact impacts. To allow for repeated breakage, artists have to provide a complex hierarchy of cracks for each fragment. The workload grows exponentially with the number of breakages, making it impractical for large scenes with thousands of fragments generated by hierarchical breakage.

Other methods have employed generic fracture patterns to introduce dynamic variability at the impact location(Su et al., 2009; Mould, 2005). These fracture patterns are pre-computed as a decomposition of a large block space, and they are applied to objects using Boolean operations, such as the level-set method(Osher and Fedkiw, 2001). However, achieving sharp edges for tiny debris requires super-high-resolution background grids, which becomes impractical for scenes involving smashing effects as well.

Volumetric Approximate Convex Decompositions (VACD) (Müller et al., 2013) similarly address the limitations of pre-fracturing by decomposing geometry into compound shapes of convex forms to achieve remarkable results. However, the resulting fragments tend to be overly regular, and their boundaries appear flat due to the planar cut. Moreover, the fracture patterns do not adapt to contact scenarios or material properties.

Schvartzman and Otaduy (2014) extend Voronoi techniques, distributing fracture fragments (cells) according to the deformation and example-based guidance, to ensure that the generated fragments correspond to external forces. While Voronoi methods successively offer similarity in shape and distribution, fracture patterns remain restricted to the underlying tessellation which prohibits realistic partial cracks and detailed patterns without rendering effects. This stochastic method also tends to miss the obvious structural weakness which should break upon a small impact.

Domaradzki and Martyn (2018) utilizes sparse voxel octree (SVO) data structure to represent the boundary of solid objects cutted by Voronoi diagram pattern. The method can model non-planar surfaces of the fractured pieces as well as their concave shapes which are not possible in previous Voronoi-based methods. It has been applied to the simulation of brittle thin shell objects like vases, pots and pitchers.

Sellán et al. (2023) introduce a novel method for pre-fracturing stiff materials by computing the shape's fracture modes, which define the fracture patterns of the object under impact. While this method demonstrates correctness over purely geometry-based approaches, it falls under the category of pre-fracturing, as the fracture simulation is static rather than dynamic, and it does not support partial cracks or secondary cracks. For large meshes, the precomputation of fracture modes can become slow and may not be suitable for real-time applications.

However, all the methods mentioned above have a fundamental limitation, namely the inaccuracy of crack patterns. In mechanics, the fracture is initialized in regions where the material is weaker (impurity or void) or the internal stress is larger. There

are two material parameters related to fracture: *strength* and *toughness*. The former describes the loading required to initialize a new crack while the latter measures the force needed to propagate an existing crack. Therefore, the crack patterns are fully determined by the external loading and local material properties. Geometry-based methods, however, define the crack patterns in a purely heuristic fashion by ignoring the underlying material properties. These methods are suitable for online real-time applications which don't demand high accuracy. In recent years, the rapid development of the game and film industry prefers physics-based methods for better realism.

1.1.2 Brittle object

In this subsection, we first review physics-based fracture formulations, including mesh-based and particle-based methods, e.g., continuum damage mechanics fractures, as described in the following paragraph. Next, we discuss the quasi-static simulation, which is essential for accelerating brittle fracture performance. Then, we review several explicit crack surface representations. The explicit crack surface aids in decoupling physical quantities and segmenting cut fragments. After that, we briefly discuss methods that resolve contact and collisions in MPM simulations with explicit crack geometry. Finally, we review related methods for adding detailed fracture textures to fragments to enhance realism.

Physics-based fracture formulation: There is a wide range of methods for enriching fracture effects with stress-based physical simulation. These include FEM (O'Brien and Hodgins, 1999; Müller et al., 2001; Koschier et al., 2014; Wicke et al., 2010; Pfaff et al., 2014), XFEM (Kaufmann et al., 2009a; Chitalu et al., 2020), BEM (Zhu et al., 2015; Hahn and Wojtan, 2015, 2016), and mesh-less methods like smoothed particle hydrodynamics (SPH) (Cleary and Das, 2008; Wang et al., 2020b), peridynamics (Chen et al., 2018) and the renowned hybrid Lagrangian-Eulerian MPM (Wolper et al., 2019, 2020).

These methods rely on the principles of continuum mechanics, which are underpinned by two fundamental premises. Firstly, the material completely occupies the space it inhabits and can be continuously divided into infinitesimally small elements. Secondly, the material exhibits homogeneity at each specific location. Guided by these principles, numerical methods discretize the objects into finite elements. These methods are categorized based on the allocation of DOFs, the resulting equations to be resolved, and the level of mesh refinement employed in simulations.

FEM DOFs correspond to vertices of a volumetric mesh, which imposes the need for remeshing during crack propagation to align the crack path with element edges. However, remeshing has several drawbacks. Firstly, remeshing introduces additional computational costs. The process of modifying the mesh structure, updating element connectivity, and transferring data from the old mesh to the new one can be time-consuming. Secondly, remeshing can lead to an accuracy loss. During the remeshing process, the original mesh is typically simplified or refined. Lost information induces accuracy loss. Thirdly, remeshing can lead to mesh distortion. Distorted elements can negatively impact the accuracy of the analysis, causing inaccurate stress or strain distributions. In severe cases, the system might fail due to degenerated elements. Lastly, solutions can become discontinuous in the remeshed region. During remeshing, the connectivity between elements changes, leading to solution discontinuities, where results abruptly change across the mesh interface.

XFEM compliments FEM by extending the standard basis functions with enrichments, thus allowing displacement discontinuities and stress singularities to be modelled without remeshing. Despite this upperhand, XFEM has additional challenges. XFEM requires the selection of appropriate enrichment functions which depends on the type and geometry of the problem, and selecting the right functions can be challenging. For example, XFEM is struggling to enrich elements with crack bifurcation/branching. The crack tip enrichment is also cumbersome, especially in 3D manifold. Besides enrichment, XFEM is sensitive to the mesh configuration, especially near the discontinuities. Large volume-ratios in Heaviside enriched elements can cause severe instability. Furthermore, integrating enriched elements requires specialized techniques, such as subdomain integration or numerical quadrature rules (see also (Chitalu et al., 2020; Richardson et al., 2009; Koschier et al., 2017) for further discussion).

BEM offers a surface-mesh based discretization, where a boundary integral form of the governing equations is solved instead of (volumetric) PDEs as in FEM. System DOFs are reduced, combined with an amortized cost of matrix assembly during propagation. BEM is however limited in its treatment of spatially varying elasticity parameters and remains dependent on a mesh for simulation which must be updated to model crack propagation. Mesh resolution and crack propagation time are reduced by adopting level sets (Hahn and Wojtan, 2015) and making simplifying assumptions for quick estimation of stress intensities (Hahn and Wojtan, 2016), but stiffness matrix pre-computation is still necessary with each crack-initiating rigid body collision. For simulation problems with higher DOFs, fast summation (*e.g.* the Fast Multipole

Method (Zhu et al., 2015)) is a solution to speed up solver time but may yield diminishing returns due to overheads on medium-sized problems.

Meshless methods, such as SPH and peridynamics, involve discretizing the domain and storing material properties with individual particles. Each particle interacts with neighboring particles within a support radius. In SPH, these interactions among particles are utilized to estimate physical quantities at specific points, such as density, velocity, and pressure. Notably, SPH can naturally accommodate significant material deformations and even cracks (Chen and Kulasegaram, 2009; Wang et al., 2019b). However, SPH can exhibit instability due to insufficient resolution and is particularly sensitive to numerical noise (Sigalotti and López, 2008). Conversely, peridynamics establishes explicit bonds between adjacent particles, which are defined by nonlocal constitutive models. These bonds can rupture when subjected to substantial internal forces, effectively mimicking crack propagation. Peridynamics has found success in simulating various materials, including brittle material (Ha and Bobaru, 2011), rocks (Rabczuk and Ren, 2017), clay (Sedighi et al., 2022), and ductile materials (Hu et al., 2020). Nonetheless, peridynamics is constrained by its tendency to cause material smashing. In regions with cracks, multiple bonds between particles may break, causing these particles to disintegrate explosively and generate tiny debris (Littlewood, 2015). It's worth noting that meshless methods are generally more computationally demanding compared to mesh-based approaches.

Continuum damage mechanics fracture: Wolper et al. (2019) develop a fully elastodynamic continuum damage MPM simulation (CD-MPM) to model phase-field fracture, producing impressive results. They combine continuum damage mechanics with MPM to present a *non local* variational phase field method to animate isotropic ductile fracture (see also the closely related work of Kakouris and Triantafyllou (Kakouris and Triantafyllou, 2017)). CD-MPM has been further extended by Wolper et al. (2019), continuing use of non local phase fields but coupled with a hybrid Lagrangian force model (Jiang et al., 2015) to encode material anisotropy. Wolper et al. (2020) partly mitigate the inability of CDM methods to represent world space cracks but achieve this at the cost of re-introducing a requirement for volume meshing akin to (Wang et al., 2019a). These methods, nonetheless, cannot simulate stiff materials due to the restriction of tiny timestep size. Moreover, materials in large excessively weakened regions fail to glue together since they lose the ability to resist tensile loading, which causes the smashing artefacts just as peridynamics (Littlewood, 2015). These methods also have

difficulties in producing sharp and clean cuts as well as in rendering discrete particles unless very high-resolution particle representation is used.

Wang et al. (2019a) produces clean cut by using Delaunay tetrahedralization of the object. The particle lies in the center of each element and the connection between particles breaks when the stress exceeds a threshold. They also introduce a pre-scoring approach to remove numerical fracture and define material failure through the elasto-plastic constitutive equations. To enhance realism, a smoothing postprocessing step is applied to the crack surface to remove the jaggy surface pattern caused by randomly sampled points and Delaunay tetrahedralization. Despite this smoothing step, the crack patterns are still heavily affected by particle sampling density/tetrahedron mesh topology and grid resolution

The method described in (Wolper et al., 2019) is meshless, but deals with non-local phase field which requires a sparse linear system with iterative solvers. Furthermore, a staggered integration is needed to update the phase value. In practice, multiple staggered iterations will be necessary for the convergence of results. Both bring in additional computational costs. A simpler scheme is also described in (Wretborn et al., 2017) where the authors use prefracturing by way of particle constraints ("glue particles"), simulating one crack at a time under mode I loading.

Our method–which is also meshless–is based on *local* phase field by Cervera and Chiumenti (2006) (see also (Homel and Herbold, 2017)), offering the usual MPM discretization, requiring no linear system solves, and precluding any requirement for a tetrahedral-mesh dependant strategy like (Wolper et al., 2020) and (Wang et al., 2019a).

Quasi-static simulation: Fully elastodynamic simulations work well for simulating brittle fracture but risk artefacts or require high number of small timesteps to complicate parameter tuning since the material is stiff. Quasi-static formulations of fracture mechanics offer a solution to this problem, without-which simulation is prohibitively expensive. A number of methods including (Chitalu et al., 2020; Hahn and Wojtan, 2015; Glondu et al., 2013; Hahn and Wojtan, 2016; Su et al., 2009; Müller et al., 2001; Koschier et al., 2014; Bao et al., 2007) present quasi-static approaches to simulating fracture as objects are treated as rigid-bodies in their free motion. Typically, the crack initiation and propagation are performed in each breakable objects using stress-based model like J-integral (Hahn and Wojtan, 2015) or displacement-correlated model (Chitalu et al., 2020). When cracks are fully developed, level-set method (Hahn and Wojtan, 2016) or geometry based method (Chitalu et al., 2020) cut the original object

into descendant fragments along the explicit crack path. Then a rigid-body solver detects the collision and contact between fragments and calculates the repulsion force to move those fragments. Thanks to this hybrid formulation, the rigid-body timestep size is generally much larger than the crack simulation timestep size. Therefore, the quasi-static simulation can handle very stiff material with reasonable simulation time.

Such quasi-static simulations have thus-far used mesh-based discretizations like BEM (Hahn and Wojtan, 2015), but we depart from this traditional viewpoint by adopting CDM to alleviate direct meshing for propagation by representing cracks as continuous fields of evolving damage variables. We can thus propagate cracks implicitly avoiding costly geometry processing, which enables the simulation of complex crack bifurcation/merging behaviour. In-line with phase-field fracture (Wolper et al., 2019), our method is based on the MPM (Sulsky et al., 1995) but coupled to rigid body simulation, where mesh resolution (material particles) can be freely chosen by the user. The only constraint is that the number of material particles is sufficient to simulate damage. Unlike (Wolper et al., 2019), our method can produce sharp and clean cut of damaged fragments with significantly less simulation time.

Explicit crack surface representations: In mesh-based methods, an explicit crack surface is needed to introduce strong discontinuity and separate the fragments. The crack surface is generally initialized as a planar element which is aligned to be orthogonal to the crack propagation direction (Hahn and Wojtan, 2015; Chitalu et al., 2020; Guo and Nairn, 2006; Liang et al., 2021). New elements are then inserted into the mesh as the new crack fronts. The disk diameter is determined by the element size or manually specified. The crack propagation stops when it reaches the boundary or approaches the other cracks. Considering the variation of material properties, e.g., toughness and strength, Hahn and Wojtan (2015) uses several sub-elements inside one large element in the crack front. This can effectively increase the crack surface mesh resolution with little extra cost. However, this explicit method can not produce complex crack geometry with branching and merging phenomenon. The algorithm itself may generate degenerated elements in severely distorted regions where the crack propagation direction changes dramatically.

Alternative methods include the aforementioned non-physical fracture patterns defined by Voronoi tesselation or convex shapes. However, these methods produce crack surface mesh that doesn't adhere to physical laws. Wang et al. (2019a) also has an explicit crack representation, which is defined by the underlying Delaunay tetrahe-

dralization of the material particles. When a crack breaks the bonds between nearby particles, the Delaunay cell edges become the crack front. By querying all broken bonds, an explicit surface mesh can be constructed from corresponding edges. Due to the random distribution of material particles, the Delaunay cells may become very jaggy at coarse region. Although the smoothing post-processing step can mitigate this artefact, the overly smooth surface loses the sharp features of brittle material fracture. Furthermore, the pre-scoring (see Sec.5 in (Wang et al., 2019a)) may produce undesirable tiny debris near the cracked region causing the so-called *artificial shattering* (Hahn, 2017). This artefact is caused by material failure in a region where most particle stresses exceed the threshold.

Representing crack surfaces with MPM is a well-known issue, often relying on level sets to capture explicit material discontinuities as object boundaries. Previous methods have typically relied on numerical fracture (Stomakhin et al., 2013), pre-specified cracks (Wretborn et al., 2017) and particle deletion (Homel and Herbold, 2017) to suffer from visual artefacts. Inspired by Voronoi tessellation (Klein, 1989), we propose another solution, adopting a *medial-surface* approximation for crack representation from the damage phase field. We first tessellate the fully damaged region and then select the corresponding cell edges as the non-manifold crack surface.

Decouple physical field & Mesh cutting: The last step in crack simulation is updating the geometry of the object under duress from the cracks. Adding the fracture in FEM requires a modification of this geometry, which is a problem as fracture surfaces tend to become complex leading to practically challenging intersection tests and ill-shaped elements that need remeshing (Wicke et al., 2010; Pfaff et al., 2014; Koschier et al., 2014; Chen et al., 2014). Workaround solutions have included element duplication (Molino et al., 2005a) and sub-element accurate constructions for mass and stiffness computations (Wojtan and Turk, 2008; Nesme et al., 2009). XFEM-enrichment coupled with an explicit surface mesh is aforementioned to bypass the need for explicit remeshing. However, cuts and cracks are typically represented as a single surface (3D) which lacks the desired branching phenomena. This surface is either pre-specified as in (Koschier et al., 2017), or requires further post-processing to add detail using texture maps (Chitalu et al., 2020; Kaufmann et al., 2009a). Hahn and Wojtan (2015) adopt level sets to simulate detailed cracks with branching using a low resolution boundary element mesh, but results in volume loss, which is not always desirable, especially when the crack is only partial.

Level sets offer a simple method to partition the domain using boolean operations by treating crack surfaces as thickened solid volumes (see *e.g.* (Hahn and Wojtan, 2015)). This approach is robust and will produce convincing results given an adequate voxel size but it can also produce visible ('hairline') gaps when cracks do not propagate to the boundary which is rare in brittle fracture. Further, treating crack surfaces as solid volumes for cutting implies that cracks are applied by cutting away thin strips of material which leads to slight volume loss. Alternative mesh-based methods (*e.g.* (Zhu et al., 2015; Chitalu et al., 2020; Sifakis et al., 2007)) work directly with crack mesh geometry to produce precise and sharp fragments without volume loss. These methods work well for manifold geometry but do not support non-manifold cracks. Further, resolving polygon intersections in general position–or with symbolic perturbation (Lévy, 2016)–is an additional source of unpredictable runtime failure during cutting as meshes will also deteriorate in quality with further cuts.

Resolving contact & collision explicitly in MPM: As a hybrid Lagrangian/ Eulerian approach, MPM resolves contact and collision implicitly thanks to its continuum velocity field assumption. However, conventional MPM is unable to handle internal cracks with displacement discontinuity, since it does not distinguish between a continuous material and two adjoining fragments of the same material. The work that treats crack explicitly can date back to (Nairn, 2003) , who adds dual velocity fields in the background grid to introduce velocity/displacement discontinuity. A following work (Guo and Nairn, 2006) generalizes this dual velocity field method into three-dimensional cases. The disadvantage of this method is that only one single crack is permitted. For non-manifold cracks, several fragments may present near the intersection point, which cannot be represented properly with two velocity fields.

Liang et al. (2021) propose extended material point method (XMPM), which adds an enrichment term to the MPM basis function similar to XFEM to yield velocity discontinuity without the need of dual velocity fields. Similar to XFEM, XMPM struggles to find a suitable enrichment function in the case of branching and merging cracks. Furthermore, XMPM needs to maintain a level-set data structure to calculate the crack's normal direction and the distance between a background grid node to the crack surface at additional computational cost.

Homel and Herbold (2017) employs phase field gradient to partition materials on both sides of the crack to resolve internal frictional contact. This method avoids the use of an explicit crack geometry and can handle arbitrary branching cracks. However,

the damage gradient can vanish in the excessively thick damaged region, leading to ambiguity in partitioning fragments (see Fig. 3.5).

CPIC (Hu et al., 2018) is another alternative to separate materials. This method can also handle arbitrary branching cracks. In CPIC, material particles and grid nodes are classified to be ”compatible” and ”incompatible” according to a signed distance field. Grid nodes receive contributions from both ”compatible” and ”incompatible” particles during particle-to-grid transfer. However, such a transfer only provides a weak decoupling of field quantities, i.e., velocity (see Appendix A). To enforce non-penetration condition, a repulsion force is also applied to push fragments which needs to be manually specified. Furthermore, the crack surfaces need to be pre-defined at the beginning of the simulation and cannot be propagated.

Add fracture details to the crack surface: Real-world brittle fracture often exhibits complex patterns which indicate the crack propagation direction and speed. This pattern is named *chevron marks* and is caused by the variation of spatial material parameters, i.e., *toughness*. Chen et al. (2014) perturbs the low-resolution simulating mesh vertices using a custom-designed material field to animate high-resolution fracture details. This method, however, doesn't consider the fracture surface's evolution temporally, potentially leading to less smooth crack surfaces in the high-resolution mesh. Hahn and Wojtan (2016) propagates the high-resolution explicit crack surface by adjusting the elasticity according to the spatially varying toughness parameter in their fundamental solution. The material parameter variations influence the crack propagation direction locally, which produces realistic fracture patterns. Chitalu et al. (2020) adds surface texture to the simulated low-resolution mesh to animate crack patterns as a post-processing step. However, all the methods mentioned above require the creation of artificial fields, which must be manually designed for each simulation. In contrast, our method leverages damage information to generate authentic crack patterns, enabling artists to easily control the appearance with just a few parameters.

1.1.3 Thin shell object

In this subsection, we first review thin shell formulations and then examine how fractures can be incorporated. Next, we discuss methods for obtaining an explicit crack path on a 2D manifold. We then briefly introduce methods that explicitly resolve collision and contact in MPM simulations. Additionally, we demonstrate how to enforce discontinuities on NURBS surfaces. Finally, we explore related methods that enable

the coupling of NURBS patches to achieve desired continuity levels.

Thin shell formulation: Simulation of thin shells in high resolution was traditionally done by embedded mesh methods (Wojtan and Turk, 2008; Bargteil et al., 2007) or using triangle mesh-based methods (Baraff and Witkin, 1998; Narain et al., 2012, 2013; Harmon et al., 2008; Pfaff et al., 2014). In the embedded mesh method, an external surface mesh is incorporated into an interior volumetric tetrahedral mesh. The volumetric mesh is employed for elasticity and plasticity simulations. The deformations of the internal mesh are then transferred to the surface mesh, which is used for collision detection and computing contact forces. Consequently, this method allows for a higher-resolution surface mesh to be used atop a lower-resolution interior mesh. Strictly speaking, this approach cannot accurately simulate the elastic and plastic behavior of real codimensional thin objects that are independent of a volumetric object.

The mesh-based methods discretizes the thin shell with triangle meshes, and use a simplified model to represent thin shells. Breen et al. (1994) describe two kinds of forces acting in thin shells: the in-plane shearing force and out-of-plane bending force. Baraff and Witkin (1998) formulate the shearing force on a per triangle element basis, and the bending force is formulated on a per edge basis. In specific, the bending force is computed by comparing the deviation of adjacent elements' normal directions. Early works primarily focus on the elasticity of thin shells. However, more recent works, such as (Narain et al., 2013) introduce a bending plasticity model to simulate folding and crumpling, preserving permanent sharp edges. They also dynamically refine the high-stress elements to get a higher-resolution representation of the crumpled regions. Similarly, Narain et al. (2012) presents a method to adaptively refine and coarsen triangle elements in regions with high-velocity gradient or large compression stress. During refinement, they create anisotropic elements that follow the curvature of wrinkles and creases to make the shell's wrinkle look smoother. Compared to a high-resolution simulation, the adaptive remeshing method significantly reduces runtime while achieving a similar visual result.

However, several hurdles exist when applying mesh-based methods. Firstly, stiff in-plane behavior causes the locking artefacts, resulting in improper bending resistance when the element edges do not align with the bending direction (Narain et al., 2013). Remeshing must be adopted to align the element edges according to an accurate calculation of the bending direction. However, remeshing is nontrivial as it can produce degenerated elements which incur instability. Furthermore, remeshing can cause

jittering artefacts when simulating relatively stiff material. Secondly, the collision detection process becomes more expensive due to the increased DOFs from refinement (Thomaszewski et al., 2005; Lu and Zheng, 2014; Harmon et al., 2008). In complex self-collision scenarios, e.g., crumpling, the collision detection step even becomes the bottleneck of simulation. Lastly, the contact force during collision is tuned manually which needs to be specified for each material and scenario. Besides, treating all collisions simultaneously can cause artificial dissipation (Harmon et al., 2008) where earlier collisions may prevent later ones.

In contrast, Lagrangian/Eulerian discretization of thin shell, i.e., (Jiang et al., 2017; Guo et al., 2018; Lv et al., 2022) can handle high-resolution models without extra cost. The shell geometry is represented by a continuum surface such as NURBS and subdivision surface, and the shell kinematics is simplified with a continuum model, i.e., Kirchhoff-Love. The continuum model can effectively avoid the locking artefacts arising from mesh-based method. The shearing force and bending force are decoupled by using different particles which facilitates the simulation of wrinkles and buckling effects. Collisions and contacts are implicitly handled through a background grid without hand-tuned impulsion force parameters. Furthermore, the frictional contact force can be easily incorporated through the decomposition of shearing force. For example, Guo et al. (2018) proposes a method to combine MPM with Kirchhoff-Love shell, which can resolve massive collisions (including self-collisions) without additional cost.

Thin shell fracture simulation: Physically-based thin shell fracture simulation can date back to the work of (Terzopoulos and Fleischer, 1988; Norton et al., 1991). Early works of thin shell fracture simulation initiate cracks via a simple stretching limit which cannot describe all physical cracks (Terzopoulos and Fleischer, 1988; Norton et al., 1991). More recent methods apply the principal stress threshold criterion, i.e., Rankine condition (Rabczuk et al., 2007). For thin shell object, the crack propagation direction is explicitly computed and then propagated similar to general 3D object using maximum tangential stress criterion (Erdogan and Sih, 1963), strain energy density criterion (Sih, 1974), displacement-correlated model (Chitalu et al., 2020) and J-integral (Li et al., 2020). These explicit methods have the same disadvantages as those for simulating brittle object fracture.

Phase field method (Francfort and Marigo, 1998; Bourdin et al., 2000; Amiri et al., 2014) is an alternative way that overcomes the drawbacks of explicit methods. The crack initiation and propagation are treated implicitly in an integrated fashion by track-

ing the evolution of the phase field in the 2D manifold. Phase field method has been successfully applied to simulate thick shell (Kikis et al., 2021), brittle thin shell (Kiendl et al., 2016), and ductile thin shell (Proserpio et al., 2021).

One limitation of the phase field method is the excessively thick crack path due to the lack of a strong discontinuity (Kiendl et al., 2016). Particles still couple in the damaged region even though they should separate physically. Therefore, cracks usually span several elements in the thickness direction (see Fig.10 in (Amiri et al., 2014) and Fig.13 in (Areias et al., 2016)). These elements are over-stretched because they lose part or all of their ability to resist tensile loading. The crack thickness can be reduced by using a higher-resolution mesh or adaptive local refinement (Paul et al., 2020) at, of course, increased computational cost. Another way to mitigate this artifact is by element deletion (Proserpio et al., 2021), which leaves a visible gap after deletion. This method doesn't conserve the system's mass and is not suitable for graphics applications. The phase value's monotonicity (Xue et al., 2021) also contributes to the excessively thick crack path. A crack cannot disappear after its formation. In other words, the material cannot be healed once damaged, which requires the phase to increase monotonically.

We also adopt a phase field approach for simulating tearing/crack propagation: we avoid the excessive expansion of the damaged region by introducing explicit crack paths that separate the crack region. Thanks to this explicit crack representation, we can enable material healing after damage. Therefore, the over-stretching artifact is greatly eased.

Explicit crack path in 2D manifold: An explicit crack path, often represented as the medial axis of the damaged phase field, is crucial for achieving a clean cut and minimizing over-stretching artifacts in thin shells. The conventional approach involves iteratively tracing the medial axis (Sherbrooke et al., 1995; Reddy and Turkiyyah, 1995). Starting from a junction point, this method incrementally follows seams until all medial axes have been traversed. This technique has been successfully applied to simple shapes like cylinders and cubes and has also been extended to more complex polyhedrons. Culver et al. (2004) introduces an approach that employs exact arithmetic and precise representations to calculate the accurate medial axis of a 3D polyhedron, which achieves orders of magnitude performance gain compared with previous methods. The method nevertheless becomes inefficient when dealing with nearly degenerated junctions or fails at a singular point. Except for tracing continuously, space subdivision

is also adopted to compute the medial axis by tessellating the domain into fixed-size voxels or octree (Etzion and Rappoport, 2002; Ma and Sonka, 1996). Vleugels et al. (1995) recursively subdivides the region which contains part of the medial axis to a specified resolution.

Another approach to computing a skeleton involves the use of a ball-shrinking algorithm. Ma et al. (2012) propose a method to generate medial axis points from surface samples and normal vectors. They obtain a maximal tangent ball by iteratively reducing its radius by comparing it with nearest neighbors. However, there are two limitations associated with this approach. Firstly, it relies on accurate surface normal values as input, which may not always be readily available. Secondly, the output consists of scattered points rather than connected curves. This lack of medial axis information is problematic, as it is essential for the separation of materials in certain applications.

The methods mentioned above have a notable limitation: they require a surface mesh representation as input. When dealing with scattered points, such as damaged particles, techniques for skeletonization of point clouds can be employed for this purpose. For example, Tagliasacchi et al. (2009) can extract a curved skeleton from a point cloud, although it is primarily suited for cylindrical shapes. Following work from Huang et al. (2013) removes the shape restrictions by utilizing the L1-medial axis of the point cloud. However, these methods often demand carefully chosen parameters and are not well-suited for 2D manifold geometries. Additionally, they may underestimate small features, such as branching thin paths. Kurlin (2015) proposes to use optimal subgraph to get a 1D homologically persistent rough approximation of a point cloud. The approximated skeleton is very noisy with many small branches. Chaudhury and Godin (2020) uses stochastic optimization to find a smooth skeleton given a good initial skeleton. However, none of these methods give a consistent approximation of the crack surface given a time-varying phase field. The crack surface obtained from a previous approximation may not necessarily be a strict subset of a subsequent approximation, leading to frequent changes in the historical crack path.

Resolving codimensional contact & collision explicitly in MPM: Due to the continuum assumption of MPM, cohesion artefact is inevitable for thin shell object with arbitrary branching cracks. Previous methods, such as those employed by (Homel and Herbold, 2017; Guo and Nairn, 2006), add two velocity fields to the grid nodes near the crack. However, in scenarios involving contact with more than two fragments, these methods cannot ensure exact decoupling nor preserve linear momentum. This limi-

tation stems from the insufficient number of velocity fields. In practice, there should be N velocity fields for N fragments, implying $N(N-1)$ contact pairs. Solving such a system becomes extremely difficult and nearly impossible when dealing with more than three fragments near a branching point. The associated computational cost and memory requirements are prohibitively high for these contact algorithms.

Phantom node method is an effective way to address multiple contacts, including self-contact and collisions between fragments. Unlike multi-velocity field method (Guo and Nairn, 2006), this method duplicates the nodes around fragments near the crack. For cases with more than two fragments, this method resolves contact pairs with the largest relative velocity iteratively, with the result converging to the real value after a finite number of iterations. In practice, just a few iterations are sufficient to ensure momentum conservation and non-penetration conditions. Phantom node method has been applied to simulate complex cracks in laminates (Van der Meer and Sluys, 2009), shells (Chau-Dinh et al., 2012), and brittle material (Mororó et al., 2022). In particular, Sun et al. (2021) improve the contact algorithm to enable the simulation of dynamic crack propagation of MPM. Nevertheless, the contact algorithm is not suitable for thin shells wherein the normal vectors are not clearly defined in the vicinity of the crack surface. In this thesis, we place particular emphasis on accurately defining the normal directions on thin shells near the crack, allowing the phantom nodes to manage collisions with precision.

Enforcing discontinuity in NURBS: As we use NURBS as our representation for the continuum thin shells, we review methods that can enforce discontinuity in NURBS. Gou and Zhu (2022) converts the NURBS basis functions into meshfree basis functions using, i.e., element-free Galerkin method (EFGM). It achieves high flexibility but sacrifices NURBS's higher-order continuity. Knot insertion is an effective way to introduce strong discontinuity inside NURBS, but is only valid for cases where the shape boundary is represented by a 1D curve (Andrade et al., 2022). For NURBS surfaces, knot insertion cannot be done locally; the entire patch needs to be split at the same uv parameters. Although T-splines (Chen et al., 2017) and LR-splines (Paul et al., 2020) are variations of NURBS that allow for local knot insertion, they tend to produce staircase artifacts as the fragments are cut off along the knots. The most popular method, extended isogeometric analysis (XIGA) (Li et al., 2020), adds an enrichment term to the NURBS basis function which can achieve exact decoupling. However, finding an enrichment function that can manage branching crack is nontrivial.

Alternatively, Kaufmann et al. (2009a) proposes harmonic enrichment functions to handle multiple, intersection, and arbitrary shaped cracks. Incorporating with an enrichment texture, it allows for fracture and cutting at a much higher resolution than the simulation mesh. However, it operates as a pre-scoring method, meaning that the crack path is predefined rather than progressively simulated. To simulate crack propagation, a time-varying enrichment texture must be provided during the simulation. Furthermore, this method has three fundamental disadvantages that limit its applicability in our system. First, the enrichment function is only C^0 continuous while our continuum shell representation needs at least C^1 continuity. Second, their approach demands a sufficiently large number of quadrature points within one element to capture the discontinuities and sharp features of cracks, which can lead to aliasing artifacts. Third, the enrichment function may fail to capture the crack when its path becomes too long and complex (see Fig.12 in their paper). In such cases, they need to manually divide the crack into several segments, which is not theoretically rigorous and convenient.

We replace the NURBS basis function in the vicinity of the crack with moving least square (MLS) approximation. It can achieve an exact cutting under arbitrary branching cracks. Furthermore, the consistency condition of MLS (i.e., that a pth order MLS approximation can reproduce exactly polynomials of less than or equal to order p) ensures exact reproduction of NURBS basis functions up-to the given order. Interpolation error due to the use of MLS is therefore practically non-existent, meaning that general crack propagation behavior is unaffected by using MLS basis functions and visual results also remain consistent with the use of NURBS basis function.

Coupling NURBS patches: Multiple NURBS patches are combined when modelling complex surface. Traditionally, patch couplings are enforced by either using the Lagrange multiplier (Mortar method) (Brivadis et al., 2015) or through a penalty formulation (Nitsche's method) (Nguyen et al., 2014). However, both methods have limitations. The Mortar method introduces additional DOFs, which not only increase the system size but also influence the simulation accuracy (Bouclier et al., 2017). Nitsche's method, on the other hand, relies on an appropriate choice of stabilization parameters, which can deteriorate the system if it is not chosen properly.

A simple yet flexible method to couple arbitrary order NURBS patches is recently proposed by Coox et al. (2017), which derives coupling constraints by virtual refinement in a 'leader-follower' formulation. This approach does not introduce new DOFs or additional parameters: the position of the follower (and higher resolution) patch

control points are updated according to those of the leader (lower resolution) patch. However, in an MPM setup, this kind of update rule violates the continuum assumption of the velocity field since the position update is not enforced through a grid-to-particle transfer. We thus propose a scheme where the position of the follower control points are updated based on the velocity in the background grid.

1.2 Problem statement

The goal of this thesis is to develop a physics-based method for general objects' fracture simulation for computer visual effects applications. Previous methods typically handle material deformation, motion, and fracture initiation and propagation as separate processes, which can introduce complexity and stability concerns. Our aim is to develop a method that can address these two issues in an integrated manner. In essence, this method should be capable of simulating fracture with automatic collision detection and contact treatment. Furthermore, it should be designed to be simple, fast, and robust for a wide range of computer visual effects applications.

We decide to use MPM as our discretization method and phase field method for the fracture simulation. MPM can achieve automatic collision detection and contact treatment without additional specialized algorithm. Phase field method allows for the implicit simulation of crack initiation and propagation. We aim to develop a unified solver that integrates the phase field evolution into the MPM framework with minimal modifications and computational cost. For generality, we consider two kinds of most common objects in computer graphics applications: brittle object and thin shell object, which have not been extensively investigated before.

When simulating brittle fracture, the primary concern is how to generate a sharp and well-defined boundary for the fragments efficiently. MPM uses particles representation, which cannot naturally render sharp features and tends to smooth them out. Additionally, the phase field method can generate small debris or scattered particles due to material separation. To address this, an explicit geometry is required to approximate the crack and separate the fragments. In general, this geometry should be an infinitely thin non-manifold surface, especially because branching cracks are common in real-world fracture. Furthermore, the MPM timestep size is restricted by the highly stiffness of brittle materials. Simulating real brittle materials like glass, marble and steel is prohibitively costly using fully elastodynamic MPM. Therefore, we need an efficient solver that can bypass the timestep size restriction while producing desirable

sharp features for brittle materials effectively.

For enhanced realism, brittle fracture surfaces should resemble real-world crack patterns, e.g., chevron marks. Previous methods have taken different approaches, including using a user-defined toughness field to influence the high-resolution crack surface propagation direction (Hahn and Wojtan, 2016) or applying a texture to perturb the crack surface vertices (Chitalu et al., 2020). However, these methods do not accurately capture the actual crack propagation direction and speed of brittle materials. Additionally, generating an artificial toughness field or texture can be a complex task. We aim to utilize only the damage information to produce realistic crack patterns, eliminating the need for manually defining toughness fields or textures.

In the case of thin shell objects, the method must be capable of cleanly separating the continuum shell into disconnected fragments. The phase field method typically generates a smeared crack, which means that quantities on both sides of the crack, such as displacement and velocity, remain continuous even when there should be a strong discontinuity between them. To address this issue, an explicit 2D crack path is essential to properly decouple these quantities. We require that this explicit crack path be consistent, meaning that the earlier crack path is a strict subset of the later crack path, to ensure that no debris is produced during the cutting process.

The shell geometry (NURBS) needs to be completely disconnected according to the explicit crack path. Previous methods have separated NURBS by using local knot insertion or enrichment techniques, which are suitable for our applications. What we need is a method that can perform arbitrary branching cuts without introducing staircase artifacts. Another issue is the over-stretching artifact near the crack boundary in thin shell fragments. This artifact is caused by material stress weakening, where the material loses its ability to resist tensile loading. While this artifact may not be prominent in simulations of brittle materials with small deformations, it becomes significant when simulating soft thin shells.

NURBS patches coupling algorithm is crucial when simulating objects composed of multiple patches. Complex objects like cars and ships are often constructed from multiple patches. Our method should have the capability to seamlessly connect these discrete patches by enforcing the appropriate boundary conditions, such as ensuring C^0 and C^1 continuity across the NURBS patches.

1.3 Contributions

The majority of the original material, results and figures in this thesis are derived from the following works. The contributions of each work are summarised as follows:

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

   (a) A crack initiation and propagation scheme, using a mesh-free local CDM model.

   (b) A method for extracting crack surfaces from material damage.

   (c) A method to model velocity discontinuity for dynamic decoupling of material particles along the crack for improved accuracy and to mitigate excessive thickening of smeared crack paths.

   (d) An algorithm for extracting rigid body fragment geometry as boolean operations between regions bounded by the crack surface and the volumetric domain without any volume loss.

   (e) A hybrid scheme to reduce the computational cost of brittle fracture simulation by replacing fully elastodynamic simulation with quasi-static simulation.

   (f) A new treatment of spatially varying toughness for enabling artistic control to produce realistic fracture patterns on the simulated crack surfaces using material damage.

2. **Fan, L.**, Chitalu, F. M., and Komura, T. *A Lagrangian/Eulerian formulation of thin-shell fracture.* (Intend to submit to TOG.)

   The contributions of this paper can be summarised as follows:

   (a) A method for simulating thin-shell fracture using local phase field method.

   (b) An algorithm to extract consistent non-manifold crack paths from a time-varying damage phase-fields.

   (c) A novel plane-stress formulation satisfying the Kirchhoff-Love assumption with fracture.

(d) An enrichment technique for interpolation functions, using moving least squares (MLS) method to model discontinuous field quantities in a domain represented by NURBS surfaces.

(e) A novel method for NURBS-patch coupling, enforcing at least C^1 continuity at the interface between patches.

1.4 Thesis structure

The remainder of the thesis proceeds as follows.

Chapter 2 introduces the theoretical background for fracture simulation. First, Sec. 2.1 introduces the basics of deformation theory and governing equations for elastic mechanics. Sec. 2.2 gives a detailed derivation of MPM and the extension to it, i.e., MLS-MPM. Sec. 2.3 describes the phase field method for fracture simulation. Finally, Sec. 2.4 introduces the NURBS surface briefly.

Chapter 3 introduces our quasi-static rigid body fracture simulation method. We describe our novel crack extraction algorithm and the fracture details enrichment approach in detail.

Chapter 4 introduces the hybrid Lagrangian/Eulerian thin shell fracture simulation method. We describe how we make extracted crack in Chapter 3 to be consistent and enforce discontinuity in NURBS surface using MLS approximation. We also present a novel approach to enforce continuity between non-conforming NURBS patches.

Finally, we summarize the thesis in Chapter 5.

Chapter 2

Theoretical background

In this chapter, we discuss the general theoretical background for fracture simulation. We start from basic deformation theory and governing equations. Then we talk about the MPM framework and MLS-MPM briefly. We further introduce the phase field method and show how to incorporate it into the general MPM framework. Finally, we describe the NURBS surface theory briefly.

2.1 Kinematics

2.1.1 Deformation gradient

In continuum mechanics, the research interest is an infinitely small element. The deformation can be represented as the change of the element's shape in two coordinate systems: the material coordinate and the world coordinate. They provide a useful measurement of any infinitely small element's position in the reference (undeformed) configuration and the current (deformed) configuration, respectively. In 3D domain Ω^3, the element's coordinate in the reference configuration and the current configuration are denoted as $\mathbf{X} = (X_1, X_2, X_3)^T$ and $\mathbf{x} = (x_1, x_2, x_3)^T$, respectively. We use $\mathbf{x} = \phi(\mathbf{X}, t)$ to denote the element's motion in two configurations where t is the time, e.g., Fig. 2.1.

The deformation tensor, $\mathbf{F}(\mathbf{X}, t) \in \mathbf{R}^{3\times 3}$, is derived by taking the derivative of ϕ w.r.t the material coordinate, which describes the Jacobian of the material at time t. In general, $\mathbf{F}$ is spatially varying across the domain.

$$\mathbf{F(X,t)} = \frac{\partial \phi}{\partial \mathbf{X}}(\mathbf{X}, t) = \frac{\partial \mathbf{x}}{\partial \mathbf{X}}(\mathbf{X}, t) \tag{2.1}$$

$\mathbf{F}(\mathbf{X}, t)$ can be written in matrix form:

$$\mathbf{F}(\mathbf{X},\mathbf{t}) = \frac{\partial(x_1,x_2,x_3)}{\partial(X_1,X_2,X_3)} = \begin{pmatrix} \frac{\partial x_1}{\partial X_1} & \frac{\partial x_1}{\partial X_2} & \frac{\partial x_1}{\partial X_3} \\ \frac{\partial x_2}{\partial X_1} & \frac{\partial x_2}{\partial X_2} & \frac{\partial x_2}{\partial X_3} \\ \frac{\partial x_3}{\partial X_1} & \frac{\partial x_3}{\partial X_2} & \frac{\partial x_3}{\partial X_3} \end{pmatrix} \tag{2.2}$$

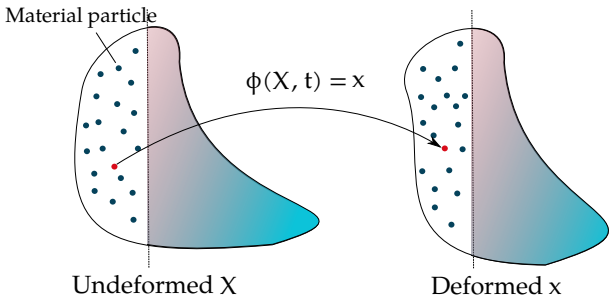


Figure 2.1: Deformation mapping.

2.1.2 Constitutive model

A constitutive model is a mathematical description of a material's behaviour under different loading conditions. It establishes the strain/stress relationship under loading. Commonly used constitutive models include linear elasticity, hyperelasticity, viscoelasticity and elastoplasticity. In this thesis, we use the hyperelastic model for most of our experiments. Note that, however, our method is not limited to hyperelasticity but can incorporate any constitutive model. For example, we use the elastoplasticity model to simulate the plastic deformation of a thin metal sheet in Fig. 4.20.

In particular, we use the nonlinear Neo-Hookean model (Wineman, 2005). Neo-Hookean is most commonly used isotropic hyperelastic material for predicting large deformation of elastic material like rubber. Unlike linear elastic materials, which follow Hooke's law and exhibit proportional stress-strain behavior, hyperelastic materials have nonlinear stress-strain relationship. The model is based on the strain energy density function, which characterizes the energy stored in the material due to deformation. The strain energy density of this model is

$$\Psi(\mathbf{F}) = \frac{\mu}{2}(tr(\mathbf{F}^T\mathbf{F}) - 3) - \mu\log(J) + \frac{\lambda}{2}\log^2(J) \tag{2.3}$$

where $tr(.)$ and $(.)^T$ are the trace and transpose of a matrix respectively, and $J = \det(\mathbf{F})$ is determinant of the deformation gradient which measures the volume change. μ and λ are constants related to material's Young's modulus E and Poisson's ratio ν.

$$\mu = \frac{E}{2(1+\nu)}, \quad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \tag{2.4}$$

2.1.3 Stress tensor

In mechanics, a stress tensor describes the magnitude and direction of internal force at each point in an infinitesimally small element. It is a second-order tensor that relates to the stress state of a physical system. In 3D space, the stress tensor is usually represented as a 3x3 matrix. Here we provide three most important stress tensors for our simulation.

1. **First Piola-Kirchoff (PK1) stress**: The First Piola-Kirchhoff stress tensor is also known as the nominal stress tensor, which relates the forces per unit reference area in the undeformed configuration to the corresponding forces per unit current area in the deformed configuration. It is the derivative of the strain energy function with respect to the deformation gradient.

$$\mathbf{P} = \frac{\partial \Psi}{\partial \mathbf{F}} \tag{2.5}$$

   For force computation, the first Piola-Kirchoff stress of the Neo-Hookean model is

$$\mathbf{P} = \mu(\mathbf{F} - \mathbf{F}^{-T}) + \lambda log(J)\mathbf{F}^{-T} \tag{2.6}$$

2. **Cauchy stress**: Cauchy stress is sometimes called true stress tensor because it is defined as the actual force per unit area acting on an infinitesimally small surface in the current deformed configuration. It is also derived from the strain energy density.

$$\boldsymbol{\sigma} = \frac{1}{J}\frac{\partial \Psi}{\partial \mathbf{F}}\mathbf{F}^T \tag{2.7}$$

3. **Kirchhoff stress**: The Kirchhoff stress is defined as the force per unit in the initial undeformed area. It takes into account the change in area due to deformation. The Kirchhoff stress relates to Cauchy stress through the determinant of deformation gradient.

$$\boldsymbol{\tau} = J\boldsymbol{\sigma} = \frac{\partial \Psi}{\partial \mathbf{F}}\mathbf{F}^T \tag{2.8}$$

2.1.4 Governing equations

The governing equations of physical simulation are conservation of mass and conservation of momentum. They state that the mass and momentum of a system remain constant without external input. Choosing an infinitely small element in the system, the conservation of mass and momentum can be written in strong form as:

$$\frac{D\rho}{Dt} + \rho\nabla\mathbf{v} = 0, \quad \rho\frac{D\mathbf{v}}{Dt} = \nabla.\boldsymbol{\sigma} + \rho\mathbf{g} \tag{2.9}$$

where $\frac{D}{Dt} = \frac{\partial}{\partial t} + \mathbf{v}.\nabla$ is the material derivative, ρ is the density, t is the time, and $\mathbf{v}$ is the velocity.

2.2 Material point method

MPM (Sulsky et al., 1995) generalizes the Particle in Cell (PIC) (Evans et al., 1957) and Fluid Implicit Particle Method (FLIP) (Brackbill et al., 1988) to solid mechanics. The method combines Lagrangian particles with Eulerian grids, precluding the need for Lagrangian mesh connectivity to simulate large deformation. Besides support for a breadth of materials and behaviours, MPM works well in handling topology change (e.g., fracture) and numerical stability. Thanks to this representation, MPM can realize automatic collision detection and contact treatment. Simulated materials have included snow melting and freezing (Stomakhin et al., 2013), complex fluids (Ram et al., 2015; Yue et al., 2015), sand (Klár et al., 2016), cloth with frictional contact (Jiang et al., 2017) and fracture simulations (Wretborn et al., 2017; Wolper et al., 2019, 2020; Wang et al., 2019a).

In MPM, the object is scattered as material particles and the background grid serves as a scratch pad where force and momentum are solved implicitly. The background grid is initialized and destroyed at the beginning and the end of each timestep. Any grid structure could be used in MPM algorithm. For simplicity, we use a fixed Cartesian grid in practice.

In this section, we summarize the MPM algorithm. We start from introducing the interpolation weight function. Then we describe the full traditional MPM scheme in one timestep. Finally, we briefly introduce the moving least-squares material point method (MLS-MPM) (Hu et al., 2018) which is an extension of the traditional MPM. In all our experiments, we use explicit time integration which is easy to implement though a larger timestep size is required than semi-implicit time integration scheme.

2.2.1 Weight function

In each timstep of the MPM simulation, the information is transferred between material particles and background grid by interpolation weight function. MPM requires the interpolation function to be local support. Furthermore, the weight function needs to be at least C^1 continuous to prevent cell-crossing error (Steffen et al., 2008). Quadratic B-splines function and cubic B-splines function satisfy the above requirements. Except otherwise stated, we use quadratic B-splines for its compacted support domain in almost all of our experiments.

The quadratic B-splines and its derivative are defined as:

$$N(x) = \begin{cases} \frac{3}{4} - |x|^2 & 0 \le |x| < \frac{1}{2} \\ \frac{1}{2}\left(\frac{3}{2} - |x|\right)^2 & \frac{1}{2} \le |x| < \frac{3}{2} \\ 0 & \frac{3}{2} \le |x|\,. \end{cases} \tag{2.10}$$

$$N^{'}(x) = \begin{cases} -sgn(x)2\,|x| & 0 \le |x| < \frac{1}{2} \\ -sgn(x)(\frac{3}{2} - |x|) & \frac{1}{2} \le |x| < \frac{3}{2} \\ 0 & \frac{3}{2} \le |x|\,. \end{cases} \tag{2.11}$$

The cubic B-splines and its derivative are defined as::

$$N(x) = \begin{cases} \frac{1}{2}|x|^3 - |x|^2 + \frac{2}{3} & 0 \le |x| < 1 \\ \frac{1}{6}(2 - |x|)^3 & 1 \le |x| < 2 \\ 0 & 2 \le |x|\,. \end{cases} \tag{2.12}$$

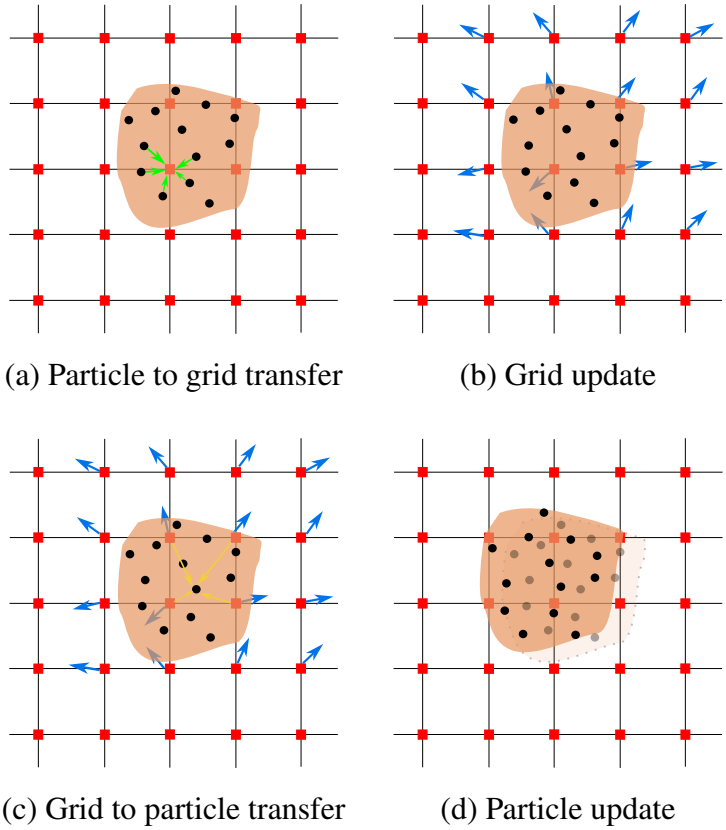
$$N^{'}(x) = \begin{cases} sgn(x)(\frac{3}{2}\,|x|^2 - 2\,|x|) & 0 \le |x| < 1 \\ -sgn(x)\frac{1}{2}(2 - |x|)^2 & 1 \le |x| < 2 \\ 0 & 2 \le |x|\,. \end{cases} \tag{2.13}$$

where $sgn(.)$ is the sign function. We use dyadic products of one-dimensional B-splines as the interpolation function in 3D.

$$N_{\mathbf{i}}^{h}(x_p) = N\left(\frac{1}{h}(x_p - ih)\right) N\left(\frac{1}{h}(y_p - jh)\right) N\left(\frac{1}{h}(z_p - kh)\right),$$

where $\boldsymbol{i} = (i, j, k)$ is the grid node ID, $(x_p, y_p, z_p) \equiv \mathbf{x}_p$ are the particle coordinates, and h is the respective grid spacing. The weight function gradient is also needed to compute the stress momentum contribution and update the deformation gradient, which can be simply calculated by taking the derivative of weight function w.r.t position as:

$$\nabla N_i(x_p) = \begin{pmatrix} \frac{1}{h}N'(\frac{1}{h}(x_p - x_i))N(\frac{1}{h}(y_p - x_j))N(\frac{1}{h}(z_p - x_k)) \\ N(\frac{1}{h}(x_p - x_i))\frac{1}{h}N'(\frac{1}{h}(y_p - x_j))N(\frac{1}{h}(z_p - x_k)) \\ N(\frac{1}{h}(x_p - x_i))N(\frac{1}{h}(y_p - x_j))\frac{1}{h}N'(\frac{1}{h}(z_p - x_k)) \end{pmatrix} \tag{2.14}$$



(a) Particle to grid transfer (b) Grid update

(c) Grid to particle transfer (d) Particle update

Figure 2.2: An overview of the full MPM algorithm. (a) Initialize a background grid. Particle's mass and momentum are then transferred to corresponding grid nodes using B-splines weight function. (b) Grid node's velocity is updated with the contribution from internal material particle's elastic force and external body force. (c) The particle's velocity are updated. (d) Update particle's position as well as deformation gradient in the next timestep.

2.2.2 Full scheme

Here we outline the update rule of full Symplectic Euler MPM algorithm. Each particle, p , has a mass m_p, velocity $\mathbf{v}_p$, initial volume V_p^0, position $\mathbf{x}_p$, deformation gradient $\mathbf{F}_p$ and material related parameters λ and μ. The grid node i's corresponding values are denoted with a subscript i. We show the algorithm in one timestep n which is denoted as a superscript in the values.

1. **Particle to grid transfer**: We first transfer particle mass and momentum to grid nodes via:

$$m_i^n = \sum_p m_p w_{ip}^n, \tag{2.15}$$

$$(m\mathbf{v})_i^n = \sum_p m_p w_{ip}^n \mathbf{v}_p^n, \tag{2.16}$$

where $w_{ip} = N(\mathbf{x}_p^n - \mathbf{x}_i)$ is the weight function of distance between an MPM particle $\mathbf{x}_p^n$ and a grid node with index i and position $\mathbf{x}_i$.

2. **Compute force contributions**: The grid node receives internal elastic force from material particles as well as body force from gravity and boundary conditions.

   (a) Compute the internal elastic force contribution from material particles.

$$\mathbf{f}_i^{int,n} = -\sum_p \frac{\partial \psi(\mathbf{F}_p^n)}{\partial \mathbf{F}_p^n} \mathbf{F}_p^{n,T} \nabla w_{ip}^n V_p^0 = -\sum_p J \boldsymbol{\sigma}_p^n \nabla w_{ip}^n V_p^0, \tag{2.17}$$

   where $\nabla w_{ip}^n = \nabla N(\mathbf{x}_p^n - \mathbf{x_i})$ is the weight gradient, $\psi(\mathbf{F}_p^n)$ is the elastic potential and $\boldsymbol{\sigma}_p^n$ is the particle's Cauchy stress.

   (b) Compute the external force contribution.

$$\mathbf{f}_i^{ext,n} = m_i^n \mathbf{g} + \mathbf{f}_i^{bd,n}, \tag{2.18}$$

   where $\mathbf{g}$ is the gravity acceleration and $\mathbf{f}_i^{bd,n}$ is the body force applied through boundary conditions.

3. **Grid update**: We update the grid node's velocity as:

$$\mathbf{v}_\mathbf{i}^{n+1} = (m\mathbf{v})_i^n / m_i^n + \Delta t (\mathbf{f}_i^{ext,n} + \mathbf{f}_i^{\mathrm{int},n}) / m_i^n, \tag{2.19}$$

   where $\Delta t = t^{n+1} - t^n$ is the timestep size.

4. **Grid to particle transfer**: The grid node's velocity is transferred back to particle as:

$$\mathbf{v}_p^{n+1} = \sum_i w_{ip}^n \mathbf{v}_\mathbf{i}^{n+1}, \tag{2.20}$$

5. **Particle update**: The particle's position is advected according to the updated velocity:

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1} \tag{2.21}$$

6. **Update particle's deformation gradient**: Finally, particle's deformation gradient is updated according to the updated grid velocity:

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \sum_i \mathbf{v}_i^{n+1} (\nabla w_{ip}^n)^T) \mathbf{F}_p^n \tag{2.22}$$

2.2.3 Moving least-squares material point method

Traditional MPM has many limitations. The original PIC formulation is stable but dissipative. The FLIP variation is more stable but noisy. Jiang et al. (2015) propose affine particle in cell method (APIC) which augments each particle a local affine during transfer between particle and grid. This formulation effectively conserves particle's angular momentum during transfer and achieves a more stable and dynamic simulation result. Based on APIC, Hu et al. (2018) proposed the MLS-MPM which replaces the B-spline shape function with MLS approximation. The derivation is further used in the stress divergence term and achieves a faster implementation. We use MLS-MPM in the brittle fracture simulation project. The main adaption of this method is summarized here. Readers are referred to Hu et al. (2018) for more details.

MLS-MPM store one additional term $\mathbf{C}_p^n$ called affine at each material particle. The affine helps to keep angular momentum information. It is initialized as an identity matrix at the begining of simulation and is updated at the end of each timestep. Compared with traditional MPM, MLS-MPM only makes three modifications to the full scheme, i.e., Step 1, 2 and 6 in Sec. 2.2.2.

In Step 1, the momentum transferred to grid node, i.e., Eq. 2.16, is:

$$(m\mathbf{v})_i^n = \sum\nolimits_p m_p w_{ip}^n (\mathbf{v}_p^n + \mathbf{C}_p^n (\mathbf{x}_p^n - \mathbf{x_i})), \tag{2.23}$$

In Step 2, the internal elastic force, i.e., Eq. 2.17, can be computed as:

$$\mathbf{f}_i^{int,n} = -\sum_p M_p^{-1} \frac{\partial \psi(\mathbf{F}_p^n)}{\partial \mathbf{F}_p^n} \mathbf{F}_p^{n,T} (\mathbf{x}_p^n - \mathbf{x_i}) w_{ip}^n V_p^0 = -\sum_p M_p^{-1} J \boldsymbol{\sigma}_p^n (\mathbf{x}_p^n - \mathbf{x_i}) w_{ip}^n V_p^0, \tag{2.24}$$

where M_p is $h^2/4$ for quadratic B-spline.

In Step 6, the particle's affine term is updated as:

$$\mathbf{C}_p^{n+1} = M_p^{-1} \sum_i w_{ip}^n \mathbf{v}_i^{n+1} (\boldsymbol{x}_i^n - \mathbf{x}_p^n)^T, \tag{2.25}$$

Then the particle's deformation gradient, i.e., Eq. 2.22, is updated using C_p^{n+1} as:

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \mathbf{C}_p^{n+1}) \mathbf{F}_p^n, \tag{2.26}$$

2.3 Phase field method

Phase field method belongs to the family of CDM (Gurson, 1977). Here, new internal state variables (termed "damage variables") are introduced into the constitutive models, where deformation and failure behaviour are coupled. Material damage is treated as a continuum where mechanical properties are weakened based on the evolving stress. PF method adopts a scalar field to track the evolution of cracks. The scalar ranges from 0 to 1 where 0 denotes healthy intact materials, and 1 denotes fully damaged material. By tracking the crack propagation implicitly, PF method can handle complex merging and bifurcation fracture behaviours. This effectively avoids costly and potentially unstable geometry operation. PF method has been applied to fracture simulation with MPM in many engineering papers, such as (Homel and Herbold, 2017; Kakouris and Triantafyllou, 2019; Zhang et al., 2022). There are also recent research papers in computer graphics (Wolper et al., 2019; Wang et al., 2020a; Zhao et al., 2020).

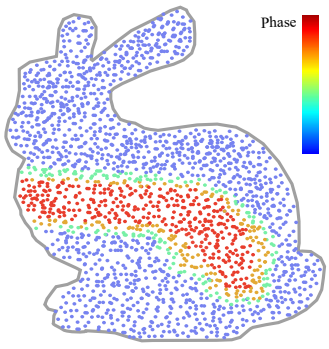


Figure 2.3: Crack is represented as a continuous phase field. The color represents the phase c, where blue dots ($c = 0$) denote healthy intact materials and red dots ($c = 1$) denote fully damaged materials. The other colors represent partially damaged materials.

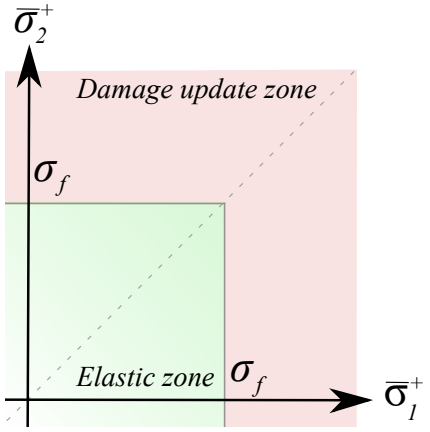


Figure 2.4: Simple 2D illustration of the Rankine damage model in the principal stretch space. Green shade: principal stress is less than the threshold σ_f, thus material stress purely a function of the current deformation gradient $\mathbf{F}$ and material parameters. Red shade: the principal stress exceeds the threshold σ_f, which leads to a material weakening through damage (see Eq. 2.28).

There are two variations of PFM method. The non-local phase-field fracture (Wolper et al., 2019; Kakouris and Triantafyllou, 2017; Borden et al., 2016) tracks a scalar field of damage variables evolving over time. This method, however, requires an additional linear system, iterative solvers as well as an increased number of parameters that are difficult to tune. Alternatively, the local phase-field formulation (Cervera and Chiumenti, 2006) evaluates the ratio of a defined maximal stress and the current local stress. Consequently, the smeared crack can be efficiently generated and with fewer parameters that are also easier to tune. With the local phase field formulation, only the Cauchy stress of Eq. 2.17 / Eq. 2.24 in the original MPM framework is required, resulting in a weakening of stress in the damaged regions. Take the conventional MPM for example:

$$\mathbf{f}_i^{int,n} = -\sum_p J\boldsymbol{\sigma}(\mathbf{F}_p^n, c)\nabla w_{ip}^n V_p^0, \tag{2.27}$$

where $c \in \mathbf{R}$, $0 \leq c \leq 1$ ($0 \equiv$ *undamaged* and $1 \equiv$ *damaged*) is the phase/damage variable that parameterizes stress. Whenever damage is increased, the particle stress $\boldsymbol{\sigma}_p^n = \boldsymbol{\sigma}(\mathbf{F}_p, c)$ is weakened (Sec. 2.3.2), but there is no additional transfer of information from particles to grid nodes besides the usual mass and velocity.

2.3.1 Updating particle damage

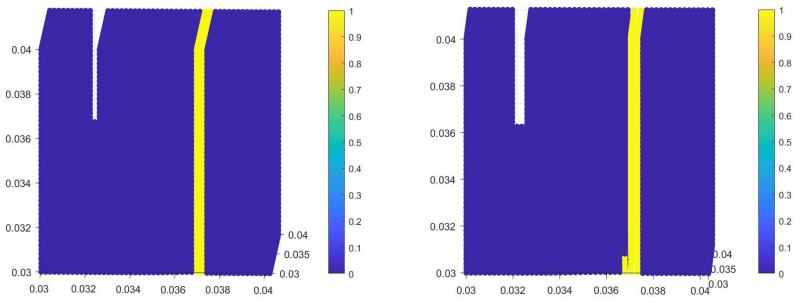
Evaluating Eq. 2.27 leads to further stress and a requirement to update damage c in the case of excessive loading. Our observation is that this configuration lends itself

to a fast evaluation of the isotropic Rankine damage model via linear strain-softening, *cf.* Eq. 10 in (Cervera and Chiumenti, 2006)

$$c = \begin{cases} (1+\mathcal{H})\left(1-\frac{\sigma_f}{\overline{\sigma}}\right), & \sigma_f \leq \overline{\sigma} \leq \sigma_f\left(1+\frac{1}{\mathcal{H}}\right) \\ 1, & \sigma_f\left(1+\frac{1}{\mathcal{H}}\right) \leq \overline{\sigma} \end{cases} \tag{2.28}$$

For a given particle, the effective stress $\overline{\sigma} \in \mathbf{R}$ is evaluated (refer to Sec. 2.3.2 for detail) and compared against the maximum principal failure stress σ_f to compute the new damage state. The term $\mathcal{H} = \overline{\mathcal{H}} l^{ch}/(1-\overline{\mathcal{H}} l^{ch})$ is the brittleness factor which is a constant, where $\overline{\mathcal{H}} = \sigma_f^2/2EG_f$. The additional terms E, G_f, and $l^{ch} = \sqrt{3h^2}$ denote Young's Modulus, Mode I fracture energy, and the characteristic length, respectively (Homel and Herbold, 2017). An illustrative example is also shown in Fig. 2.4.

We note that there exist other techniques for updating damage from the effective stress, such as exponential softening (Cervera and Chiumenti, 2006) (and non-local PFF itself), but linear strain-softening is a sufficient model for defining a criterion for crack propagation, and automatically selecting the direction of crack propagation as a function of the stored elastic energy as shown in Fig. 2.5.



(a) Linear softening (b) Exponential softening

Figure 2.5: A simple test using linear strain softening and exponential strain softening. They get quantitatively the same results, i.e., crack propagation speed and propagation direction.

2.3.2 Effective Stress weakening

Having outlined the damage update rule given the local effective stress at a material point in the previous section, we now describe the details for 1) computing this effective stress, 2) invoking the update rule, and 3) determining the real (weakened) stress.

We update damage according to the hypothesis of strain equivalence: *The strain associated with a damaged state under the applied stress* $\boldsymbol{\sigma}$ *is equivalent to the strain*

associated with its undamaged state under the effective stress $\overline{\boldsymbol{\sigma}}$ (*cf.* (Lemaitre and Chaboche, 1978; Cervera and Chiumenti, 2006; Choo and Sun, 2018)).

Therefore, to update damage we first compute the effective stress tensor $\overline{\boldsymbol{\sigma}} \in R^{3\times 3}$, meaning that we assume for a moment that there is no damage as we evaluate Eq. 2.7 using the current state variables. With $\overline{\boldsymbol{\sigma}}$, we perform Singular Value Decomposition (SVD) from which we set the effective stress $\overline{\sigma} = \overline{\sigma}_1$ as the largest eigenvalue, and proceed to evaluate Eq. 2.28 which yields c^{n+1} as the new damage state.

The real stress $\boldsymbol{\sigma}$ is calculated with our updated damage variable c^{n+1}, which we use to evaluate our elastic forces (*cf.* Eq. 2.27). To compute this real (weakened) stress, we first split the effective stress $\overline{\boldsymbol{\sigma}}$ into a tensile part $\overline{\boldsymbol{\sigma}}^{+}$ and a compressive part $\overline{\boldsymbol{\sigma}}^{-}$, using principal stress directions (*cf.* Eq. (3) in (Cervera and Chiumenti, 2006))

$$\overline{\boldsymbol{\sigma}}^{+} = \sum_{i=1}^{3} \langle \overline{\sigma_i} \rangle \boldsymbol{q}_i \otimes \boldsymbol{q}_i, \qquad \overline{\boldsymbol{\sigma}}^{-} = \overline{\boldsymbol{\sigma}} - \overline{\boldsymbol{\sigma}}^{+}. \tag{2.29}$$

The notation $\otimes$ is the dyadic product of two vectors, and we have also used $\langle \cdot \rangle$ to denote the Macauley brackets

$$\langle x \rangle = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}.$$

The term $\overline{\sigma_i}$, $\overline{\sigma_i} > \overline{\sigma}_{i+1}$ in Eq. 2.29 is the ith eigenvalue of the effective stress $\overline{\boldsymbol{\sigma}}$, with $\boldsymbol{q}_i$ being the corresponding eigenvector from SVD. Thus, a weakening of the stress is achieved through a scaling of the tensile part of the effective stress $\overline{\boldsymbol{\sigma}}^{+}$ by

$$\sigma_i = \begin{cases} (1-c)\overline{\sigma_i} & \overline{\sigma_i} > 0 \\ \overline{\sigma_i} & \overline{\sigma_i} \leq 0 \end{cases}. \tag{2.30}$$

We combine this weakened tensile part and the (unchanged) compressive part to give

$$\boldsymbol{\sigma} = \sum_{i=1}^{3} \sigma_i \boldsymbol{q}_i \otimes \boldsymbol{q}_i, \tag{2.31}$$

as the final stress that we used to evaluate Eq. 2.27.

2.4 Basics of NURBS

This section briefly describes some basic terminologies about NURBS surface and the corresponding basis functions. A more detailed explanation can be found in a book (Piegl and Tiller, 1996).

NURBS is built from B-spline. In 1D case, a B-spline can be defined by a knot vector and a set of control points. The knot vector is a sequence of non-decreasing coordinates in the parameter space ξ: $\Xi = \{\xi_1 = 0, \xi_2, ..., \xi_i, ..., \xi_{n+p+1} = 1\}$, where ξ_i is called *knot*, n is the number of control points and p is the order of the B-spline representing the smoothness. The open knot vector has a multiplicity of $p+1$ at the first and last knot. Basis function $N_{i,p}(\xi)(i = 1, 2, ..., n)$ corresponding to each control point can defined recursively using Cox–deBoor recursion formula (Cox, 1972):

If $p = 0$:

$$N_{i,0}(\xi) = \begin{cases} 1, & \xi_i \leq \xi < \xi_{i+1} \\ 0, & \xi < \xi_i \quad || \quad \xi \geq \xi_{i+1} \end{cases} \tag{2.32}$$

If $p \geq 1$:

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi) \tag{2.33}$$

The basis function $N_{i,p}$ is non-negative and has a compact support radius of $\xi \in [\xi, \xi_{i+p+1}]$. A knot interval $[\xi_i, \xi_{i+1}]$ is usually defined as an element or a cell in the parameter space, and thus $N_{i,p}$ spans $p+1$ cells. For example, a quadratic ($p = 2$) B-spline basis spans three cells and a cubic ($p = 3$) B-spline basis spans four cells, respectively. Note that we use quadratic B-spline in all our experiments in this thesis although the extension to higher order B-spline is straightforward. Fig. 2.6 shows a quadratic curve defined by five control points.

The first order derivative of the basis function is:

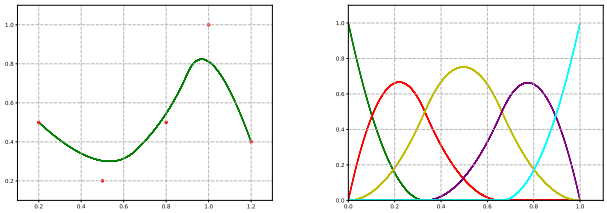
$$N'_{i,p}(\xi) = \frac{p}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi) \tag{2.34}$$

Higher order derivatives can be defined recursively. Eq. 2.35 gives the expression of the kth order derivative:

$$N^k_{i,p}(\xi) = \frac{p}{\xi_{i+p} - \xi_i} N^{k-1}_{i,p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}} N^{k-1}_{i+1,p-1}(\xi) \tag{2.35}$$

By defining two knot vectors in the ξ direction $\Xi_1 = \{\xi_1 = 0, \xi_2, ..., \xi_i, ..., \xi_{n+p+1} = 1\}$ and in the η direction $\Xi_2 = \{\eta_1 = 0, \eta_2, ..., \eta_i, ..., \eta_{m+p+1} = 1\}$, respectively, a 2D B-spline surface is computed as:

$$\mathbf{BS}(\xi, \eta) = \sum_{i=1}^{n} \sum_{j=1}^{m} N_{i,p}(\xi) M_{j,q}(\eta) \mathbf{P}_{i,j} \tag{2.36}$$



(a) (b)

Figure 2.6: NURBS curve and control points' basis functions. The knot vector is [0.0,0.0,0.0,0.33,0.66,1.0,1.0,1.0]. (a) The red points are control points, and the green curve is the second-order NURBS curve. As an open knot vector, the first and last value has a multiplicity of 3. (b) Basis functions of the control points. Each basis function spans three cells.

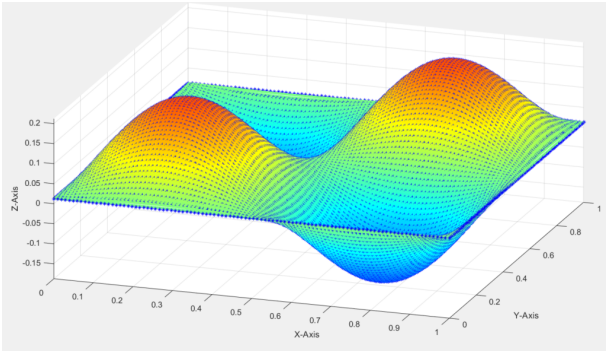


Figure 2.7: A 2D NURBS surface. * denotes control points. The color represents the height of the NURBS surface.

where $N_{i,p}(\xi)$ and $M_{j,q}(\eta)$ are 1D basis functions of order p and q, respectively, and $\mathbf{P}_{i,j}$ are control points' positions.

A NURBS surface can be constructed by assigning each control point a weight as:

$$\mathbf{NS}(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} R_{i,j}(\xi,\eta)\mathbf{P}_{i,j} \tag{2.37}$$

with

$$R_{i,j}(\xi,\eta) = \frac{N_{i,p}(\xi)M_{j,q}(\eta)\omega_{i,j}}{\sum_{I=1}^{n}\sum_{J=1}^{m} N_{I,p}(\xi)M_{J,q}(\eta)\omega_{I,J}} \tag{2.38}$$

where $\omega_{i,j}$ are weights. By definition, this set of basis functions satisfy partition of unity condition. Fig. 2.7 shows a second-order NURBS surface constructed by 100x100 control points.

Chapter 3

Brittle object fracture

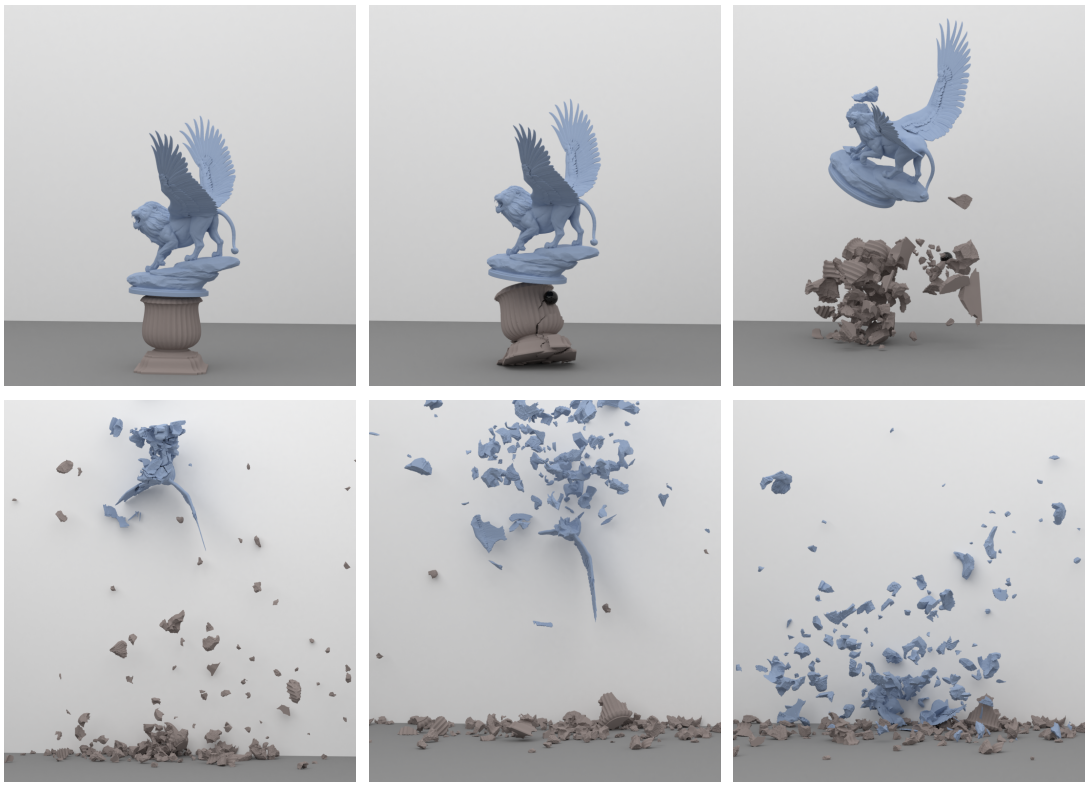


Figure 3.1: We offer a *mesh-free* crack propagation method using phase field method with locally defined damage updates. This is achieved by combining a novel adaptation of MPM, and Voronoi tessellations to define thin embeddings of geometry on smeared crack paths. The figure shows a winged lion on a column, both of which are smashed into pieces after colliding with a billiard ball.

3.1 Introduction

The depiction of detailed destruction in objects that are composed of highly-stiff material such as brick, concrete and mortar is highly desirable for visual effects in the game and film industry. The realism of scenes where buildings collapse with explosions, windows blasting, and monsters destroying bridges can greatly affect the immersion of the viewer and thus their impression about the entire spectacle. As a result, many researchers in computer animation have been developing techniques to simulate such scenes.

Highly-stiff fracture is notoriously difficult to simulate robustly and accurately in practice. In mesh-based methods, the popular FEM supports fracturing of brittle material but domain remeshing and element refinement are necessary which will compromise practical use and visual results. XFEM overcomes issues of remeshing but has difficulties with robustness in partitioned elements due to basis enrichment. Alternatively, boundary element methods offer renowned visual quality (see e.g., Hahn and Wojtan (Hahn and Wojtan, 2015)) but are ill-suited for problems with large surface-area-to-volume ratios like thin glass.

Continuum damage mechanics (CDM) offers another approach. Here, new internal state variables (termed "damage variables") are introduced into the constitutive equations, where deformation and failure behaviour are coupled. Material damage is treated as a continuum where mechanical properties are weakened based on the evolving stress.

In this chapter, we propose an approach to simulating complex brittle fracture with a rigid-body-to-CDM coupling. We delegate crack propagation to CDM, simulating fracture upon detecting an excessive rigid-body impact. We offer an implementation using *local* CDM discretized using the MPM, which is simple, practical, and offers superior crack bifurcation and merging behaviour than previous mesh-based methods. During propagation, we mitigate excessive thickening of the smeared crack path by introducing a novel scheme to model velocity discontinuity. We achieve this using an explicit surface which we extract from the smeared crack path using a novel technique based on Voronoi tessellations. Our results also show that our approach is able to propagate cracks in-line with all three loading modes of fracture analysis, with few changes to our MPM discretization. The simulation result of our method is competitive with, and typically better than, existing methods for simulating brittle fracture.

We are also inspired by methods that model fracture-surface detail through their

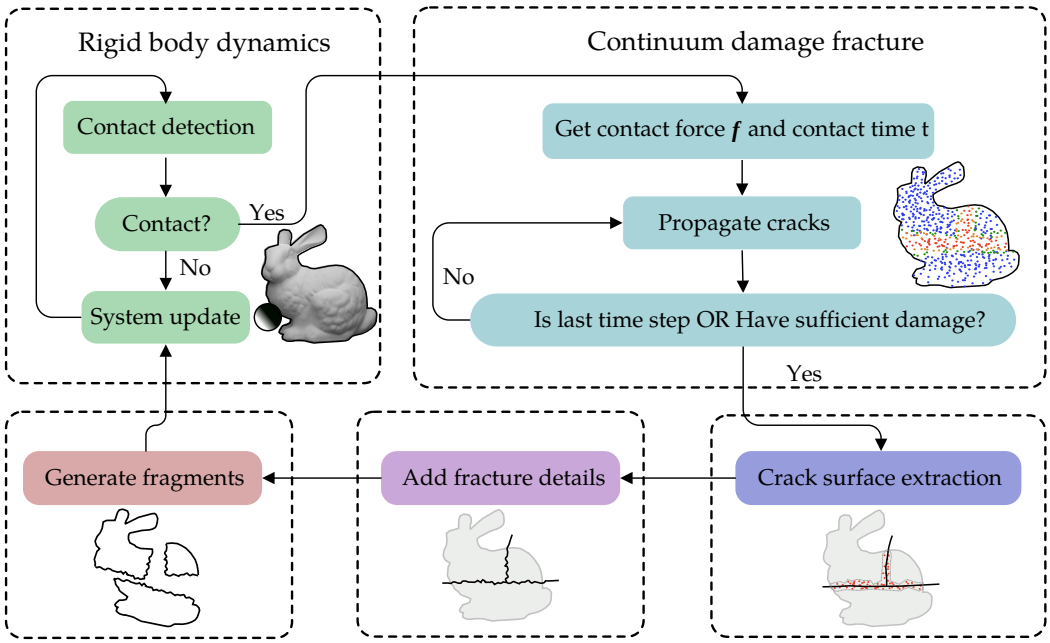


Figure 3.2: Schematic overview of our simulation framework.

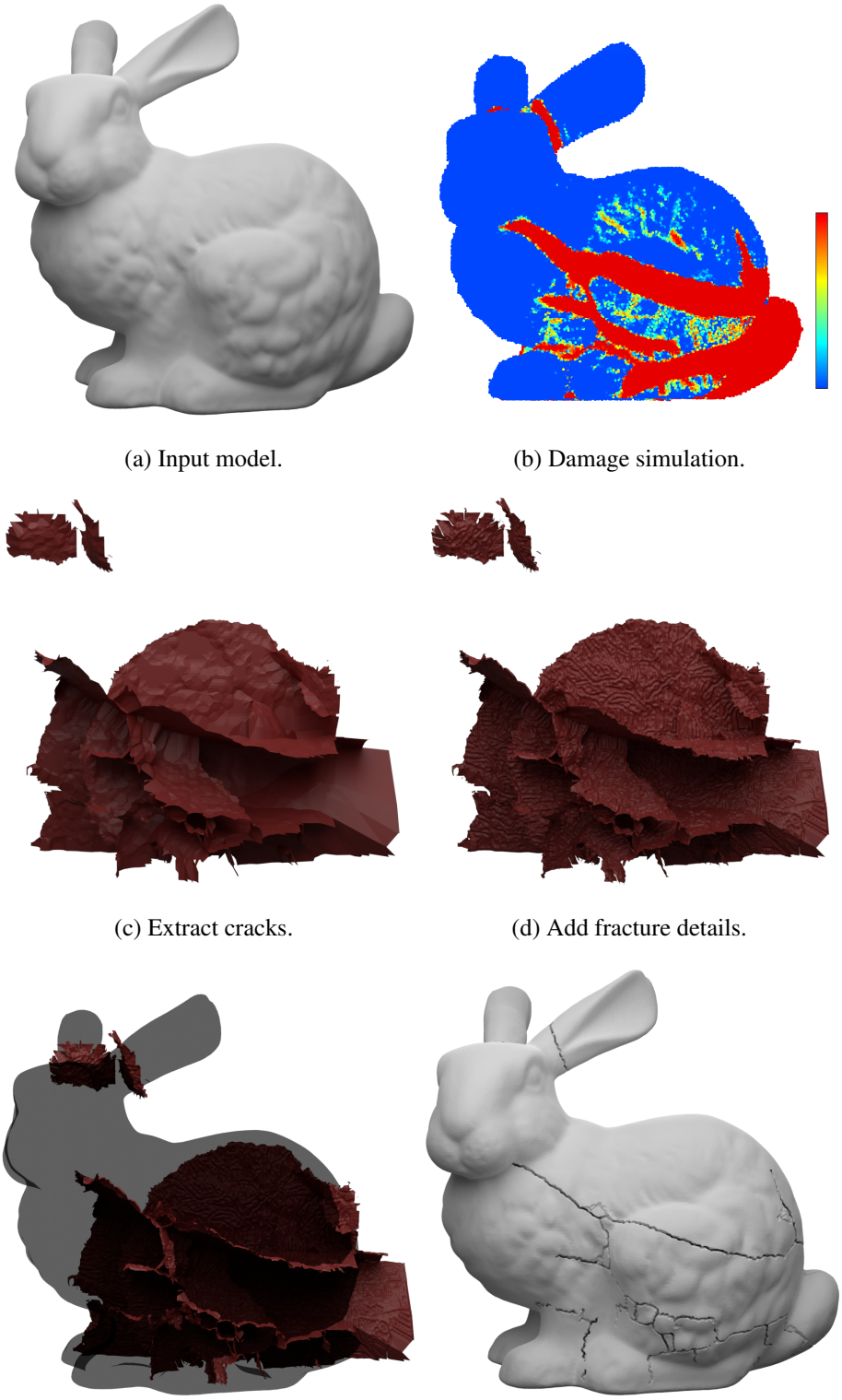
treatments of spatially varying toughness (Hahn and Wojtan, 2015; Chen et al., 2014), and propose a new detail enrichment technique that enables crack surfaces to emulate complex material grain-structure when paired with our damage variables.

We show that our method can produce a wide range of complex fracture effects with different objects and materials. Our system can manage large scale scenes with many objects colliding and breaking into pieces, thanks to the efficiency of the hybrid rigid-body-to-CDM coupling.

3.2 Method overview

In this section, we outline our brittle fracture simulation method. An illustrative overview is also provided in Fig. 3.2.

Given a high-resolution mesh, it is first decimated for collision purposes (Sec. 3.6.1) followed by rigid body simulation (Sec. 3.6.3) although any other rigid body solver can be used with appropriate modifications. When a rigid body object collides, we construct external contact forces as boundary conditions from impulses by using the Hertzian contact model following Glondu *et al.* (Glondu et al., 2013). These contacts are applied as concentrated point-forces–defined in object-space–using ghost particles.



(a) Input model. (b) Damage simulation.

(c) Extract cracks. (d) Add fracture details.

(e) Overlay. (f) Fragments after cutting.

Figure 3.3: A visual summary of the stages to fracture an object. (a) A high resolution Stanford Bunny model. (b) One million particles with excessive damage after simulation (Sec. 2.3). The red region is where material particles are fully damaged. (c) Resulting approximation of the potentially non-manifold crack surface(s) (Sec. 3.3). (d) Modelling of fracture-detail patterns (Sec. 3.7). (e) Visual aide depicting relative crack placement. (f) Crack manifestation on the input model. Note: the partial cracks (as represented by thin gaps in (f)) are only used for demonstration purposes.

To start fracture simulation, we sample particles inside the colliding object's volume, using Poisson disk sampling (Bridson, 2007), which is necessary to solve for our continuum damage cracks with MPM. We assume the velocity of all sampled particles to be 0 since we are performing a quasi-static simulation. The global velocity of the particles doesn't impact the crack propagation. Using a time-dynamic simulation, we then apply boundary conditions to update particle deformation gradients, stresses and damage using MLS-MPM (Sec. 2.2.3). Based on the material particles' stresses, we update the damage if the principal stress exceeds the failure threshold (Sec. 2.3). Continuum crack propagation stops if a prescribed number of damaged particles is exceeded, or the number of fracture time steps is reached to terminate propagation.

Crack surface extraction proceeds as shown in Fig. 3.3 with further detail in Sec. 3.3. At a given time step we: 1) Isolate fully-damaged particles; 2) Find the implicit surface(s) bounding these particles to create the crack volume(s); and 3) Tessellate these volume(s) to determine the explicit crack surface(s). We use these surfaces to enforce displacement discontinuities using two velocity fields (Sec. 3.4), and to model spatially varying toughness by adding fracture detail (Sec. 3.7). We also compute fragment geometry (Sec. 3.5), using the crack surface(s) which is added into the rigid body system as new objects.

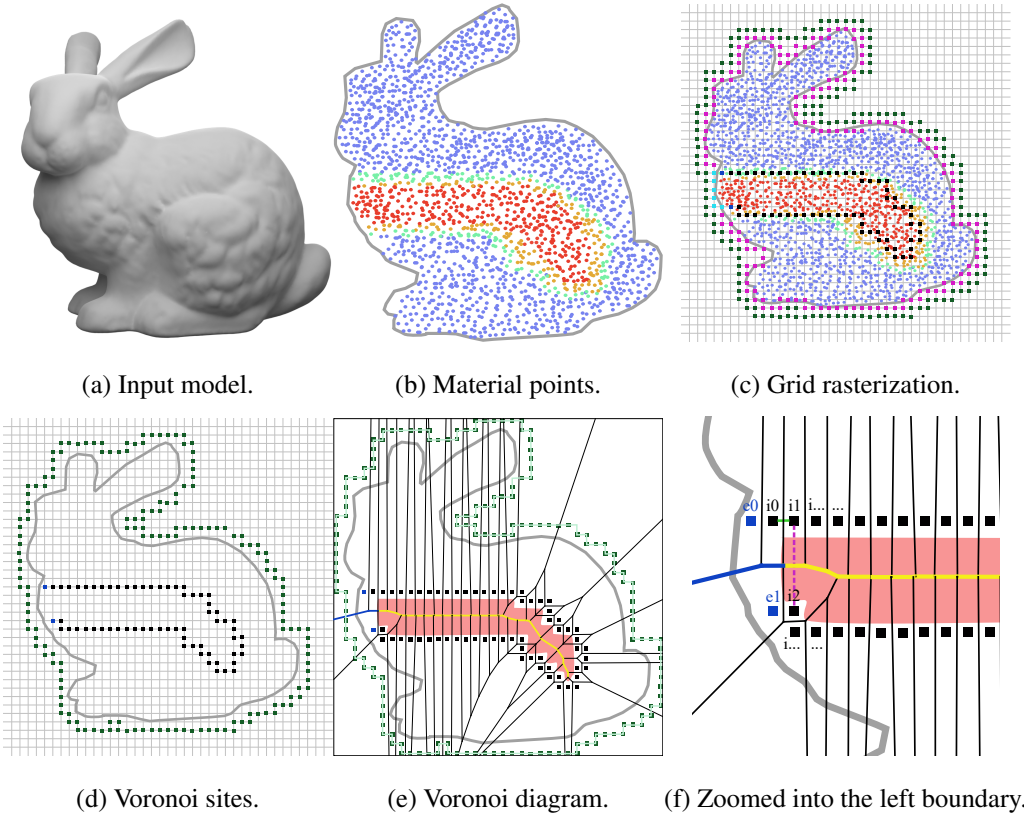
3.3 Extract non-manifold crack surface from damage

Material damage will span expanses of perceptively thick volume, which introduces ambiguity when one wishes to place a thin surface to e.g., treat material separation or estimate crack orientation.

A naïve solution for splitting material is particle deletion but the resulting artefacts, which include volume loss, leave much to be desired. Pre-scoring (Wang et al., 2019a) is also infeasible because 1) it may produce numerous unintended fragments if all mesh edges are marked as 'failed edges' as shown in Fig. 10 and Fig. 12 in (Wang et al., 2019a); and 2) paths between the connected 'failed edges' in a single damaged region are not guaranteed to split a crack path along a single surface, which is desirable. In this section we describe our solution to the problem of disambiguating a surface from damaged particles, which is useful for separating material.

Gist: Apply a Voronoi tessellation (Klein, 1989) of the damaged-particle region, and treat the resulting embedded medial-surface as an approximation of the crack surface.

An illustrative overview is provided in Fig. 3.4.



(a) Input model. (b) Material points. (c) Grid rasterization.

(d) Voronoi sites. (e) Voronoi diagram. (f) Zoomed into the left boundary.

Figure 3.4: An overview of how we extract a crack surface from the set of fully damaged particles (see also Tab. 3.1): (a) Input model. (b) Material points. (c) Rasterizing material point properties onto a grid, where nodes are classified according the region of the domain that they surround. The black, blue and cyan nodes are the set $\mathcal{D}$ which surround the fully damaged particles. Black I and blue nodes $\mathcal{E}$ are used to compute the Voronoi diagram while the cyan nodes $\mathcal{S}$ represent the damaged domain-boundary. Pink nodes $\mathcal{B}$ represent the domain boundary, while the green nodes $\mathcal{B}'$ are the dilated domain-boundary which are also used to compute the Voronoi diagram. (d) Collectively, the set of nodes $\mathcal{B}' \cup \mathcal{V}$, $\mathcal{V} = I \cup \mathcal{E}$ are used to compute Voronoi diagram and clip Voronoi cells. (e) The tessellated domain and embedded crack surface (yellow and blue line). (f) Determining the side of the medial surface (blue and yellow line) on which the blue squares (nodes) lie by deducing from their nearest neighbour that is colored black.

Upon rasterising material particles onto a grid, we categorize the nodes of this grid into sets that define the sites (input points) of the Voronoi tessellation, and the bounding surface of the domain. The resulting Voronoi diagram has Voronoi cells which are clipped using a dilated bounding surface of the domain. We then extract the Voronoi faces corresponding to the medial-surface of the *fully* damaged particle region as our crack surface. A list of important set notations used in this section can be found in Tab. 3.1.

Set	Definition	Description
$\mathcal{D}$	$\longrightarrow$	Fully-damaged particle boundary.
$\mathcal{B}$	$\longrightarrow$	Domain boundary.
$\mathcal{B}'$	$\longrightarrow$	Dilated domain-boundary.
$\mathcal{S}$	$\mathcal{D} \cap \mathcal{B}$	Damaged domain-boundary.
$\mathcal{V}$	$\mathcal{D} \backslash \mathcal{S}$	Tessellation input.
I	$\mathcal{V} \backslash \mathcal{E}$	Interior mid-surface nodes (Sec. 3.3.3).
$\mathcal{E}$	$\mathcal{V} \cap \mathcal{N}$	Mid-surface extension nodes.
$\mathcal{N}$	Sec. 3.3.4	Neighbours of nodes $\in \mathcal{S}$

Table 3.1: A summary of the set notation that is used to classify grid nodes in our crack surface extraction method.

3.3.1 Rasterizing particles onto the grid

We construct a grid with grid-spacing h_v, $h_\mathrm{v} > \bar{s}_\mathrm{p}$, where $\bar{s}_\mathrm{p}$ is the average particle spacing [1]. Particles (in undeformed material space) are then rasterized onto this grid to identify the sets of nodes surrounding distinct regions of the domain by

$$u_\mathbf{i} = \sum\nolimits_\mathrm{p} w_\mathbf{ip} u_\mathrm{p}, \tag{3.1}$$

where $u_\mathbf{i}$ is the interpolated value at node $\mathbf{i}$, $w_\mathbf{ip}$ is the weight, and u_p is any interpolated quantity of particle p, such as damage. The same kernel function $w_\mathbf{ip}$ is used as in our MPM simulation (*cf.* Sec. 2.2), which implies that the rasterized crack path volume thickness will be at-least $3h_\mathrm{v}$ because this kernel has an influence radius of $1.5h_\mathrm{v}$.

Defining region boundaries on the grid is equivalent to finding the outermost of active nodes ($u_\mathbf{i} > 0$). A node is on the boundary if at least one neighbour is inactive ($u_\mathbf{i} = 0$).

3.3.2 Computing the Voronoi diagram

Our medial surface is an approximation from the 3D Voronoi diagram, representing a subset of Voronoi faces whose respective Voronoi cells correspond to the grid nodes $\mathcal{V} = \mathcal{D} \backslash \mathcal{S}$ that *partially* surround the region of fully damaged particles (*cf.* Fig. 3.4 (c)). Briefly, the grid nodes $\in \mathcal{S}$ are those that define the damaged parts of the domain boundary (see also: Sec. 3.3.4 for further detail). An illustration of $\mathcal{V}$ is also shown in Fig. 3.4 (d) and (e) as black and blue nodes.

[1] It is possible to re-use our MPM grid (Sec. 2.2), in which case $h_\mathrm{v} = h$

Algorithm 1: Select internal mid-surface Voronoi faces

```
1  for gridNodeID ∈ I do
2      nodeVoroCell ← GetVoroCell(gridNodeID);
3      for neighbourVoroCell ∈ nodeVoroCell.GetNeighVoroCells() do
4          isInSet ← neighbourVoroCell.gridNodeID ∈ I;
5          areAdjacent ← TaggedAsAdjacents(gridNodeID,
             neighbourVoroCell.gridNodeID);
6          areOpposites ← TaggedAsOpposites(gridNodeID,
             neighbourVoroCell.gridNodeID);
7          if isInSet ∧ !areAdjacent ∧ !areOpposites then
8              midPoint ← GetMidPoint(nodeVoroCell, neighbourVoroCell);
9              damage ← SampleGrid(midPoint);
10             if damage == 1 then
11                 SaveSharedFace(nodeVoroCell, neighbourVoroCell);
12                 TagAsOpposites(nodeVoroCell, neighbourVoroCell);
13             else
14                 TagAsAdjacents(nodeVoroCell, neighbourVoroCell);
15             end
16         end
17     end
18 end
```

Outer Voronoi cells are additionally clipped using a surface of the dilated-boundary nodes $\mathcal{B}'$ (*cf.* Fig. 3.4 (e), outer boundary green nodes) to prevent the medial-surface from re-entering our object's volume when the shape is concave.

With a clipped Voronoi diagram, the crack surface(s) can thus be identified as the list of Voronoi faces that represent the medial surface(s) of each region of damaged particles, which we describe in the following paragraphs.

3.3.3 Identifying interior crack faces

The computed Voronoi diagram (Sec. 3.3.2) consists of polygons that we call Voronoi faces, where each such face is equidistant to two grid nodes $\in \mathcal{V}$ that correspond to the part of the damaged-region boundary that is inside our object (i.e., $\mathcal{V}$ excludes grid nodes corresponding to our object's surface). To select the Voronoi faces that define planar patches of the medial surface, we first identify a subset of grid nodes $I \subset \mathcal{D}$, $I \notin \mathcal{B}$ that were part of the input to the Voronoi tessellation (*cf.* Fig. 3.4 (e), black nodes). In this subset, we access each node's corresponding Voronoi cell and the

Algorithm 2: Search nearest node in I

```
   input : gridNodeID, I
   output: Nearest grid node ∈ I to gridNodeID
 1 /* search immediate neighbours on grid to gridNodeID                    */
 2 adjoiningGridNodesIDs ← ...
 3 adjoiningGridNodesIDsInSet ← ∅
 4 for adjoiningGridNodeID in adjoiningGridNodesIDs do
 5 |   if adjoiningGridNodeID ∈ I then
 6 |   |   adjoiningGridNodesIDsInSet.Add(adjoiningGridNodeID)
 7 |   end
 8 end
 9 result ← NULL
10 /* gridNodeID may be surrounded by nodes ∈ I                           */
11 if adjoiningGridNodesIDsInV.IsEmpty() then
12 |   result ← BFSFindNearestInSet(gridNodeID, I)
13 else
14 |   result ← adjoiningGridNodesIDsInSet.Any()
15 end
16 return result
```

neighbouring Voronoi cells to get their shared face. On this face, we determine whether it belongs to the medial surface if-and-only-if the midpoint between the respective grid nodes is located in a fully damaged region, which we determine by interpolating simulated damage values on the grid using Eq. 3.1 (see also: Line (9) and Line (11) in Algo. (1)). To prevent adding duplicate faces, we only visit faces that have not been tagged (i.e., marked as ‘adjacent’ or ’opposite’).

3.3.4 Identifying extending crack faces

We now describe how we select the remaining Voronoi faces which extend the medial surface to the domain boundary to allow for material separation (*cf.* the blue edges in Fig. 3.4 (e)).

As our problem is concentrated on areas where the grid nodes that surround the fully-damaged material particles are next to our object’s surface, we bring to focus the grid nodes

$$\mathcal{E} = \mathcal{V} \cap \mathcal{N}, \tag{3.2}$$

Algorithm 3: Check if two nodes $\in \mathcal{E}$ are adjacent

```
ø          input : nodeVoroCell, neighVoroCell, E, I
           output: Boolean: TRUE if nodeVoroCell on same side neighVoroCell
/* find any nearest using Algo. (2)                                          */
startNodeID ← FindAnyNearestNodeInSet(nodeVoroCell.gridNodeID, I)
endNodeID ← neighVoroCell.gridNodeID
if neighVoroCell.gridNodeID ∈ E then
    endNode ← FindAnyNearestNodeInSet(neighVoroCell.gridNodeID, I)
end
/* current node ∈ E and neighbour share a neighbour ∈ I                      */
isAdjacent ← startNodeID == endNodeID
if isAdjacent then
    return true
end
/* retrieve state computed from Algo. (1)                                    */
voroAdjNodeIDSetCur ← GetVoroAdjNodeIDs(startNodeID)
voroOppNodeIDSetCur ← GetVoroOppNodeIDs(startNodeID)
voroAdjNodeIDSetNext ← ∅
voroOppNodeIDSetNext ← ∅
visitedVoroNodeIDSet ← ∅
foundInAdjSet ← false
foundInOppSet ← false
/* Find valid path from from startNode to endNode using BFS.                 */
while !foundInAdjSet ∧ !foundInOppSet do
    for nodeID ∈ voroAdjNodeIDSetCur do
        for voroAdjNodeID ∈ GetVoroAdjNodeIDs(nodeID) do
            if voroAdjNodeID == endNodeID then
                foundInAdjSet ← true
                isAdjacent ← true
            else
                if voroAdjNodeID ∉ visitedVoroNodeIDSet then
                    voroAdjNodeIDSetNext.Add(voroAdjNodeID)
                end
            end
        end
        for voroOppNodeID ∈ GetVoroOppNodeIDs(nodeID) do
            if voroOppNodeID == endNodeID then
                foundInOppSet ← true
                isAdjacent ← false
            else
                if voroOppNodeID ∉ visitedVoroNodeIDSet then
                    voroOppNodeIDSetNext.Add(voroOppNodeID)
                end
            end
        end
    end
    if !foundInAdjSet ∧ !foundInOppSet then
        for nodeID ∈ voroOppNodeIDSetCur do
            for voroAdjNodeID ∈ GetVoroAdjNodeIDs(nodeID) do
                if voroAdjNodeID == endNodeID then
                    foundInOppSet ← true
                    isAdjacent ← false
                else
                    if voroNodeID ∉ visitedVoroNodeIDSet then
                        voroOppNodeIDSetNext.Add(voroAdjNodeID)
                    end
                end
            end
        end
    end
    voroAdjNodeIDSetCur ← voroAdjNodeIDSetNext
    voroAdjNodeIDSetNext ← ∅
    voroOppNodeIDSetCur ← voroOppNodeIDSetNext
    voroOppNodeIDSetNext ← ∅
    return isAdjacent
end
```

where $\mathcal{V}$ is the set of input nodes used to create our Voronoi diagram (Sec. 3.3.2), and the set $\mathcal{N}$ holds all grid nodes which are the neighbours of the nodes $\mathcal{S} = \mathcal{D} \cap \mathcal{B}$ that correspond to the fully-damaged area on our object's surface (*cf.* Fig. 3.4 (c), cyan-colored grid nodes). Intuitively, Eq. 3.2 defines the set of nodes $\mathcal{E}$ whose Voronoi cells are next to the medial surface but these cells are located *outside* the fully-damaged region (*cf.* Fig. 3.4 (e)(f), blue nodes).

For each node $\mathbf{i} \in \mathcal{E}$ we determine the side of the medial surface on which this node is located using the nearest node $\in I$ whose orientation w.r.t the medial surface (i.e., 'opposite' and 'adjacent' cell information) is already known (from Sec. 3.3.3 and Algo. (1)).

We then keep those Voronoi faces that lay on the opposite side of the medial surface. For each grid node $\in \mathcal{E}$, we first access its Voronoi cell and the corresponding neighbouring cells. Given a pair of nodes $\in \mathcal{E} \cup I$, we cast the problem of finding their relative orientation to one of finding their nearest nodes $\in I$ (Algo. (2)), from which we can deduce orientation. An example is shown in Fig. 3.4(f), where the orientation information of 'e0' and 'e1' is determined by their nearest neighbours 'i0' and 'i2', respectively. Since the span of any rasterized region will be at least $3h_{\mathrm{v}}$ (Sec. 3.3.2), neighbouring grid nodes are guaranteed to be located on the same side as the medial surface.

3.4 Enforce material discontinuity by using two velocity fields

The method described in Sec. 2.3 are sufficient to model propagation but do not account for material discontinuity at grid-cell resolution which is problematic with little-to-no deformations. Artificial wielding of damaged particles will exacerbate elastic- and fracture energy, and contribute to the thickness of the crack - thereby affecting fragment distributions. Three approaches exist to resolve this issue: CRAMP (Nairn and Guo, 2005; Guo and Nairn, 2017) uses an embedded mesh of massless particles but requires procedures for explicit mesh propagation in arbitrary directions with no solution for handling self-intersections, and it does not handle branching. CPIC will also use a mesh but it offers a limited decoupling of velocities (Appendix A), and requires a pre-specified discontinuity surface that is also manifold.. Damage field gradient (DFG) partitioning (Homel and Herbold, 2017) foregoes the use of a mesh to

infer crack orientation at grid nodes by using gradients ∇c. This method however fails to classify nodes in regions with excessive damage (thick cracks) where the gradient is zero, e.g., Fig. 3.5. Inspired by these approaches, we propose a simple but novel scheme to enforce separation during crack propagation, taking advantage of our crack representation as a medial-surface approximation of material damage.

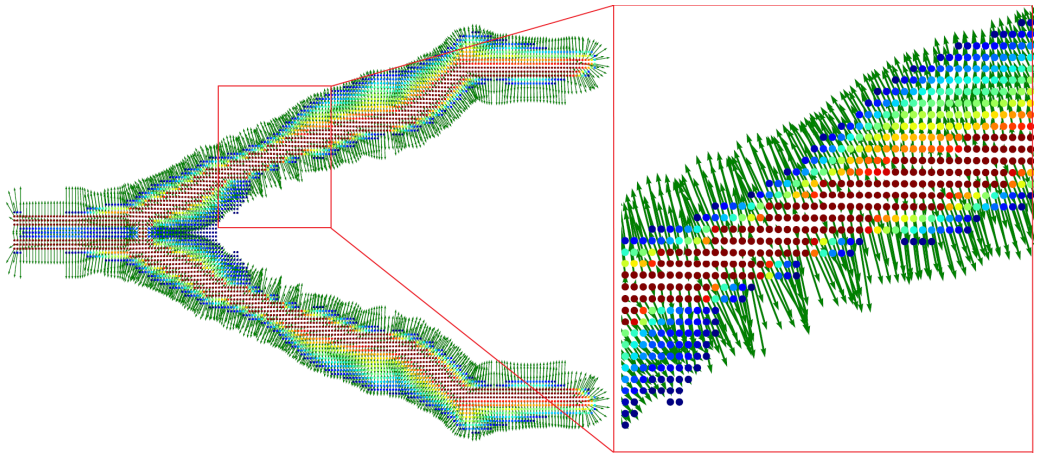


Figure 3.5: The damage gradient vanishes in excessive thick crack region using DFG method. Colors show the damage phase value and arrow lengths show the magnitude of the damage gradient.

3.4.1 Discontinuous velocity fields

Assume for a moment that a crack surface exists (which we describe in Sec. 3.3) since velocity discontinuity occurs with the existence of a crack. Thus let $\mathbf{v}_{\mathbf{i},\xi}$ be the velocity contribution to grid node $\mathbf{i}$ from particles that are compatible by sharing the same velocity field ξ. A particle p will belong to the field ξ at node $\mathbf{i}$ according to

$$\xi = \begin{cases} 0 & \text{if } \mathbf{n}_p \cdot \mathbf{n}_\mathbf{i} \geq 0 \\ 1 & \text{if } \mathbf{n}_p \cdot \mathbf{n}_\mathbf{i} < 0 \end{cases},$$

where $\mathbf{n}_p = \mathbf{x}_p - \mathbf{x}_c^p$ and $\mathbf{n}_\mathbf{i} = \mathbf{x}_\mathbf{i} - \mathbf{x}_c^\mathbf{i}$ are the respective crack normal estimates, with $\mathbf{x}_c^p$ and $\mathbf{x}_c^\mathbf{i}$ being the particle's and grid node's closest points on the crack respectively (Fig. 3.6). A particle whose distance to the crack exceeds its support radius (i.e., $||\mathbf{x}_p|| > 1.5h$) is assumed to be on the same side as its neighbourhood nodes, which are those in the support radius of this particle.

A disadvantage of this formulation is that the crack normal estimates rely on a closest-distance measure, hence a misclassification–and consequently, spurious G2P or P2G transfers–will occur for particles in the vicinity of a branch. Intuitively, this slightly reduces exactness of decoupling. This problem is, however, mitigated by the

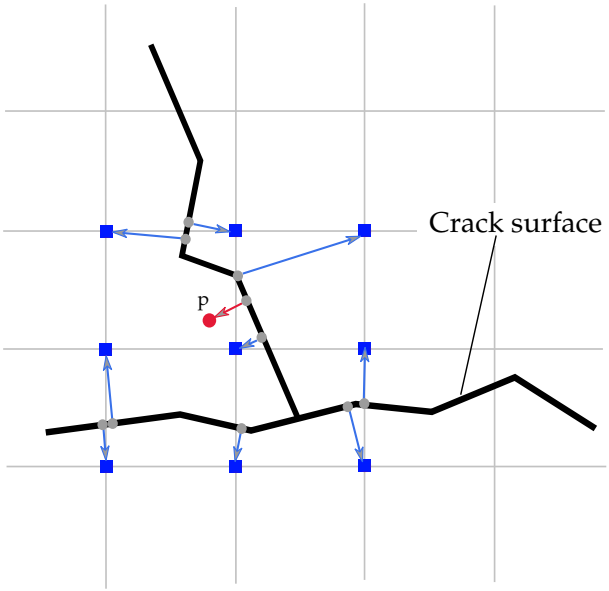


Figure 3.6: Estimating crack orientation at a particle p and the grid nodes in its influence radius.

fact that such spurious transfers–as low as they will be–do not persist beyond the kernel influence radius from the crack branch.

With this description, we make the following changes to our MLS-MPM discretization:

$$m^n_{\mathbf{i},\xi} = \sum_{p\in\xi} m_p w^n_{\mathbf{i}p} \tag{3.3}$$

$$(m\mathbf{v})^n_{\mathbf{i},\xi} = \sum_{p\in\xi} m_p w^n_{\mathbf{i}p}(\mathbf{v}^n_p + \mathbf{C}^n_p(\mathbf{x}_\mathbf{i} - \mathbf{x}^n_p)) \tag{3.4}$$

$$\mathbf{f}^{\text{int},n}_{\mathbf{i},\xi}(c) = -\sum_{p\in\xi} V^0_P w^n_{\mathbf{i}p} M^{-1}_p J_p \boldsymbol{\sigma}(\mathbf{F}_p, c)(\mathbf{x}_\mathbf{i} - \mathbf{x}^n_p), \tag{3.5}$$

where each grid node–within kernel radius from the crack–now has two velocity fields receiving contributions from respective particles. The center-of-mass velocity is

$$\mathbf{v}^{n+1}_\mathbf{i} = \frac{m^n_{\mathbf{i},0}\widetilde{\mathbf{v}}^{n+1}_{\mathbf{i},0} + m^n_{\mathbf{i},1}\widetilde{\mathbf{v}}^{n+1}_{\mathbf{i},1}}{m^n_{\mathbf{i},0} + m^n_{\mathbf{i},1}},$$

where $\widetilde{\mathbf{v}}^{n+1}_{\mathbf{i},\xi}$ is the node velocity that is computed from elastic forces using particle contributions corresponding to a field $\xi = 0, 1$. This modification prevents a spurious transfer of information between particles and nodes on opposing sides of the crack due to the non-conforming nature of MPM kernel functions.

Enforcing separation: Under compressive loading, material will interpenetrate without adequate treatment of contact on the crack surface. Thus, the nodal velocity field

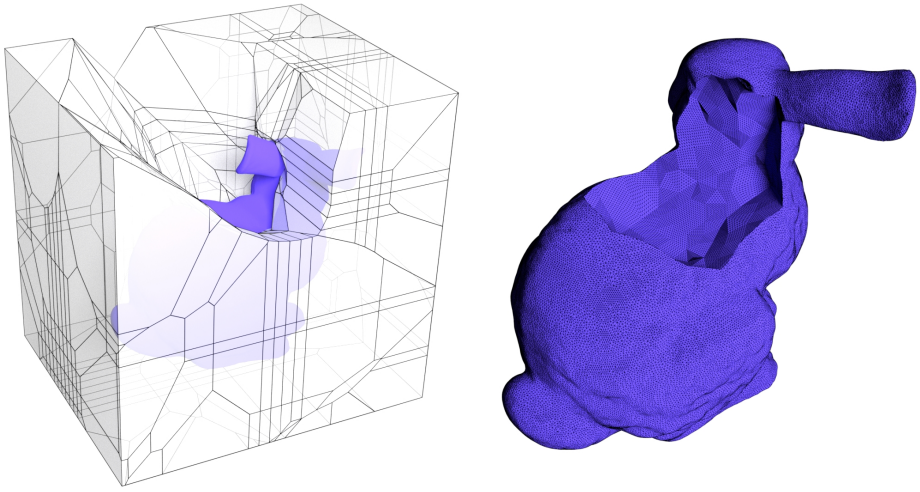
is adjusted to give

$$\mathbf{v}_{\mathbf{i},\xi}^{\mathrm{n}+1} = \widetilde{\mathbf{v}}_{\mathbf{i},\xi}^{\mathrm{n}+1} + \frac{\mathbf{f}_{\mathbf{i},\xi}^{\mathrm{ct}}}{m_{\mathbf{i},\xi}^{\mathrm{n}}}\Delta t, \tag{3.6}$$

where $\mathbf{f}_{\mathbf{i},\xi}^{\mathrm{ct}} = m_{\mathbf{i},\xi}^{n}/\Delta t(\mathbf{v}_{\mathbf{i}}^{n+1} - \widetilde{\mathbf{v}}_{\mathbf{i},\xi}^{n+1})\mathbf{n}_{\mathbf{i},\xi}$ is our contact force. We let this force to be non-zero when Eq. 3.7 is satisfied:

$$\left(\mathbf{v}_{\mathbf{i}}^{\mathrm{n}+1} - \widetilde{\mathbf{v}}_{\mathbf{i},\xi}^{\mathrm{n}+1}\right)\mathbf{n}_{\mathbf{i},\xi} < 0, \tag{3.7}$$

where $\mathbf{n}_{\mathbf{i},\xi}$ is the normal corresponding to the field ξ at node $\mathbf{i}$, with $\mathbf{n}_{\mathbf{i},\xi=0} = \mathbf{n}_{\mathbf{i}}$ and $\mathbf{n}_{\mathbf{i},\xi=1} = -\mathbf{n}_{\mathbf{i}}$. We neglect tangential contact force since this mode of interaction has a negligible effect on mitigating the aforementioned exacerbated-energy artefacts which motivate our solution.



(a) (b)

Figure 3.7: Boolean intersection between the domain (bunny) and the boundary surface of a *coarse* Voronoi tessellation region: (a) Input configuration, (b) resulting fragment, without fracture detail.

3.5 Extract fragments

Crack extraction (Sec. 3.3) will identify lists of Voronoi faces corresponding to the crack(s), where these faces collectively define open surfaces (sheets) that must then be intersected with the domain boundary to produce new fragments. This section describes how we compute this fragment geometry using our crack surface, Voronoi diagram and the domain boundary, by computing fragments as intersections of a given domain and Voronoi tessellation regions.

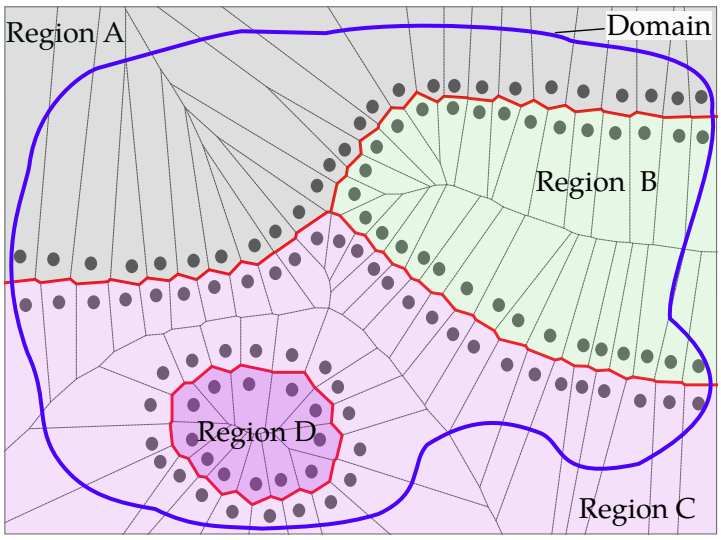


Figure 3.8: An example illustration of Voronoi tessellation regions (shaded) that are separated by crack faces (red). In this example, our Voronoi diagram is clipped against a bounding box but any dilated shape which encapsulates the domain could be used. We extract fragments as a boolean intersection between a region and the domain. Note that we always use grid nodes as inputs to our tessellation in practice.

3.5.1 Identifying Voronoi-tessellation regions

We now describe how we define and extract Voronoi tessellation regions (heretofore referred to simply as "regions") that are bounded by the crack surfaces. Each such region is a collection of Voronoi cells–from the Voronoi diagram in Sec. 3.3.2–that surround a fragment. An illustrative overview is shown in Fig. 3.7 and Fig. 3.8.

We use breadth-first search (BFS) to find these regions, starting from any unvisited Voronoi cell (*initial step*) to build a list of cells which will form a region (*cf.* Fig. 3.8). A queue data structure is also used when building this list. Any unvisited cell is first inserted, making it the current cell at the front of the queue. The neighboring cells that share a face with the current cell are enqueued if 1) it is not in the queue, 2) it has not been visited, 3) and this face is not a crack face. The current cell is then popped from the queue, marked 'visited', inserted to the current list after checking all of its faces. We continue from the next element in the queue until it is empty, after-which a single region is formed by the cells in the list. We then restart from the initial step if there still exists any unvisited cell to find the next region.

(a) Input model. (b) Extracted non-manifold crack surface. (c) Extracted fragment surface.

(d) Volume-preserving cut using MCUT. (e) openVDB full cut. (f) openVDB partial cut.

Figure 3.9: Different fragment cutting methods: (a) Input model. (b) Extracted non-manifold crack surface. (c) Extracted fragment surface using the algorithm in Sec. 3.5. (d) Cut the object by the crack surface in (c) using mesh based strategies (MCUT: (Chitalu et al., 2020)). The cut is volume-preserving. (e) Cut the object by the crack surface in (c) using level-set (openVDB). A small portion of volume is lost in the cracked region. (f) Cut the object by the crack surface in (b) using level-set (openVDB). The partial cut is desirable in some scenarios.

3.5.2 Cut fragments in the material space

The boundary faces of a region (*cf.* Fig. 3.7, a) will thus form a solid mesh with-which we can compute our fragments as boolean intersections with the domain boundary. Our solution to extracting fragments is thus amenable to either level sets (e.g., using tools like OpenVDB (Museth et al., 2013)) or mesh based strategies (like Chitalu *et al.* (Chitalu et al., 2020) and Zhou *et al.* (Zhou et al., 2016)) for cutting and without issues of volume loss since our intersection inputs are now solids with a manifold surface.

It remains possible to cut the domain directly using our non-manifold crack surface but as thickened volumes with level sets as in (Hahn and Wojtan, 2015), which is useful for showing hairline cracks as thin gaps in the material for added visual effects. In Fig. 3.9, we show the difference between different cutting methods.

3.6 Rigid-body simulation

In this section, we describe how to couple the MPM simulation with a rigid body solver. We first show that we decimate high-resolution mesh to a lower resolution for collision detection in Sec. 3.6.1. Then we introduce the Hertzian model for generating contact impact upon contact in Sec. 3.6.2. Finally, we show that we control the fragments generation by setting a few parameters in Sec. 3.6.3.

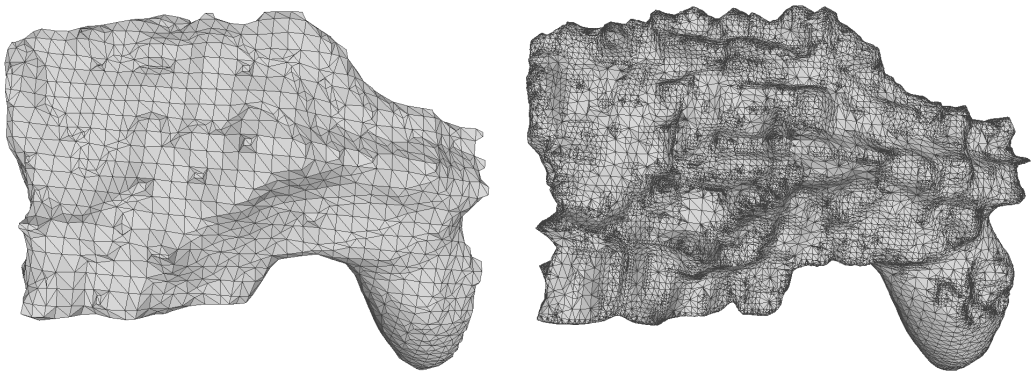
3.6.1 Decimate mesh

Meshes of different resolutions are used for simulation and collision detection. Before simulation, we convert the object's surface mesh into a level-set using openVDB. The zero level-set implicitly defines the object's surface, whose resolution is dependant on the resolution of openVDB's voxel size. A smaller voxel size level-set keeps fine details of the object but at the cost of high computational cost. On the contrary, a larger voxel size brings faster performance but may lose details such as sharp edges. To balance between performance and accuracy, we use the higher resolution mesh for rendering, and the lower resolution mesh for collision detection. Since the collision is done with a BVH and most of the fragments have convex shapes, a low resolution mesh is sufficient for this purpose. Furthermore, we also sample material points and do crack simulation using the low resolution mesh.

The higher resolution mesh is decimated with Libigl library (Jacobson et al., 2018). A user specified threshold, N_{df} , controls the number of mesh's faces after decimation. In practice, we set $N_{df} = 10K$ which is sufficient for a robust and accurate collision detection although N_{df} varies with the complexity of the geometry. Fig. 3.10 shows the rendering mesh and simulation mesh of a typical fragment.

3.6.2 Contact impact

The contact impact computed by the rigid-body solver sets the boundary conditions of our MPM fracture simulation. The impact specifies the contact force and duration time. The contact force is computed from integral of impulse reported by the rigid-body system. To do this, we first build a traction field from collision impulse. Then we map the collision points from the rigid-body system to the closest element in the collision mesh. According to Hertzian model (Glondu et al., 2013), each collision point contributes $I\bar{n}/(t_c A_e)$ traction to the element e to corresponding input force vector. $\bar{n}$



(a) (b)

Figure 3.10: For optimal performance, we decimate the high-resolution mesh to a low-resolution for simulation purpose. The original high-resolution mesh is kept for rendering. (a) Simulation mesh: 2880 vertices and 5756 faces; (b) Rendering mesh: 40507 vertices and 81010 faces.

and A_e are the outward normal and element area of the element e, respectively. Finally, I and t_c are contact impulse and duration time, respectively.

The Hertzian model estimates the contact duration time using two elastic spheres or one elastic sphere with a plane. A sphere s has the elastic modulus E_s, radius R_s and mass m_s. The equivalent elastic modulus E, radius R and mass m are computed as:

$$E = \left[\frac{(1-V_1^2)}{E_1} + \frac{(1-V_2^2)}{E_2} \right]^{-1} \tag{3.8}$$

$$R = (R_1^{-1} + R_2^{-1})^{-1} \tag{3.9}$$

$$m = (m_1^{-1} + m_2^{-1})^{-1} \tag{3.10}$$

The contact duration time t_c is computed as:

$$t_c = 2.87 \left(\frac{m^2}{E^2 R v} \right)^{\frac{1}{5}} \tag{3.11}$$

where v is the relative velocity of two spheres.

The elasticity parameter and density of the objects are specified by the material parameters from user input. The contact velocity and impulse are calculated from the rigid-body system. Since the fragment is not an ideal sphere, we calculate the equivalent radius as the distance between the contact point and the fragment's center of mass. The volume is computed using the high-resolution mesh which is then used to calculate the mass.

The contact is finally represented as a force vector $\mathbf{f}_g^b$ applied at a point $\mathbf{x}_g$. We choose this point as a ghost particle which then transfers the contact force to the corresponding background grid node i:

$$\mathbf{f}_\mathbf{i}^{bd,n} = \sum_g \mathbf{f}_g^b w_{\mathbf{i}g}^n \tag{3.12}$$

where $w_{\mathbf{i}g}^n$ is the weight between grid node i and ghost particle g, and $\mathbf{f}_\mathbf{i}^{bd,n}$ is the external contact force as in Eq. 2.18.

3.6.3 Rigid-body solver

Once a fracture simulation finishes, new fragments are generated accordingly. We replace the original object with its fragments. Before the simulation, we calculate the *original volume* of each breakable object in the system. During fracture simulation, we pass this information to all it descendent fragments. By comparing ratio of the fragment's volume to the *original volume*, Vr , we classify four kinds of objects similar to Hahn and Wojtan (2015). In practice, we set three thresholds to control the generation of fragments, VR_t, VR_s, and VR_l.

1. Tiny fragments ($Vr < VR_t$): These fragments are so small that we don't allow them to break further for efficiency reason. In order to speed-up collision detection, we use their convex hull to represent them.

2. Small fragments ($VR_t \leq Vr < VR_s$): These fragments cannot break either. However, they are represented by meshes to enable a more accurate collision detection.

3. Large fragments ($VR_s \leq Vr < VR_l$): These fragments are breakable and can fracture again upon a sufficiently large impact. Their shapes are also represented by meshes.

4. Extremely large fragments ($VR_l \leq Vr$): These fragments are so large, which is nearly close to the original object. For efficiency purpose, we reuse the sampled MPM particles in further fracture simulation. They are breakable in further fracture simulation. Their own low-resolution meshes are still used for collision detection.

Note that all four kinds objects are represented by high-resolution meshes in the final rendering. In most of our experiments, we set $VR_t = 0.05$, $VR_s = 0.2$, and $VR_l = 0.98$, respectively.

3.7 Add realistic brittle fracture details

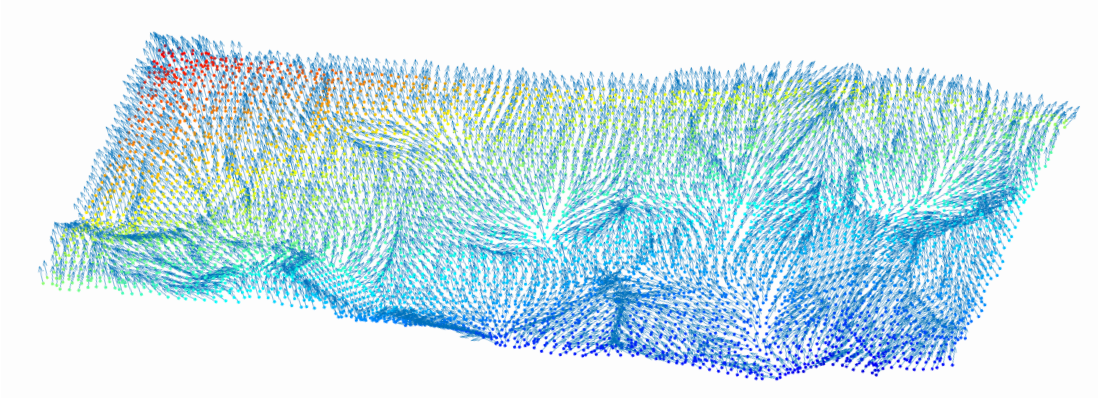


Figure 3.11: Gradient vector field visualisation from a patch of the crack surface shown in Fig. 3.3. The node-color blue denotes the 'earliest' regions of damage with red being the 'latest' among the considered set of damaged particles. Intermediate node colors reveal the gradual progression of damage over time.

Crack surfaces computed by the medial axes as described in Sec. 3.3 can sometimes be excessively flat; we now describe an approach to enrich our extracted crack surface with fracture detail as a tool for artistic control. Real-world fracture surfaces yield complex and unique patterns resulting from the microscopic variations in crack front stress. In brittle material, fracture patterns (termed 'river-lines' or 'chevron marks') are characterised by time-dependent cleavages due to irregularities in the material, giving rise to plateaus connected by sheer ledges and observable directions of crack-front motion (Dieter, 1986). By tracking the crack front, one can influence the arising patterns (as in Hahn and Wojtan (Hahn and Wojtan, 2015)) but this is a non-trivial task for particle-based methods like MPM. Moreover, the smeared nature of continuum damage mechanics (CDM) makes tracking the exact crack front's trajectory even harder. We solve this problem by modelling the time evolution of damage at material particles which produces vector fields that inform localised perturbations of crack mesh vertices.

Damage time vector fields: To track the rough trajectory of the crack front, we

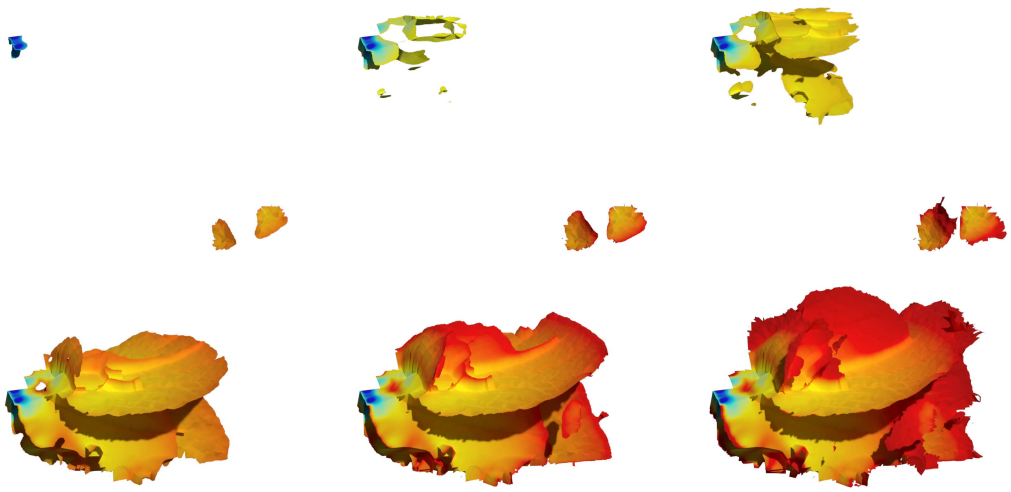


Figure 3.12: Crack propagation visualized by damage time. Blue color represents smaller damage time (earlier damage). Red color represents larger damage time (later damage).

introduce a damage time variable τ for each material particle $\mathbf{x}_p$, which records the time when this particle is fully damaged (i.e., when $c = 1$). Our idea is to treat the damage time gradient as an estimate of the crack propagation direction, which we can use to model chevron marks as a simplified scheme. Moreover, we depart from physical models which describe crack fronts as disrupted in their straight paths by irregularities in the material, causing bends and bumps. An example depiction of damage time along a crack surface is shown in Fig. 3.11 and Fig. 3.12.

In practice, a grid structure is used to approximate the gradient, where the damage time at a grid node $\mathbf{i}$ is approximated by

$$\tau_{\mathbf{i}} = \frac{\sum_p w_{\mathbf{i}p}\tau_p}{\sum_p w_{\mathbf{i}p}}. \tag{3.13}$$

Assuming w is a quadratic kernel function (*cf.* Eq. 2.10), we compute the gradient of an arbitrary point α on the grid using

$$\nabla\tau_\alpha = \sum_{\mathbf{i}} w_{\mathbf{i}\alpha} M^{-1}[\mathbf{x}_{\mathbf{i}} - \mathbf{x}_\alpha]^\intercal \tau_{\mathbf{i}}, \tag{3.14}$$

where $M = h_c^2/4$ is a constant using h_c as our grid spacing which is separate from previously defined grid-spacings in Sec. 2.2 and Sec. 3.3, $\mathbf{x}_i$ and $\mathbf{x}_\alpha$ are coordinates of

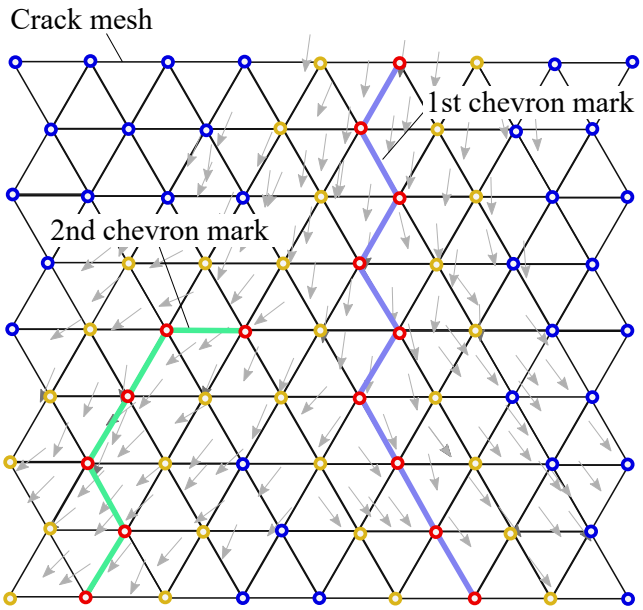


Figure 3.13: A visual depiction of the steps in Algo. (4). We trace chevron marks according to a damage-time vector field, visiting vertices in the neighbourhood of each path (colored green and magenta). Red: vertices of a chevron mark. Blue: vertices not yet visited. Yellow: vertices in the influence radius of at-least one chevron mark.

a node **i** and our arbitrary point a, respectively. The derivation of Eq. 3.14 is provided in Appendix B.

Modelling fracture patterns: We now describe how we trace gradient flow lines of the damage time vector fields to form the chevron marks. We first refine along the crack surface mesh from triangulation, triangulating polygons until the maximum edge length is less than a threshold d.

Next, we trace along the time gradient on the crack surface to compute the flow paths. An outline of the steps is provided in Algo. (4). We start a flow path from any unvisited vertex that has the smallest damage time τ. To extend the path, we search forward by a fixed step size d in the gradient direction $\nabla\tau$ from current vertex to find the nearest unvisited vertex on the crack mesh. Nearby vertices that are perpendicular to the gradient direction and within a distance of another threshold s will also be marked as visited, which allows us to control width and smoothness. For a single flow path, we iterate until the maximum length l_c is reached or a visited vertex is encountered. This procedure is repeated until all the vertices are visited.

Crack mesh vertex perturbation: Having traced our paths representing gradient flow on the refined crack mesh, we add the fracture detail to this mesh to form chevron

Parameter	Description
d	Crack mesh refinement maximum edge length Chevron line extension step size $\uparrow \equiv$ *less detailed chevron marks*
l_c	Chevron line maximum length $\uparrow \equiv$ *longer*
s	Chevron line influence radius $\uparrow \equiv$ *Increase width/flatness*
h_c	Detail grid spacing $\uparrow \equiv$ *Smoothing*
G	Global perturbation magnitude $\uparrow \equiv$ *Height*
m	Ratio of enhanced chevron lines $\uparrow \equiv$ *Higher variation*

Table 3.2: Fracture detail control parameters. "↑" means increasing.

marks. We perturb each crack mesh vertex in the normal direction $\mathbf{n}_v$, weighting by proximity to the closest flow path(s). By default each vertex defining a flow path is initially assigned a constant perturbation value $\beta = 1$ of one, while the remaining vertices are assigned an initial value of zero ($\beta = 0$). Observing that chevron marks tend to have a higher altitude further away from their source, we update the perturbation value by $\beta \leftarrow \beta + (1 - r)$ along the flow path, where r is the ratio between the distance along the path from a vertex to the source on the flow path and the length of this flow path. In practice, we only select the top m $(0 \leq m \leq 1)$ longest flow paths for this procedure, where m is a user parameter. We then use our background grid to interpolate to neighbouring vertices. Perturbation values are transferred to grid nodes using $\beta_{\mathbf{i}} = \sum_v w_{\mathbf{i}v}\beta_v$, where $\beta_{\mathbf{i}}$ is the perturbation value of node $\mathbf{i}$, $w_{\mathbf{i}v}$ is the weight, and β_v is the perturbation of crack mesh vertex v. Conversely, we distribute perturbation values to mesh vertices by $\beta'_v = \sum_i w_{\mathbf{i}v}\beta_{\mathbf{i}}$.

In summary, a vertex $\mathbf{x}_v$ of our refined crack mesh is perturbed according to $\mathbf{x}'_v = \mathbf{x}_v + \mathbf{n}_v\beta'_v G$, where $\mathbf{x}'_v$ is the displaced coordinate, and G is user-parameter that we introduce to control the global height. Moreover, having more flow paths in effect will result in a larger perturbation magnitude since we do not scale the height value β'_v by the weight-sum $\sum_{\mathbf{i}} w_{\mathbf{i}v}$, which is desired. A list of the parameters controlling fracture patterns is provided in Tab. 3.2.

Algorithm 4: Chevron lines on the crack surface

Input: crackMesh, τ, l_c, d, s

Output: chevronMarks

```
1  visited ← ∅
2  while not all vertices are visited do
3      chevronMark ← ∅
4      /* Current vertex (one with smallest damage time)                    */
5      x ← GetEarliest(crackMesh.GetVertices() \ visited)
6      chevronMark.Insert(x)
7      visited.Insert(x)
8      for i ← 0 to l_c do
9          x_shift ← x + ∇τ(x) * d
10         /* Position and normal of nearest                                */
11         x_near, n_near ← GetNearestVertexData(x_shift, crackMesh)
12         if x_near ∉ visited then
13             chevronMark.Insert(x_near)
14             /* Direction perpendicular to chevron line at x              */
15             gradientPerp ← ∇τ(x) × n_near
16             /* Search using +step and -step                              */
17             step ← gradientPerp * s
18             nearestSet ← GetNearestVertices(crackMesh, x_near, step)
19             visited.Extend(nearestSet)
20             x ← x_near
21         else
22             Break
23         end
24     end
25     chevronMarks.Insert(chevronMark)
26 end
```

3.8 Results

We present our results in this section, which are obtained using an Intel Core i9-7920X processor (2.90 GHz, 24 cores) with 62.6GB RAM. We first show common test cases that are typically employed in mechanics literature to validate our method (Sec. 3.8.1). We then present our animation results combining rigid body dynamics with CDM fracture, which are also shown in our accompanying videos (Sec. 3.8.2). Rigid-body dynamics are simulated using Bullet (Coumans, 2015), where we have used the multi-threaded implementation with 'GImpact' collision shapes. Our MPM simulation (*cf.* Sec. 2.2 and Sec. 3.4) is also multi-threaded, which we implement with OpenMP (OpenMP Architecture Review Board, 2015). Throughout our experiments and demos, crack surfaces are extracted every ten time-steps for particle decoupling (Sec. 3.4). We apply cracks on our objects (Sec. 3.5) using OpenVDB (Museth et al., 2013) which is a fast and robust alternative to mesh based methods [2]. Mesh processing, like decimation, is also handled with OpenVDB using mesh-to-level-set conversions on coarsened grids, and Voronoi diagrams (*cf.* Sec. 3.3) are created using Voro++ (Rycroft, 2009).

Coupling rigid-body dynamics to an elastodynamic simulation requires a transition from the rigid to the deformable model. Elastic models resolve collisions using forces, while rigid body models operate with impulses which are independent of the (rigid body) time step size. We convert impulses to forces, scaling by a collision duration that is estimated with a Hertzian contact model as in related quasi-static methods (Glondu et al., 2013; Koschier et al., 2014; Hahn and Wojtan, 2016). The resulting force is then applied on the MPM grid via ghost particles as described in Sec. 3.6.2.

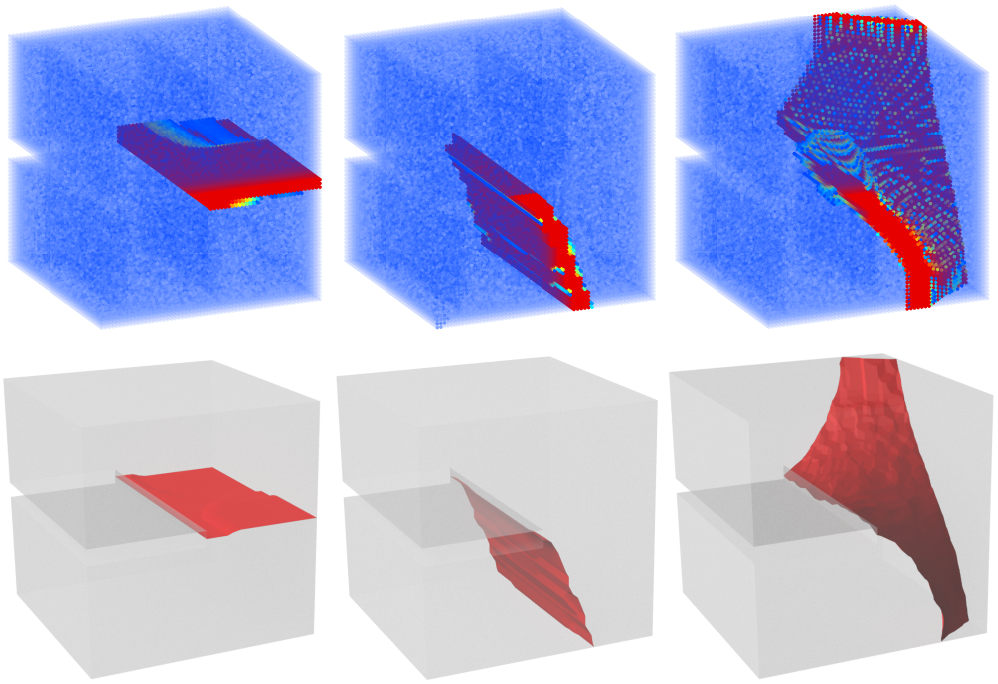
The source code for crack extraction and cutting can be downloaded from the following link. [3]

3.8.1 Method validation

Mode tests: We first show a cube with a planar edge crack under Mode I, II and III loading as shown in Fig. 3.14, respectively. Our simulation can produce qualitatively

[2] We also tested our method using the mesh based algorithm of Chitalu *et al.* (Chitalu et al., 2020) whose code is available online. Their method is also fast and generally worked well but tended to produce near-degenerate polygons on fragment meshes with consecutive cuts, which required additional remeshing.

[3] https://github.com/Linxu-Fan/crack-extract



(a) Mode I (b) Mode II (c) Mode III

Figure 3.14: Mode tests Irwin (1957): Top row shows simulated damage (Sec. 2.3) and bottom row is the extracted crack surface (Sec. 3.3). Mode I: The opening mode, where the crack opens perpendicular to the crack plane which is caused by tensile loading. Mode II: In-plane sliding mode, where the crack faces are displaced on their plane, normal to the crack front. Mode III: Out-of-plane tearing mode, where the crack faces are displaced on their plane, parallel to the crack front.

similar results to what is expected for all three modes of fracture loading, although the crack surface reveals subtle artefacts, e.g., for Mode I, due to our tessellation algorithm and resolution.

Resolution dependency: Smeared cracks can have wide or perhaps jagged surfaces, depending on the material and boundary conditions, but this phenomena may also be produced erroneously in a simulation. In Fig. 3.15 we evaluate the effect of particle resolution and characteristic length to validate our baseline propagation model (Sec. 2.3). Our method is resolution-dependent, where the detail of crack propagation (branching patterns) is affected by the number of particles and the characteristic length. However, the visual effect is mitigated by our fragment extraction routine (Sec. 3.5). The general smeared crack path direction is similar when varying the resolution but an increase will induce additional branching to create off-shoots that emanate from the primary crack path(s).

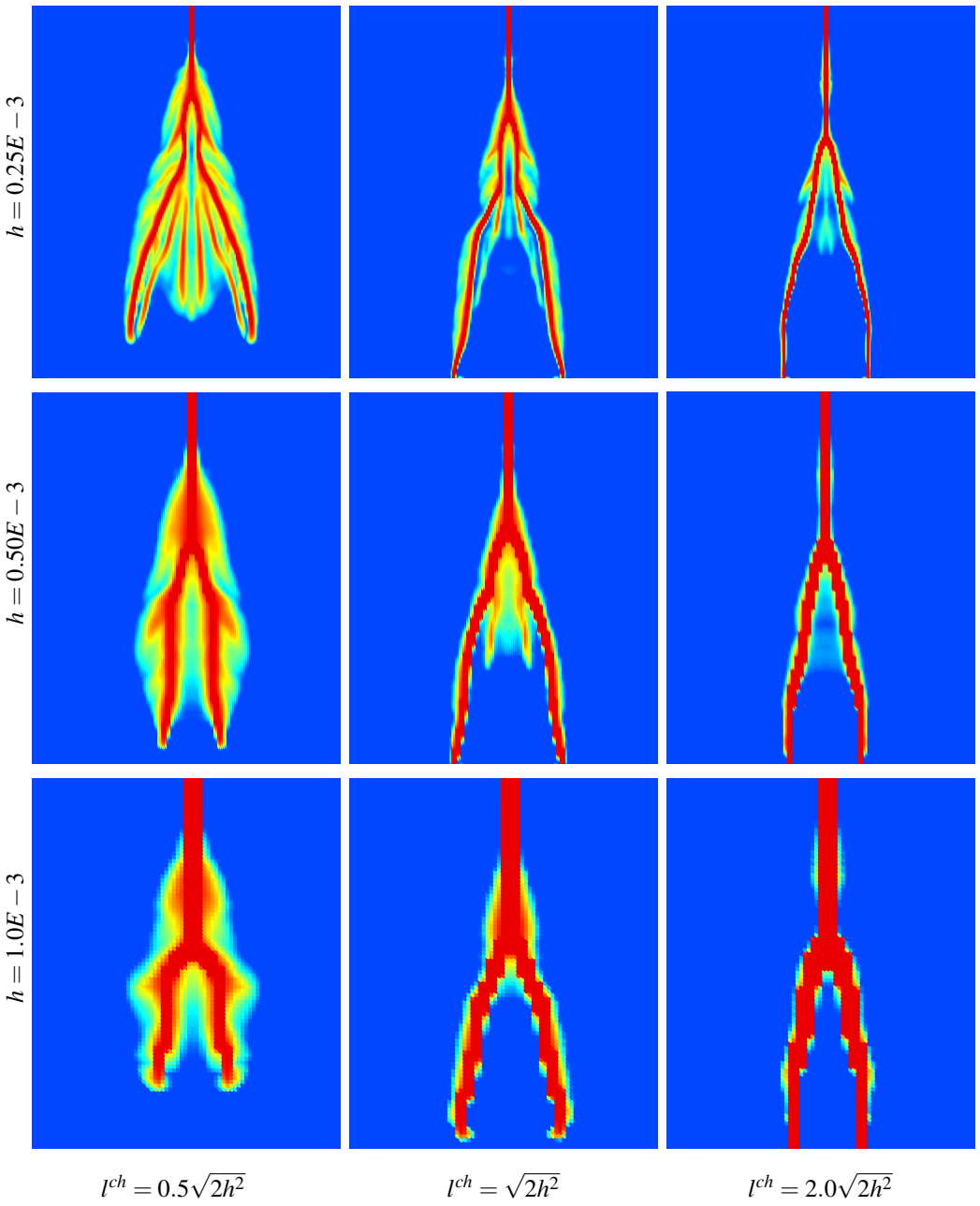
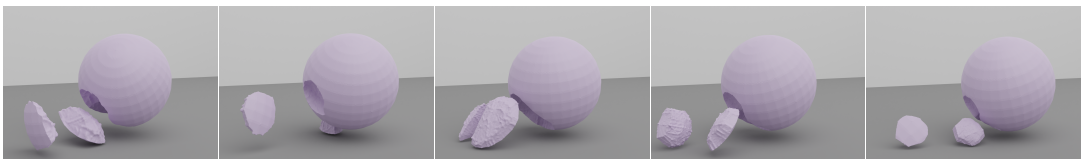


Figure 3.15: Pre-notched rectangular plate test, where we vary the characteristic length (left to right) and the number of particles (bottom to top): We show a simulation of branching (from the steps described in Sec. 2.3) using three configurations of grid spacing. Notch-width is three times the grid spacing, which is initialised as a smeared crack running half the plate-length, and later propagated to the end of the plate (top-to-bottom). Each row and column use the same grid resolution/spacing and characteristic length respectively. Our average particle spacing is approximately half the grid spacing (four particles per grid cell).

These off-shoots will not be visible in practice however as they tend to form partial cracks due to insufficient tip-stress such that–when combined with our extraction al-



(a) 16k (a) 32k (a) 64k (a) 128k (a) 256k

Figure 3.16: Breakage result at different resolutions (number of particles), where all configurations result in the sphere breaking into three parts with similar shape (and volume). Using a relatively low number of particles is possible due to our crack extraction method which is robust to variations in resolution.

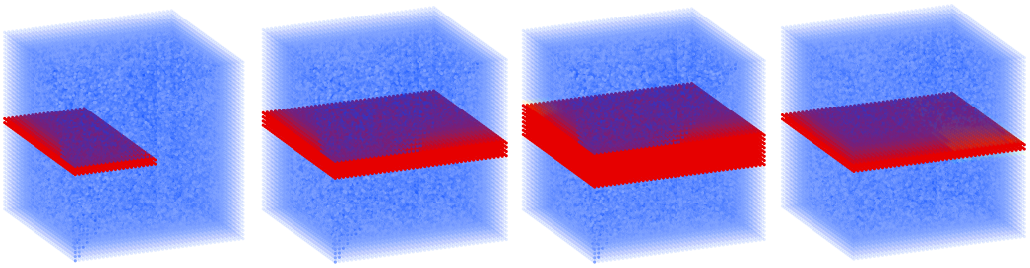
gorithm (Sec. 3.5)–they are discarded to leave only the fully propagated primary crack paths that occur irrespective of the resolution. Thus, using a relatively low number of particles in our simulations (*cf.* Sec. 3.8.2) is possible due to our crack extraction method which is robust to variations in resolution and– hence–crack thickness. A further example is shown in Fig. 3.16, demonstrating the consistency of our visual result w.r.t the change in resolution.

Observed differences in smeared crack-path thickness and the number of branches in Fig. 3.15 occur for two reasons: First, the thickness of the pre-specified crack (in terms of the number of layers of particles) is the same in all variations of parameters - thus it appears thinner with higher resolution. Second, the total fracture energy is scaled in direct proportion to the total volume of damaged particles (see Eq. C.2). Less energy is released along the–now thinner–primary crack path(s), whereby the remaining energy is instead released through further branching to balance external loading via the conservation of energy principle.

The characteristic length l^{ch} will also influence crack velocity to require a resolution-dependent tuning (Cervera and Chiumenti, 2006). We found that using the grid-cell diagonal length $l^{ch} = \sqrt{3h^2}$ works (in 3D) well with low resolution (number of particles) but will, of-course, leads to inconsistencies at higher resolution, e.g., changes in damage distributions.

Kernel dependency: We successfully mitigate the effect of kernel-function choice on smeared-crack path thickness by constraining radial-influence across the crack. Fig. 3.17 shows the influence of kernel choice on smeared-crack path thickness (number of damaged particles). Using the single-velocity field approach (Sec. 2.2.2), the cubic and quadratic kernels lead to eight and four layers of damaged particles respectively, due to the size of kernel radius and having a single continuous velocity field.

Conversely, note the similarity in crack thickness between Fig. 3.17 (a) and (d). Our decoupling strategy (Sec. 3.4) limits damage-spread to just two layers, which correspond to the opposing brittle-extract-crack sides.

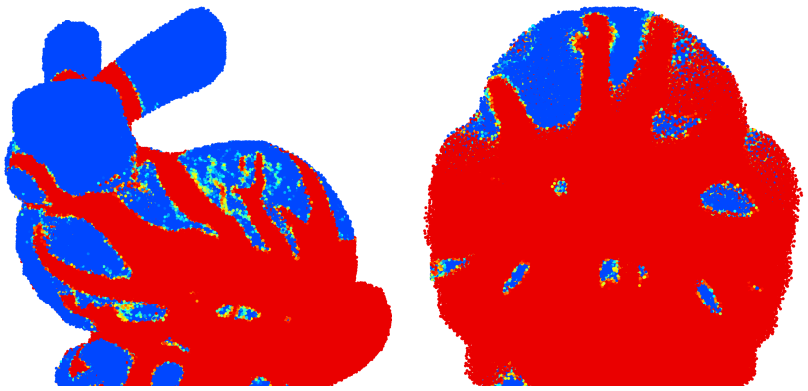


(a) Initial crack (b) Quadratic kernel (c) Cubic kernel (d) Ours (Sec. 3.4)

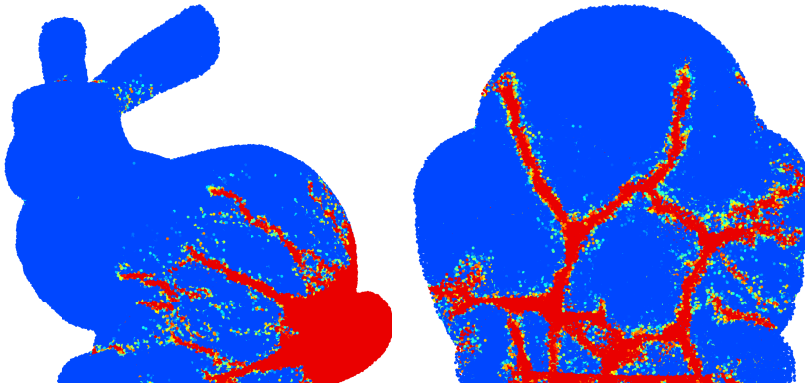
Figure 3.17: Interpolation effect on smeared crack path thickness. A crack is propagated under Mode-I loading. The cubic- and quadratic kernels have a support radius of $2h$ and $1.5h$ respectively. We have used 81356 particles with a time-step size of 5.0E-6 in this experiment.

Discontinuous velocity fields: The advantage of our multi velocity field scheme is also revealed in examples with prolonged contact/Neumann boundary conditions. Fig. 3.18 shows an example with the Stanford Bunny, where a force is applied to the tail and in the direction of the torso. After 5600 timesteps (1.0E-7 seconds per step with a million particles) our approach, using multiple velocity fields, produces smeared crack paths resembling physical propagation. Conversely, using a single velocity field produces exercabated damage spanning the torso. In this example, the single velocity field approach produces over $3\times$ the number of damaged particles ($> 670K$) compared with the multi-velocity field approach ($219K$). Excessive damage in the tail region is due to compressive loading which results from the deformation. Moreover our phase field formulation only 'weakens' the tensile component of stress [4], thus compressive forces still work to effectively squeeze the material, causing tensile loading in perpendicular directions.

[4] In general, physical materials fail mainly due to tensile straining (Cervera and Chiumenti, 2006).



Single velocity field (Sec. 2.2.2)



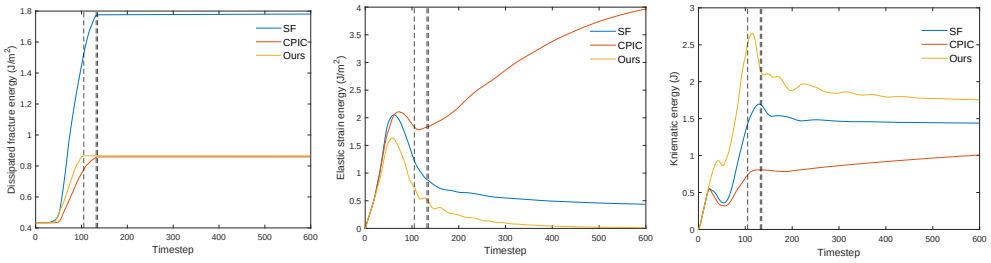
Multiple velocity fields (Sec. 3.4)

Figure 3.18: A visual comparison (side view and cross-section at the torso) between simulating propagation using a single velocity field *vs.*multiple velocity fields introduced along the crack path.

Energy: We also evaluate the relationship between crack propagation and the change in three types of energy, using the same experimental setup as in Fig. 3.17 (a) (Mode-I). These energies are fracture-, elastic- and kinetic energy. Fig. 3.19 plots results where we analyse and compare our decoupling strategy against CPIC (Hu et al., 2018) and the single-velocity field local CDM (SF) (Sec. 2.2.2).

The dissipated fracture energy (see Fig. 3.19 (a), and Eq. C.2) of our new method and CPIC are similar but with our approach propagating the crack at a faster rate. Both of these methods produce two layers of damaged particles, while SF produces four layers (*cf.* Fig. 3.17) to more-than-double the fracture energy after material separation.

Fig. 3.19 (b) further demonstrates our method's ability to provide a strong decoupling of velocity fields once fragments are separated at grid-cell resolution. Elastic energy (Eq. C.1) tends toward zero, denoting equilibrium as expected. Conversely, SF



(a) Fracture Energy (b) Elastic Energy (c) Kinetic Energy

Figure 3.19: The change in energy w.r.t. crack propagation (using same setup as in Fig. 3.17 but only using the quadratic kernel). The dashed vertical lines represent the time-step when material is separated. Material separation occurs at different time-steps for each method. Single velocity field (SF) approach from Sec. 2.2.2 takes 135 time-steps (0.675ms); CPIC (Hu et al., 2018) take 132 time-steps (0.6ms); and ours take 105 time-steps (0.525ms).

reveals a similar profile but has a consistently higher energy that also tends toward zero but still differ by approx' 0.5 units after 600 steps due to weak decoupling across the crack. CPIC, on the other hand, fails to converge for similar reasons. Moreover, CPIC particles contribute a constant local velocity to nodes that are on the opposite side of the crack and within kernel radius. Furthermore, as the deformation is inherently linear (small), damaged particles and nodes within kernel radius and on opposing side of the crack will interact for a prolonged period to exacerbate energy (*cf.* Fig. 3.19 (c), CPIC). Refer to our supplementary document for details on the weak decoupling associated with CPIC. In general, the decreasing elastic and fracture energies of our method can be seen as a gradual conversion to kinetic energy $\varepsilon_k = \int_\Omega \frac{1}{2}\rho v^2 dx$, which is shown in Fig. 3.19 (c).

Adding fracture details: We show in Fig. 3.20 that by controlling a handful of parameters (*cf.* Tab. 3.2), we can produce a variety of realistic fracture patterns. A few examples are shown where chevron marks vary in length, width and height to produce interesting contours on fragments. We are able to model numerous surface patterns, from concrete to rock, clay and more.

We also show results in Fig. 3.20 (e) where we use Chen *et al.* (Chen et al., 2014) to add details on the surface. Here we use a volume texture where dot patterns running from top to bottom to produce chevron marks. It can be observed chevron marks appear for areas where the direction of the texture and the crack surface are tangential, but not otherwise. A detailed 3D texture that matches the geometry of the crack surface is needed for producing effects equivalent to our method. In contrast, our methods yield

physically correct fracture patterns without the modification of toughness parameters like (Hahn and Wojtan, 2016) or artificial input like a texture map(Chitalu et al., 2020).

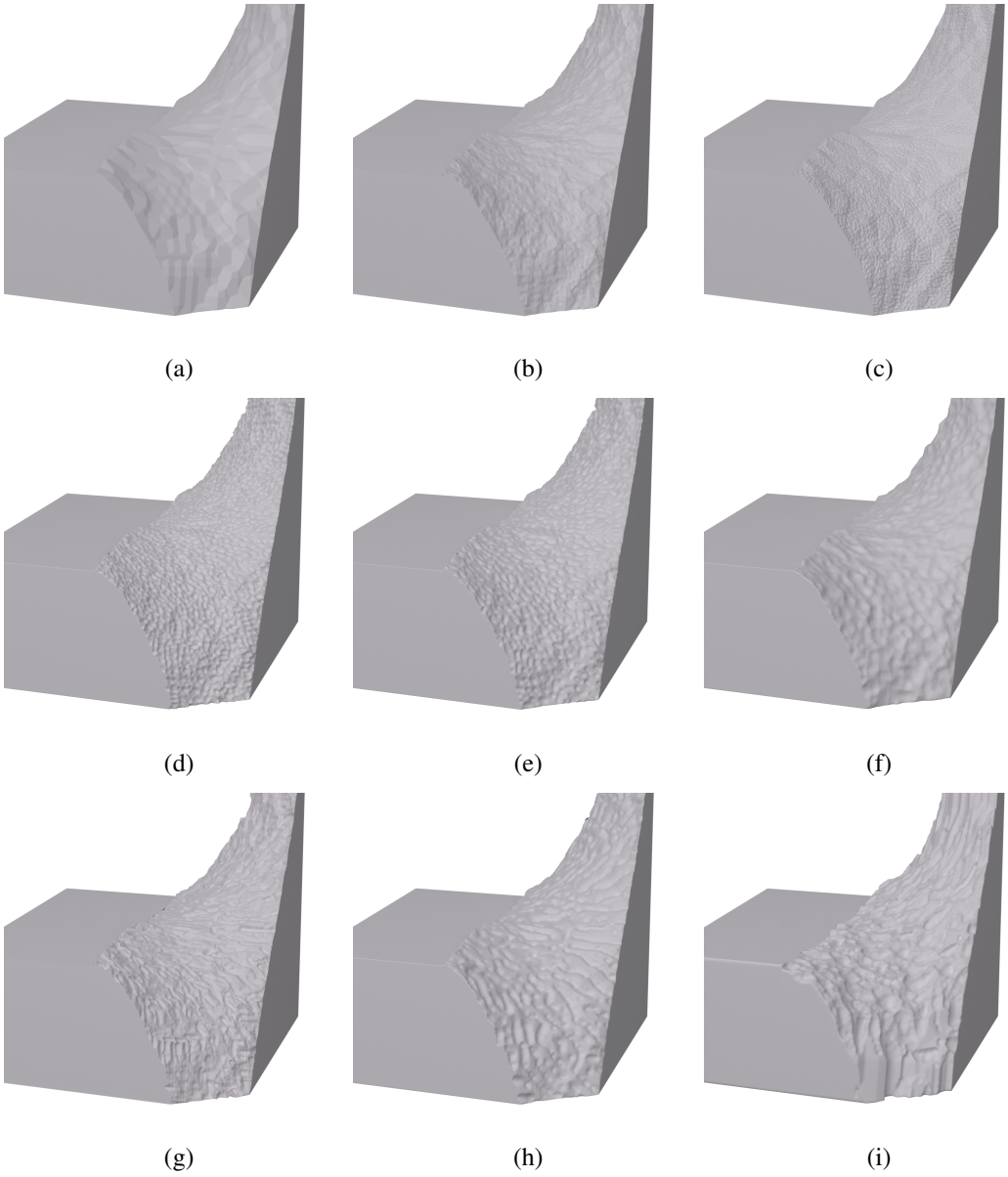
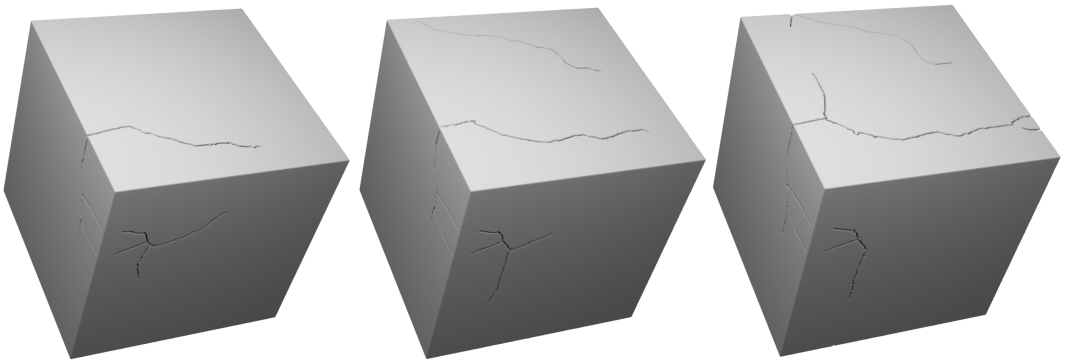


Figure 3.20: Fracture surface detail on a Mode III loading-test result (*cf.* Fig. 3.14). (a) Default crack surface (Voronoi medial surface); (b)-(h): Depictions of detail-enhancement with increasing levels of severity using our algorithm that is described in Sec. 3.7. (e) Results by Chen et al. (Chen et al., 2014).

Fracture stopping criteria: In-line with the goal of allowing sufficient control over the simulation, we demonstrate that using thresholds to control when fracture propagation should stop is practical. Fig. 3.21 shows three examples where we vary the threshold governing when our MPM simulation should stop (proportion of particles that are fully damaged). Increasing the threshold correlates with extensions of partial



(a) (b) (c)

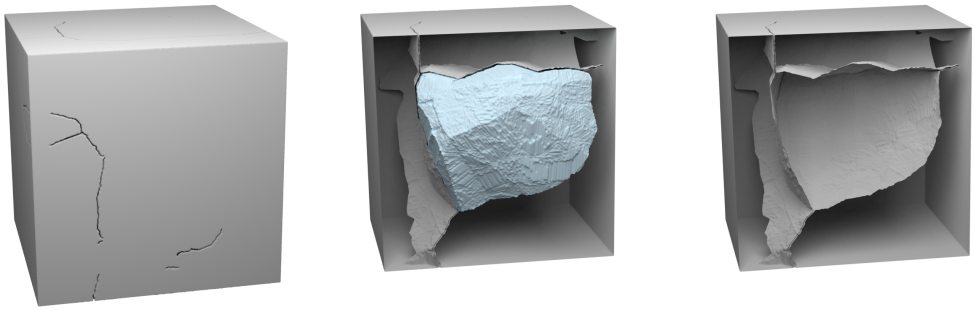
Figure 3.21: Using damage thresholds as a fracture stopping criteria. The images depict the resulting partial cracks corresponding to a percentage of damaged particles: (a) 10%, (b) 20% and (c) 30%. *Note: these partial cracks are used here only for demonstration purposes.*

fractures that are manifested on the surface of our model, providing a reliable solution to the problem of terminating fracture propagation. Notably, using a non-heuristic solution remains desirable for more control over the creation incomplete (non-through) fracture surfaces.

Interior void formation: Here we show that not only do we successfully simulate branching and partial cracks, we can also model the formation of voids, resulting from crack merging. An example is shown in Fig. 3.22. A current limitation however is that simulating such configurations results in popping artefacts in the rigid body system, which are caused by unstable contact solvers.

3.8.2 Dynamic rigid body fracturing

Besides our verification tests and comparisons we performed additional simulations with dynamic rigid body motion and fracture. Thus, we summarise the results of our rigid-body-to-CDM coupling in this section.



(a) Surface view (b) Interior fragment (c) Interior void

Figure 3.22: Here we show that our method also inherently supports the formation of interior voids due to crack bifurcation and merging.

Figure 3.23: We break a 1.2cm thin glass and show its resulting fragments (shards). This illustrates the diverse ability and strength of our coupling of local CDM with rigid-body methods, as fragments are formed from diverging and merging cracks.

Fig. 3.23 presents an example of glass breakage that is attainable with our method, capturing the intricate behaviour of cracks in thin materials. In Fig. 3.24 each fracture simulation is discretized with just 16k particles to produce over 430 fragments following impact between the Stanford bunny and a high-velocity bullet. Fig. 3.25 presents a simple setup where three armadillos are smashed against a wall: By also varying our rigid body impulse threshold to initiate fracture, we can control the num-

Figure 3.24: A bunny is hit by a bullet.

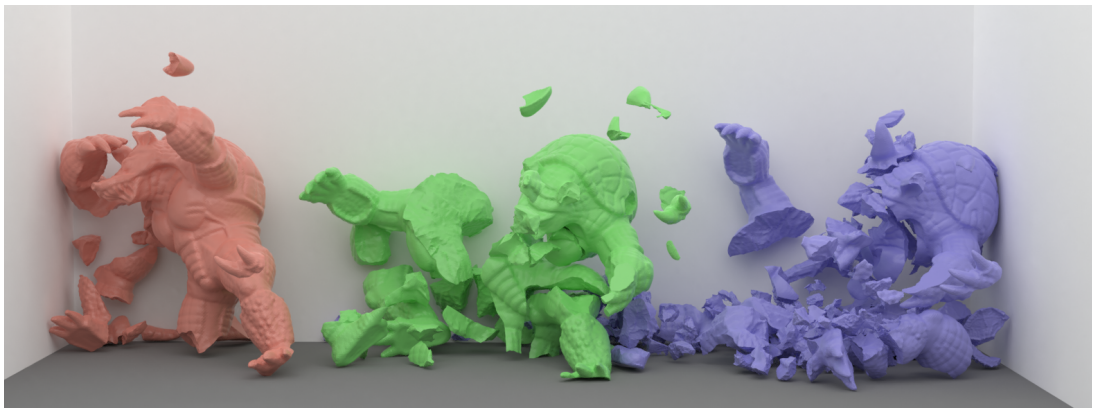
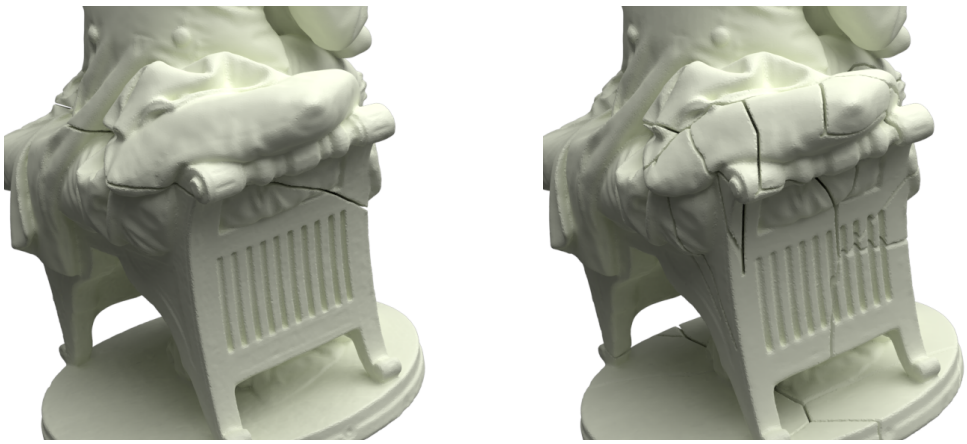


Figure 3.25: Controlling the number of fragments: Three armadillos are smashed against a wall, with each shattering according to the impulse threshold RB_{imp}. From left to right, we use 1e4, 5e3, 1e3, respectively).

ber of fragments while setting the minimum prescribed a number of fragments per fracture simulation to a default value of 1. The model on the left is fractured into just 13 fragments and the middle into 28 while the model on the right has 267 fragments to demonstrate variation. Further examples are also shown in Fig. 3.28 (c) and (d), and Fig. 3.1, which reveal the rich effects achievable with our method. These examples show that our method is able to robustly handle complex fractures while simulating on intricate shapes to produce sharp pieces with shatter effects.

We also summarise overall performance in Tab. 3.3, which shows that our method runs in reasonable time (at most 5.75 hours). For reference, CD-MPM (Wolper et al., 2019), takes over 30 hours to simulate 10 seconds of the 'crab' demo (with over 900k particles). Our method achieves 5x speedup. Note that the crab's material is relatively soft, which is, nonetheless, the most rigid one in all their demos. This is because CD-MPM uses a fully elastodynamic simulation. Simulating an ideal rigid object requires



(a) XFEM Chitalu et al. (2020) (b) Ours

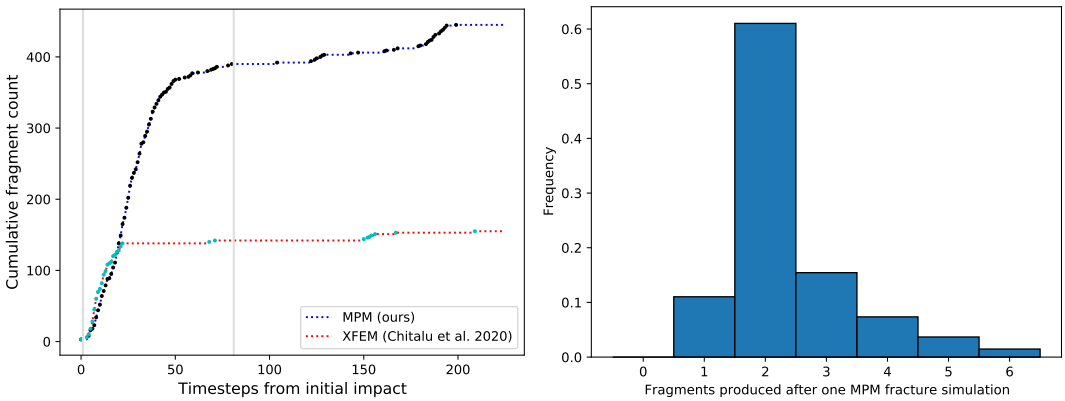
Figure 3.26: Crack-bifurcation comparison: We propagate multiple cracks per fracture, whereas XFEM handles at most one. Scene: 'Jenner' statue falls from a pedestal to initiate fracture (See supplementary video).

an extremely tiny timestep size, e.g., 1.0E-7, which is inherently impossible. Otherwise, tens of thousands of hours are needed even for the simplest scene. Therefore, a direct comparison with CD-MPM is not viable.

We also compare our method with XFEM (Chitalu et al., 2020) (Fig. 3.26 and Fig. 3.28) to evaluate the visual depiction of crack branching and merging behaviour, as well as general rigid body fracture effects. Our method is competitive with XFEM and offers the advantage of propagating several cracks per fracture while XFEM handles at most one. An example case in which such propagation behaviour will be useful is in depicting dynamically evolving cracks for visual effects. Fig. 3.26 demonstrates benefit of our method in this case, where branched cracks develop over the duration of one fracture to yield intricate propagation behaviour that can be grasped visually.

Overall breakage–as expressed in the total number of rigid body fragments–that will be achieved by Chitalu *et al.* (Chitalu et al., 2020) is also dependent directly on the number of fracture-initiating rigid body collision events because they handle one crack at a time[5]. In Fig. 3.28, we show that despite having the same material parameters, at most 80 fragments are produced with XFEM (a) whereas we produce up-to 246 fragments (b). Fig. 3.27 (a) plots the cumulative total number of fragments for the scene shown in Fig. 3.28, where our method demonstrates a superior fragment count. For this demo, we produce up to 2.75$\times$ the number of fragments as XFEM (*cf.* Fig. 3.28

[5]This is a strict upper-bound since XFEM (Chitalu et al., 2020) will generate at most 2^N fragments, where N is the total number of fracture-initiating rigid body collisions events.



(a) Cumulative total (b) Fragments per sim.

Figure 3.27: Fragment generation and distribution in Fig. 3.28. (a) Cumulative number of fragments generated over time. (b) Distribution of the number of fragments created per fracture simulation with our method.

(d)). Most fragments are generated in the time span starting from the initial impact and lasting 80 steps (see vertical lines in Fig. 3.27). In this span, we produce approx' 2.78× as many fragments per timestep (approx' 4.78) compared to XFEM (approx' 1.7). A frequency distribution for the number of fragments generated per fracture simulation using our method is also shown in Fig. 3.27 (b), which demonstrates that our fracture simulations typically produce two or more fragments (up to six here).

(a) XFEM Chitalu et al. (2020) (b) Ours

Figure 3.28: Animation comparison. XFEM Chitalu et al. (2020) is limited to generating at most two fragments per fracture, whereas we generate two or more fragments to allow higher fragment counts.

Fragment volumes: Fig. 3.31 shows the distribution of fragment volumes (from the simulation shown in Fig. 3.24), with comparison to Mott's formula (Elek and Jaramaz,

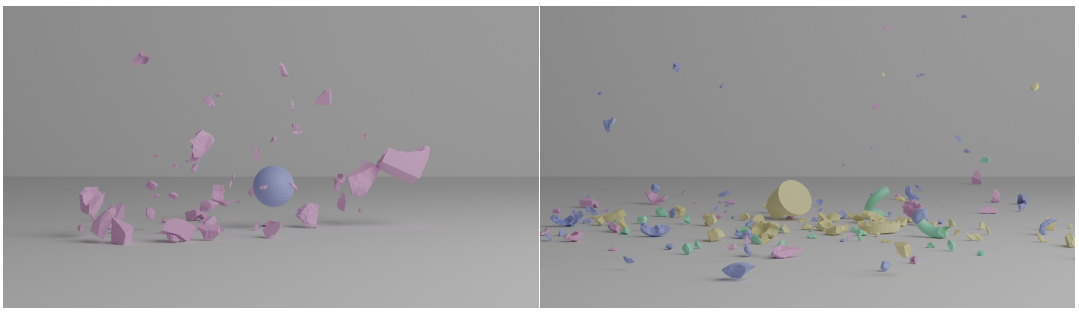


Figure 3.29: Four simple shape objects drop to the ground and shatter.

Figure 3.30: Two armadillos move towards each other and break into small fragments due to collision.

2008). A perceptively high number of fragments is produced, but extremely small-volume fragments are under-represented. We closely follow theoretical predictions for large fragments and we get a high number mid-sized fragments, but cannot represent 'tiny' pieces. One explanation is that we tend to have large expanses of damaged regions that are then compounded into a single crack surface during Voronoi extraction (Sec. 3.3).

Also, partial cracks–resulting from our fracture stopping criteria–are discarded when we construct Voronoi tessellation regions since fragment extraction (Sec. 3.5) will only identify whole fragments. This reduces the likelihood of incomplete fractures–as represented by hairline gaps as in Fig. 3.21–to then influence the creation of even smaller fragments since crack propagation no longer stops at interior boundaries. It is possible too that small-to-medium sized fragments have insufficient mass for generating sufficient impulse to initiate further breakage.

The smallest fragment(s) produced with our experiments is on the order of approx-

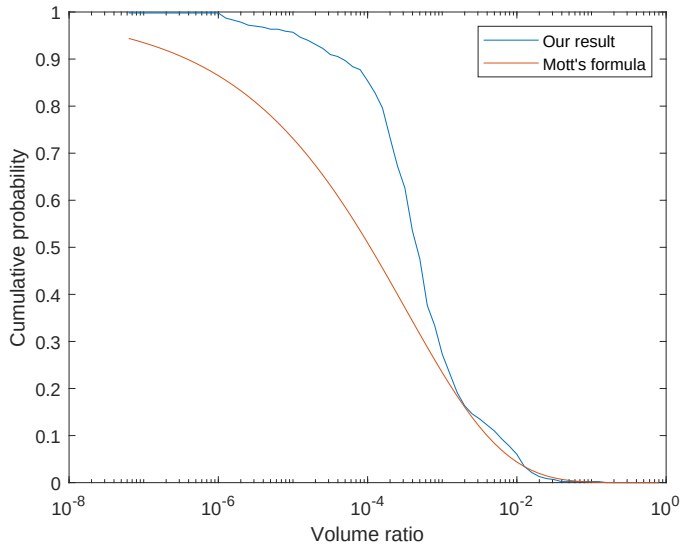


Figure 3.31: Comparison of fragment volume distribution with Mott's Formula Elek and Jaramaz (2008).

imately 10^{-6} (volume ratio), as shown in our analysis of Mott's Formula in Fig. 3.31. This analysis is based on the bunny demo of Fig. 3.24 which provides the finest breakage and the second-lowest particle count. In this analysis, the lowest fragment's volume ($3.51E-9$ units) is approximately 0.5E-3 times the original volume ($7.01E-4$ units), where the grid spacing is $7.05E-3$ units. It is possible to produce even smaller fragments since our result is simply dependent on the nature of arising contacts/interactions. Thus, in principle we are limited only by the rigid body configuration. As noted within our discussion, fragment size would also be smaller if partial cracks are kept since propagating cracks would be able to terminate on interior boundaries, thereby producing finer volumes.

3.9 Conclusions

We have presented a simulation of brittle dynamics fracture using a material point method. We compute stresses on material particles by introducing the usage of damage variables that are updated local to each particle. Both crack initiation and propagation are represented by these damage variables as smeared paths in the material which evolve over time. We have also described an approach to guarantee material separation at grid-cell resolution, enabling discontinuous velocities fields for increased accuracy independent of kernel-function choice. Our approach can simulate complexly branching cracks while avoiding explicit meshing procedures during propagation.

Scene	Vox	ρ	N_p	RB_{imp}	$RB_{\#frags}$	$MPM_{\widetilde{\#\Delta t}}$	$MPM_{dthresh}$	$MPM_{\widetilde{\Delta t}}$	$MPM_{\#\Delta t}$	MPM_{tot}	$RB_{\#\Delta t}$	RB_{tot}	Sim_{tot}	d	l_c	s	h_c	G	m
Fig. 3.24	0.005	2800	16k	5.0E3	434	45	0.35	9.13E-2	11264	1.01E2	600	6.28E3	1.03E4	0.2	50	1.0	2.0	2.0	0.05
Fig. 3.25	0.002	7000	20k	4.0E3	245	98	0.35	9.21E-2	13483	1.25E3	700	9.96E2	4.12E3	0.2	60	2.0	2.0	2.0	0.1
Fig. 3.23	0.005	2450	1.06M	1.0E4	104	30	0.35	4.92E1	3000	1.50E4	1800	3.60E3	1.88E4	1.0	50	1.0	1.0	0.25	0.1
Fig. 3.28b	0.002	5800	16k	1.0E3	246	83.3	0.35	1.10E-1	39407	4.32E3	1200	6.67E2	1.20E4	0.2	60	1.0	2.0	1.5	0.05
Fig. 3.1	0.003	5800	50k	6.0E2	458	101	0.35	2.80E-1	32835	9.24E3	1400	9.78E2	1.79E4	0.2	60	2.0	2.0	2.5	0.1

Table 3.3: Performance summary. Columns: VDB voxel size (Vox); material density ($\rho(kg/m^3)$); MPM particles (N_p); Rigid body impulse threshold to initiate fracture (RB_{imp}); Final number of rigid body fragments ($RB_{\#frags}$) Average number of MPM timesteps per fracture simulation ($MPM_{\widetilde{\#\Delta t}}$); Prescribed minimum number of fragments per fracture simulation ($MPM_{dthresh}$); Average CPU wall-clock time per MPM timestep ($MPM_{\widetilde{\Delta t}}$); Total number of MPM timesteps ($MPM_{\#\Delta t}$); Total CPU wall-clock time spent on MPM simulation (MPM_{tot}); Total number of rigid body timesteps ($MPM_{\#\Delta t}$); Total CPU wall-clock time spent on rigid body simulation (RB_{tot}); Total CPU wall-clock time to run simulation, including I/O, crack extraction, mesh processing etc. (Sim_{tot}); The columns d , l_c , s , h_c , G , and m represent fracture detail parameters (*cf.* Tab. 3.2). The remaining parameters that we use are as follows: Young's Modulus ($E = 3.2E8$); Poisson's ratio ($\nu = 0.2$); MPM timestep size($\Delta t_M = 1.0E-7$); Mode-I fracture energy ($G_f = 200(J/m^2)$); Principal failure stress ($\sigma_f = 200$). The characteristic length l_{ch} is set to the grid-voxel diagonal. All timings are given in seconds.

We define our non-manifold crack geometry, which we extract from the simulated damage, as a medial surface of a Voronoi tessellation. We select Voronoi faces according to their embedding in the damaged regions and protrusions with the domain boundary to allow for material separation. The resulting mesh(es) from simulation are then enriched, with our novel procedures to trace fracture patterns using particle damage time. Consequently, our method is able to produce detailed effects amenable to artistic control, which when combined with rigid body methods, allow for a rich set of dynamic rigid shatter effects.

We generally follow the model of Chen *et al.* (Chen et al., 2014), combining a low-DOF simulation with crack extraction and adding fracture detail, but we offer a distinct formulation using material points combined with a novel surface extraction algorithm whose output is enriched with simulation-dependent fracture detail. Our damage-time vector fields allow for a simulation-dependent and controllable solution to the problem of 'adding noise' to fracture surfaces, emulating Lagrangran crack-front motion with material particles to trace chevron lines. This approach offers a practical advantage over the use of textures (material strength fields), which must be produced separately of simulation (e.g., via Perlin noise), or even manually. Moreover, the resulting detail that is produced outside of simulation will not correlate with actual simulation, thus hindering achievable results.

The significance of our approach is also highlighted when we compare with other

techniques representing the state-of-the-art. Unlike FEM (e.g., (Pfaff et al., 2014; Koschier et al., 2014)), we handle crack branching and merging without re-meshing, where complete and partial fractures are handled implicitly. Qualitatively similar results as XFEM (Chitalu et al., 2020) are produced but XFEM is limited in the number of fragments generated per fracture. Our main advantage over BEM (Hahn and Wojtan, 2016, 2015) is in handling materials of thin dimensions, where BEM is limited (see also Chitalu *et al.* (Chitalu et al., 2020) for a further discussion). Our method is also amenable to modelling heterogeneous material, which BEM does not support. Further, since BEM applies cracks as thickened solid volumes, it is susceptible to producing thin hairline gaps removing thin strips of material, which are occasionally desirable but rarely seen within brittle fracture (and will induce volume loss). Conversely, our method is free from such partial cracks while yet retaining the ability to incorporate them into visual results when desired. Fracture simulation using fully-elastodynamic MPM simulation (as in e.g., (Wolper et al., 2019)) is impressive but slow, typically requiring particles in the scale of hundreds of thousands or millions, whereas we typically require just tens of thousands. Fully-elastodynamic MPM also struggles to produce fast and rigid shatter effects due to sticky artefacts caused by artificial plasticity, which we overcome thanks to our CDM-to-rigid-body coupling with extracted cracks.

Figure 3.32: A failed cut where a crack leaves the domain and re-enters. The crack is initialized in the right leg. We extract crack lines but cannot constrain them to the right leg since it is a concave shape. The protruded crack faces cut the left foot even though there is no impact there.

Limitations and future work: While we simulate damage on material particles, we

extract crack surfaces according to a Voronoi diagram. Crack surface contours therefore tend to be strongly influenced by the structure of the extracted Voronoi faces that form the original medial surface. This can lead to noticeably flat regions on parts of the crack surface (e.g., tail of bunny in Fig. 3.3 (c)), but we found that to our solution to enhancing fracture detail can mitigate such artefacts reasonably well.

Likewise, the quality of our crack surface *approximation* is dependent on grid-resolution, and the shape and thickness of the damage region(s): Particles are effectively compounded into a medial surface as a best-case solution to the ambiguous problem of placing a surface within (and outwith) a damage-volume. Thus, harnessing additional data, e.g., on damaged particles themselves, to inform crack placement is ideal.

Our Voronoi diagram may also be seeded with particles that define the boundary of our material-damage region(s) but this will introduce multiple extra steps. We would need to first isolate each fully-connected region of damaged particles. Then the surfaces that surround each damaged region need to be computed by tessellation, implicit surfaces etc. We also need to exclude the faces that overlap with the domain boundary so that the medial axes can cut through the domain. These will require extra computation and the parameters may need to be carefully tuned to produce satisfactory results. It is worth noting too that artefacts (flat patches) on the extracted cracks would also be more-or-less the same when using particles for generating the Voronoi diagram since we extract cracks in material space where our grid spacing $h_{\mathrm{v}} \equiv h$ is computed from total volume and the total number of particles as in our simulation.

Fragment extraction (Sec. 3.3) can be limited for clipping protruded crack faces, which may create additional pieces as a result of not clipping crack faces that leave and re-enter the domain. Moreover, our routine is designed to work with a dilated volume, which encloses the domain, where this volume is either a convex envelope or a (possibly concave) wrap-around of the domain. The latter dilated wrap-around permits intersections that respect the domain boundary but requires modifying Voronoi structures to re-map and merge clipped polygons of cells which is tedious to implement. For scenes allowing partial cuts, e.g., Fig. 3.26, the dilated volume can effectively clip protruded cracks and prevent crack re-enter. Conversely, a convex envelope (e.g., dilated domain bounding box) is more easily implemented with existing routines (e.g.,via Voro++ (Rycroft, 2009)) but it is suited for convex shapes to prevent crack faces that protrude through the domain boundary from re-entering. Our implementation used the convex envelope as we found its issues to be rare but for Fig. 3.25, where an escaped

part of the crack surface re-enters the domain to intersect the left foot. We also show the cut in an earlier frame in Fig. 3.32.

In addition, spontaneous stress distributions throughout the domain during propagation can serve to exacerbate what appear to be spontaneous fractures (e.g.,Fig. 3.25) that occur away from impact points. These are dependent on rigid body contacts giving rise to dynamic crack paths that are also influenced by material properties.

To propagate cracks we use a time-dynamic, explicitly integrated MPM simulation which is thus only stable for time steps within a CFL limit. However, we typically require just a small number of particles per fracture simulation (e.g., 50k), which serves to alleviate concerns about performance.

Our method applies simulated cracks in material space, therefore it is inherently inadequate for ductile fracture. Despite this, the ability of our approach to allow discontinuity complements ductile material models, which is useful for a two-way coupling as in Hu *et al.* (Hu et al., 2018), where cracks are simulated in world space but as a fully elastodynamic simulation.

While our damage time fracture detail algorithm produces rich artistic control capabilities (*cf.* Tab. 3.2), achieving the desired output can be a challenge. Our parameters provide only supplementary control since the chevron mark directions are governed by damage time variables resulting from simulation. This supplementary control, however, is able to produce interesting effects as shown in Fig. 3.20, making it a practical solution for enriching simulation results.

The parameters within our method could be more intuitive, and the development of reformulated parameters may assist with enabling such an interface. However, tuning is made easier by the fact that the effect of each of our parameters is independent from others and with low sensitivity to minor perturbation for facilitating control.

Our glass demo (Fig. 3.23) works well to produce visible ring-like 'wave' patterns but requires many particles (e.g., one million). Our grid size in the depth (or thickness) dimension must span at least three cells for correct evaluation of force derivatives using the quadratic kernel. This entails having at least six layers of particles (176k per layer) where a layer spans $0.84\text{m}\times0.84\text{m}$. Furthermore, as rigid-body dynamics are currently handled as a post process in this demo, we manually specify Dirichlet boundary conditions along the border to ensure desired breakage. We found that cracks in thin materials tend to be under-resolved with *point* loading from a rigid body contact, causing them to stop prematurely due to a weakening of tensile stress in directions that are perpendicular to the load. We currently solve this problem by distributing the

load on a circular area around the point of contact, which works well but propagation patterns around this point are then dependent on the damage within the area. Thus, finding a general solution to the problem of exacerbated damage around contact points is suggested for future work.

Objects with a high volume-to-surface-area ratio can also have large regions that are fully damaged (e.g., the bunny tail in Fig. 3.3); such regions still remain even when using multiple velocity fields as shown in Fig. 3.18. As we only weaken the tensile stress component, the possibility for weakening the compressive component of stress is suggested for future work to minimise the thickness of damaged regions. In terms of visual expectation for such regions, animators may prefer to see a large amount of small fragments, while our medial surface-based cutting algorithm produces rather large fractions, whose sizes and shapes depend on the way the branches spread in the rest of the volume. We can possibly apply non-physical fracture patterns based on Voronoi diagrams (Schvartzman and Otaduy, 2014; Su et al., 2009) to for producing many small fragments.

Chapter 4

Thin shell object fracture

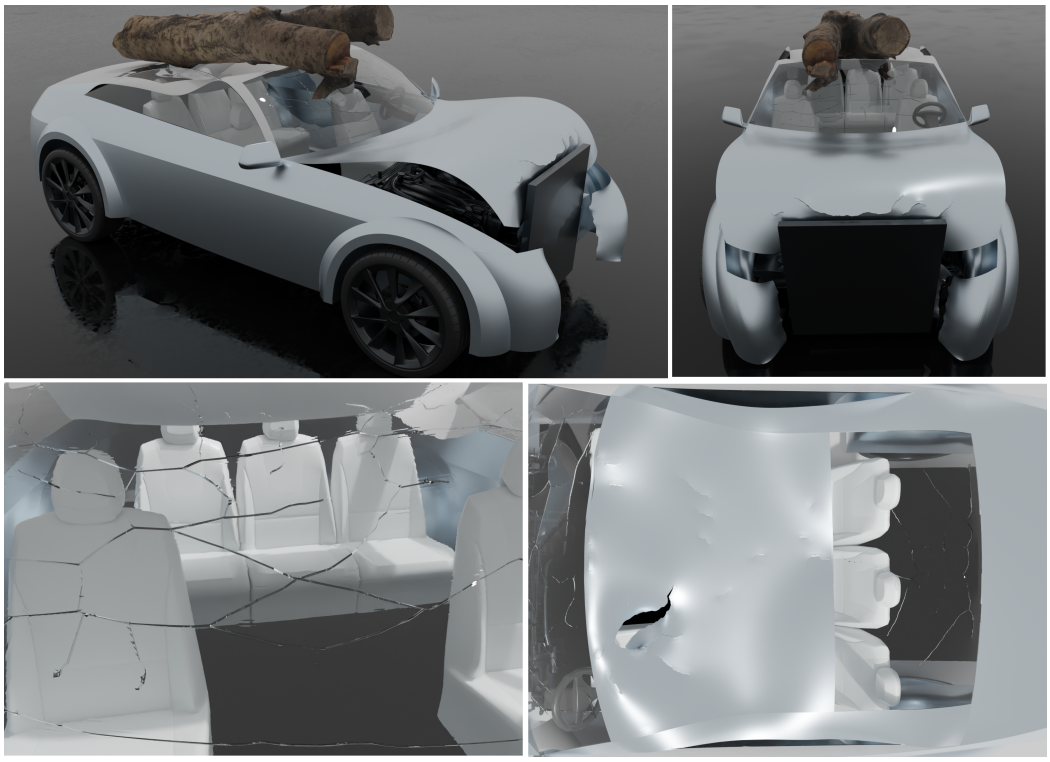


Figure 4.1: Our hybrid Lagrangian/Eulerian method can effectively simulate complex objects made of various materials in challenging scenarios. A sports car composed of 12 NURBS patches is simulated. At first, a heavy impactor hits the bumper and breaks it into three pieces. The bonnet bents under compression, which leaves permanent deformation and fractures. Then a tree falls down towards the car's top. One small branch hits the front window and generates branching cracks. Note that the front window's fragments don't fly away upon breaking as we want to simulate laminated glass instead of regular glass, see also Fig. 4.29. The car's back window also shows a similar fracture pattern. Its top is made of rigid steel which gets permanent deformation and cracks like the bonnet. A small branch penetrates the top and leaves a big hole on the left side.

4.1 Introduction

Simulating volumetric thin-shells like cloth, rubber and sheets of metal is at the core of many applications in computer graphics, including complex visual effects in film. Unfortunately, most thin-shell formulations assume a topologically-fixed continuum, which makes them challenging to simulate with fracture effects. In particular, the recent hybrid Lagrangian/Eulerian formulation (Jiang et al., 2017; Guo et al., 2018) can simulate complex effects including bending resistance and (automatic) contact handling but is confined to a continuum setting without tearing nor fracture effects in elastic materials. To overcome this issue, alternative strategies have been developed for inducing this *discontinuity* of field quantities using a variety of mesh-based methods. For instance, some techniques replace the continuous assumption with a discrete shell model at the cost of simplifying measures of bending (i.e., flexural) energy over mesh edges and with separate treatment of contact (Grinspun et al., 2003). Other methods propose using thin plates (Busaryev et al., 2013), or 2D finite elements (Pfaff et al., 2014) with stress-based explicit remeshing and refinement for handling crack propagation.

In this chapter, we resolve continuum shell fractures and tears by modelling the evolution of a time-dependent damage field with CDM. In Chapter 3, we consider the case of brittle fracture in 3D volumetric solids, but this formulation is based on a rigid-body simplification via coupling and thus does not easily generalize to soft materials represented as thin shells. Conversely, other attempts have focused on soft volumetric solids to derive dynamic material damage evolution. In particular, Wolper et al. (2019) simulate dynamic fracture involving large elastoplastic deformation, while their follow-up work (Wolper et al., 2020) adopts a geometric approach (Wang et al., 2019a) to crack-modelling for simulating fracture in anisotropic materials. Our method is tailored toward continuum shells based on the Lagrangian/Eulerian formulation of Guo et al. (2018), but goes further and establishes a new model for simulating detailed and consistent crack-paths for a variety of materials characterized by a high ratio of width to thickness.

Our key contribution is a novel formulation of fracture in thin shells that can be used to simulate effects in soft and near-brittle materials that are represented as continuum shells. These effects can be easily incorporated into existing hybrid Lagrangian/Eulerian techniques, thus providing dynamic crack merging and bifurcation behaviors. To achieve this, we depart from previous methods by also capturing strong

(i.e., material) discontinuities using damage phase-fields: We combine Kirchhoff-Love elasticity, NURBS surfaces and a MLS approximation for simulating the propagation of arbitrary crack paths. This *fracture* simulation is simulated by tracking the evolution of phase fields as a function of stress, where complex crack merging and bifurcation behaviors are also handled automatically. To address the over-stretching artefacts, we update the phase fields based on the current stress state thanks to our explicit crack representation. An algorithm for ensuring consistent crack-paths is also presented, where newly traced paths over time (extracted from temporal phase-fields) are an exact extension of the previous time-steps. We then couple this explicit crack geometry with the MLS method for interpolating discontinuous field quantities: MLS is especially suited for estimating functions over a domain given a set of pointwise values of this function. Its use the vicinity of the crack allows us to interpolate field quantities over arbitrarily shaped sub-domains in the co-dimensional manifold, e.g., as determined by the crack placement. The result is an efficient approach to decoupling fragments in material space, seamlessly confining the transfer of information between control- and quadrature-points to only those that reside on the same side of the fracture surface in this space. To simulate complex arbitrary surface geometries, we also present a novel NURBS patch coupling method, which guarantees C^1 continuity between patches to ensure that interpolating functions (and their derivatives of order less than or equal to two) are square integrable over the entire shell midsurface. The effectiveness of our method is demonstrated in numerous examples depicting a wide range of complex thin shell fractures.

The rest of this chapter is organized as follows. We describe our Kirchhoff-Love shell formulation in Sec. 4.2. Next, we outline our unified approach for simulating shell dynamics and crack propagation under the MPM framework (see Sec. 4.3). We then describe how we update damage using a novel phase field method that enables material healing to consistently extract the crack (see Sec. 4.4). This is followed by how we stably compute the stress that satisfies the Kirchhoff-Love zero transverse (normal) stress condition on cracked surfaces. We also show how we separate fragments in material space using meshfree MLS NURBS (see Sec. 4.6). Details about separating fragments in world space and multi-object contact algorithm are described (see Sec. 4.7). Then we introduce a new method to couple multiple NURBS patches (see Sec. 4.8). Finally, we describe how to get the physical rendering mesh from material points, NURBS patches, and cracks in Sec. 4.9. We present the experimental results in Sec. 4.10 and conclude the paper in Sec. 4.11.

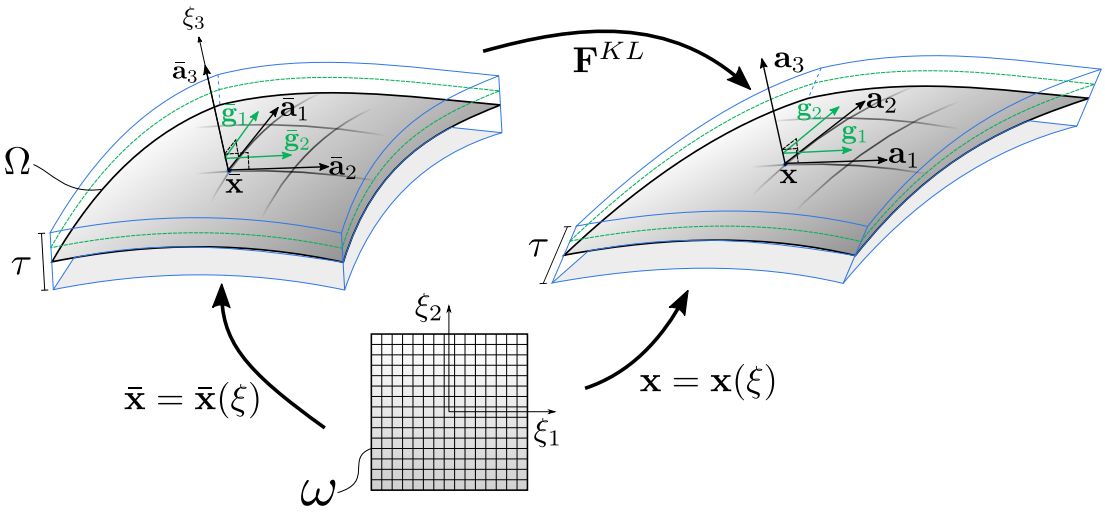


Figure 4.2: Mapping between parameter domain ξ, reference/undeformed surface (left), and deformed surface (right) of a Kirchhoff-Love shell. Boundaries of the shell are shown by solid blue lines. A shell layer is shown by dashed green lines.

4.2 Shell kinematics

In this section, we describe about the Kirkchhoff-Love formulation used for simulating thin shells, which is based on the model in Guo et al. (Guo et al., 2018). We begin by establishing the notation that we adopt in this chapter. The overbar $\bar{\cdot}$ indicates a variable is defined with-respect-to the reference/undeformed configuration. We also follow the general convention that Greek letter indices, e.g., α are used for 2D parameterization, thus taking values 1 and 2, and Latin letter indices, e.g., i are used for 3D parameterization, thus taking values $1,2$ and 3. Tab. 4.1 further summarises the other basic definitions and concepts that we use.

Consider a thin-shell with a mid-surface Ω and thickness τ as shown in Fig. 4.2. The kinematic motion of this shell is given by:

$$\begin{cases} \bar{\mathbf{r}}(\xi) = \bar{\mathbf{x}}(\xi_1,\xi_2) + \xi_3 \bar{\mathbf{a}}_3(\xi_1,\xi_2), \\ \mathbf{r}(\xi) = \mathbf{x}(\xi_1,\xi_2) + \xi_3 \mathbf{a}_3(\xi_1,\xi_2) \end{cases} \tag{4.1}$$

where $\bar{\mathbf{r}} \in \mathbb{R}^3$ is the *undeformed* position of a material point with curvilinear coordinates $\xi \in \mathbb{R}^3$, and $\bar{\mathbf{x}} \in \mathbb{R}^3$ is its position in the mid-surface i.e., with $\xi_3 \equiv 0$. The corresponding vector $\bar{\mathbf{a}}_3 = \frac{\bar{\mathbf{a}}_1 \times \bar{\mathbf{a}}_2}{|\bar{\mathbf{a}}_1 \times \bar{\mathbf{a}}_2|} \in \mathbb{R}^3$ is the normal to the mid-surface at (ξ_1,ξ_2), where $\bar{\mathbf{a}}_i = \frac{\partial \bar{\mathbf{x}}}{\partial \xi_i}$ is the tangent direction. The respective counterpart $\mathbf{r}$ (with $\mathbf{a}_3$ and $\mathbf{x}$) is the image under stretch and shear loading of the continuum shell.

Notation	Definition
ω	Parameter domain of shell.
Ω	Spatial/world domain of shell mid-surface.
$\mathbf{x} : \omega \to \Omega$	A mapping in the shell mid-surface.
$\mathbf{r} : \omega^\tau \to \Omega^\tau$	A mapping in the shell volume.
ξ	Coordinates in the parameter domain.
$\mathbf{a}_\alpha = \frac{\partial \mathbf{r}}{\partial \xi_\alpha}$	Covariant basis vector in the mid-surface.
$\mathbf{a}^\alpha = \frac{\partial \xi_\alpha}{\partial \mathbf{r}}$	Contravariant basis vector in the mid-surface.
$\mathbf{g}_i = \frac{\partial \mathbf{r}}{\partial \xi_i}$	Covariant basis vector in the shell volume.
$\mathbf{g}^i = \frac{\partial \xi_i}{\partial \mathbf{r}}$	Contravariant basis vector in the shell volume.
$\mathbf{a}_3 = \frac{\mathbf{a}_1 \times \mathbf{a}_2}{\lvert \mathbf{a}_1 \times \mathbf{a}_2 \rvert}$	Normal to the mid-surface.

Table 4.1: Notation and definitions of continuum thin shell.

4.2.1 Shell motion

Continuum shell motion $\mathbf{r}(\xi)$ is composed of two parts. These are the Kirchhoff-Love $\mathbf{r}^{KL}$ component and shearing/compression component $\mathbf{r}^S$ respectively that make up the shell motion as

$$\mathbf{r}(\xi) = \mathbf{r}^S(\mathbf{r}^{KL}(\xi)). \tag{4.2}$$

The Kirchhoff-Love model is characterised by two assumptions about the continuum shell. The first assumption is that fiber-lines which are normal to the shell mid-surface shall remain orthogonal to this mid-surface as the shell deforms. The second assumption stipulates that the thickness τ of this shell shall not change with this deformation to give

$$\mathbf{r}^{KL}(\xi) = \mathbf{x}(\xi_1, \xi_2) + \xi_3 \mathbf{a}_3^{KL}(\xi_1, \xi_2), \tag{4.3}$$

as the Kirchhoff-Love component of motion where $\mathbf{a}_3^{KL} = \frac{\mathbf{a}_1 \times \mathbf{a}_2}{\lvert \mathbf{a}_1 \times \mathbf{a}_2 \rvert}$ is a unit vector representing the normal at (ξ_1, ξ_2) with $-\tau/2 \leq \xi_3 \leq \tau/2$. This component shall therefore induce zero stress acting in the normal direction to the mid-surface, which is also known as the zero transverse (normal) stress condition. The shearing/compression component will induce a change of length *and* direction of fiber lines.

Given the volumetric mapping of the continuum shell from a reference to the de-

formed configuration $\phi(\bar{\mathbf{x}}) = \mathbf{r}(\bar{\mathbf{r}}^{-1}(\bar{\mathbf{x}})) : \bar{\Omega}^{\tau} \rightarrow \Omega^{\tau}$ its Jacobian

$$\mathbf{F} = \frac{\partial \phi}{\partial \bar{\mathbf{x}}} = \sum_{i}^{3} \mathbf{g}_i \otimes \bar{\mathbf{g}}^i, \tag{4.4}$$

is the deformation gradient representing a linear approximation of change-of-shape around $\bar{\mathbf{x}}$, where $\otimes$ is the tensor/outer product acting on the covariant $\mathbf{g}_i$ and contravariant basis vector $\bar{\mathbf{g}}^i$, respectively (see also Tab. 4.1). This definition of $\mathbf{F}$ permits the multiplicative decomposition as (*cf.* Eq. 4.2)

$$\mathbf{F} = \mathbf{F}^S \mathbf{F}^{KL}, \quad \mathbf{F}^S = \sum_{i}^{3} \mathbf{g}_i \otimes \mathbf{g}^{KL,i}, \quad \mathbf{F}^{KL} = \sum_{i}^{3} \mathbf{g}_i^{KL} \otimes \bar{\mathbf{g}}^i \tag{4.5}$$

where $\mathbf{g}_i^{KL} = \frac{\partial \mathbf{r}^{KL}}{\partial \xi_i}$ and $\mathbf{g}^{KL,i}$ are the covariant and contravariant basis vectors, respectively, that satisfy the aforementioned zero transverse stress condition (see also (Clyde et al., 2017)).

4.2.2 Elastic potential, yield condition and return mapping

With the deformation gradient, the total elastic potential is determined as a sum of local energy density contributions

$$\begin{aligned} \Psi &= \Psi^{KL} + \chi^S \\ &= \int_{\omega} \int_{-\frac{\tau}{2}}^{\frac{\tau}{2}} \left[\psi(\mathbf{F}^{KL,E}) + \chi(\mathbf{F}^{S,E}) \right] \left| \frac{\partial \bar{\mathbf{r}}}{\partial \xi} \right| d\xi, \end{aligned} \tag{4.6}$$

where $\psi(\mathbf{F}^{KL,E})$ and $\chi(\mathbf{F}^{S,E})$ are the elastic potential density functions due to Kirchhoff-Love and shearing motion, respectively. We use the shearing energy density function of Guo et al. (2018) and their return mapping algorithm for plastic yielding but extend their in-plane stress to satisfy the Kirchhoff-Love assumption when considering fracture (see Sec. 4.5).

4.3 Discretization using MPM

In this section, we first describe how we model surfaces with NURBS patches, and then summarise discretization of the continuous equations for simulation with the MPM, providing a high-level perspective for our specific contributions described in later sections. The overview of method is shown in Fig. 4.3.

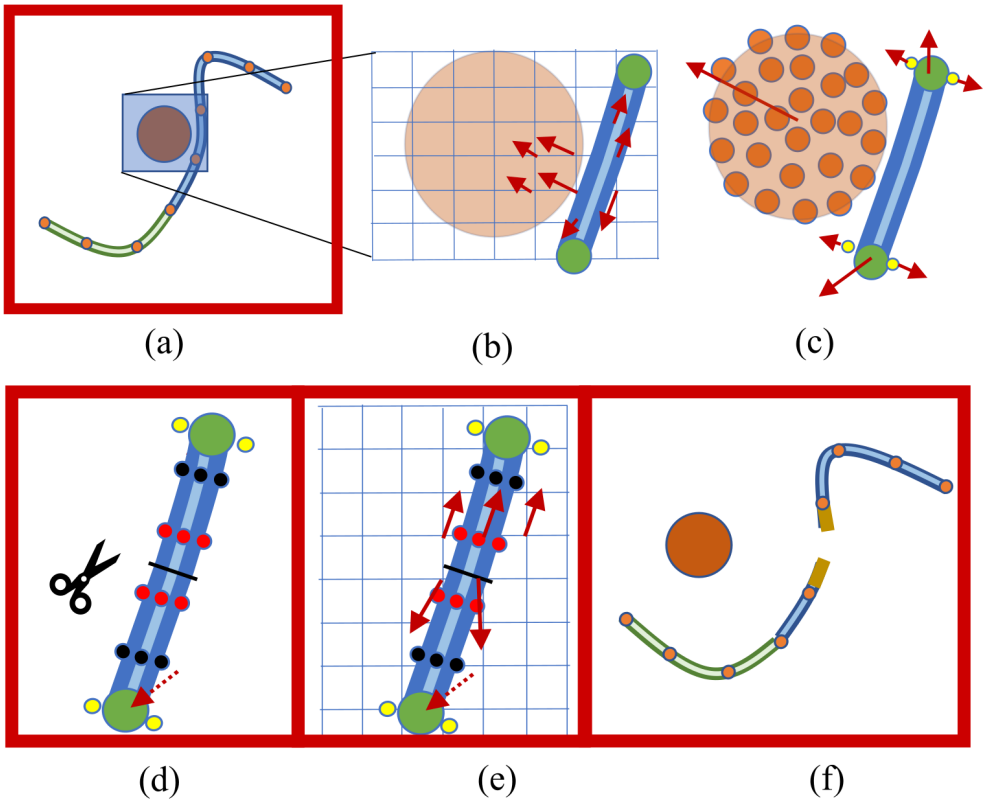


Figure 4.3: The system overview. The steps surrounded by the red boxs are where our contributions exist: (a) The thin shell is represented by three layers of thin shells to which MPM particles and quadrature particles are attached (see Sec. 4.2). The geometry of the shell is determine by the control points of the NURBS patch in the middle layer. Multiple NURBS patches composed of different material can be coupled using our formulation (see Sec. 4.8.) (b) PG Transfer/Momentum Update: Particle properties are transferred to the grid and internal forces are computed based on the deformation gradient. The grid velocity is updated accordingly. (c) GP Transfer, Advection: Particle locations and deformation gradients are updated. (d) Plasticity update, damage update and crack extraction: The stress is computed at quadrature particles and their damage is updated (see Sec. 4.4.1). A cut is made according to the damage at the quadrature particles (see Sec. 4.4.2). (e) In the next round of momentum update, the separation force is computed (see Sec. 4.7), resulting in the cut of the thin shell. (f) The geometry near the cut is represented by a MLS formulation (see Sec. 4.6).

4.3.1 NURBS-based Lagrangian representation

We represent the shell kinematics as a second-order NURBS surface satisfying C^1 continuity on the shell surface, which is required by the Kirchhoff-Love model. The NURBS surface is a representation that is widely used in the industry for object design; using the NURBS as a representation allows us to directly apply simulation to models designed for prototyping. Also, its polynomial nature allows us to not only design smooth, continuous surfaces with less computational cost, but also to well ap-

proximate the geometry of the cut/cracked region by a MLS representation, which is also a polynomial representation. Finally, its long history and rich knowledge in terms of its characteristics allows us various complex operations such as coupling them for producing complex continuous surfaces (Coox et al., 2017).

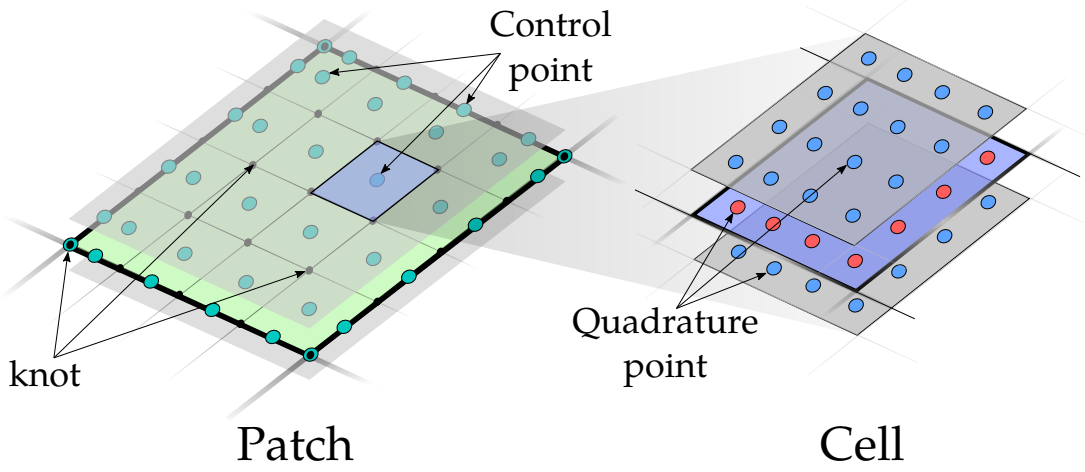


Figure 4.4: Schematic illustration of the discrete representation of our thin-shell as a NURBS surface. The surface of a simulated domain (e.g., cloth) is generally defined by several NURBS patches, but we consider only one here. A patch is composed of three layers, the mid-surface (denoted by green in the patch or blue in the cell) and the laminae as surfaces parallel to this mid-surface. Each layer is prescribed with equidistant local (patch) coordinates called knots, as well as the control points and cells. The control points determine the geometry of the patch. A cell corresponds to a structured collection of quadrature points, which assists with defining support radi in a patch during interpolation. The quadrature points are where we evaluate physical quantities like stress.

We now describe how NURBS patches are formed. Control points are regularly spaced across a single NURBS patch that is composed of cells as shown in Fig. 4.4. A cell represents a grouping of quadrature points in the shell. There are $(M-2)(N-2)$ of these cells in a patch, where M and N are the number of controls points along each axis spanning the parametric plane of the patch to give a total MN points. Furthermore, $(M-2)(N-2)$ of these control points will be spaced with interval l, from which the cell-size is determined as l^2. The control points at the boundary of the patch will be evenly spaced with half the interval $l/2$. Finally, the knots represent prescribed local coordinates in a patch, and are sampled at the corners/boundary of the cells (or triply duplicated at the boundary). In essence, we use this discrete NURBS surface representation for simulation, where control points will be the advected particles of the continuum shell in MPM.

4.3.2 MPM particle classification and simulation

Inline with Guo et al. (2018), particles are classified into four groups as: traditional particles for modelling non-thin shell objects, (I), control particles for modelling the thin shell objects (II), shearing/compression particles for modelling non-KL deformation of the thin shells (III) , and quadrature particles for evaluating physical properties of the thin shell (IV). At any given timestep n, the particles of types I, II and III will have position $\mathbf{x}_p^n \in \mathbb{R}^3$, velocity $\mathbf{v}_p^n \in \mathbb{R}^3$, initial mass $m_p \in \mathbb{R}$ and volume $V_p^0 \in \mathbb{R}$. Particle types are further characterised by the additional physical properties they carry. Type-I store a deformation gradient $\mathbf{F}_p^n \in \mathbb{R}^{3\times3}$ as in standard MPM, while Type-II particles are the NURBS control points, which define the geometry of the thin shells. Type-III store an elastic shearing basis vector $\mathbf{a}_{p3}^n \in \mathbb{R}^3$. And Type-IV store the Kirchhoff-Love component of the deformation gradient $\mathbf{F}_q^{KL,n} \in \mathbb{R}^{3\times3}$, its elastic component $\mathbf{F}_q^{KL,E,n} \in \mathbb{R}^{3\times3}$ and plastic component $\mathbf{F}_q^{KL,P,n} \in \mathbb{R}^{3\times3}$. A damage variable $c \in \mathbb{R}$ is also stored at these Type-IV particles, with another variable c^{his} as the largest value of c over the history of crack evolution (see Sec. 4.4.1 for detail).

During simulation, Type-IV quadrature particles will not directly participate in the mass and momentum transfer of MPM but are instead only used to store the elastic deformation state. We sample a 4×4 stencil of these particles in each of the three layers of a shell, per cell, which represent the mid-surface, lower and upper lamina (see Fig. 4.4, right). To accommodate shearing, a Type-III particle is also assigned to each cell with the same parameter coordinates as the Type-II particle (control point) at the center.

Our simulation is based on standard MPM discretization (Sec. 2.2.2) but make several changes based on our contributions to account for the aforementioned particle grouping, and fracture. We also adopt the PIC formulation (rather than APIC Jiang et al. (2015)) as it is inherently free of angular momentum terms that are most suitable for simulating fluids due to overly-energetic motion that do not necessarily favour simulations like cloth (*cf.* Fig. 4.26). The changes are summarized in the paragraphs that follow, where we use notations $\Gamma^{(I)}$, $\Gamma^{(II)}$, $\Gamma^{(III)}$, and $\Gamma^{(IV)}$ to denote sets of the four types of particles, respectively.

1. Particle to grid transfer: We only transfer Types I, II and III particle's mass and momentum to grid nodes as Types IV particles are massless quadrature points.

2. Compute force contributions: The internal elastic force is computed as:

$$\mathbf{f}_{\mathbf{i}}^{\text{int},n} = \mathbf{f}_{\mathbf{i}}^{(\text{I}),n} + \mathbf{f}_{\mathbf{i}}^{(\text{II}),n} + \mathbf{f}_{\mathbf{i}}^{(\text{III}),n} \tag{4.7}$$

which gathers contributions from respective particles. The term $\mathbf{f}_{\mathbf{i}}^{(\text{I}),n}$ is the canonical MPM force term of Type-I particles, which is evaluated from the elastic potential ψ (we use the Neo-Hookean model). The generalized force from Kirchhoff-Love quadrature points (via Type-II particles) is given by

$$\mathbf{f}_{\mathbf{i}}^{(\text{II}),n} = -\sum_{p\in\Gamma^{(\text{II})}} w_{\mathbf{i}p}^{n} \sum_{q\in\Gamma^{(\text{IV})}} V_q^0 \tilde{\mathbf{P}}(\mathbf{P},c) : \frac{\partial \mathbf{F}_q^{KL,Etr}}{\partial \mathbf{x}_p^{KL}}. \tag{4.8}$$

We have $\mathbf{P} = \frac{\partial \psi}{\partial \mathbf{F}^{KL}}(\mathbf{F}_q^{KL,Etr})$ as the *trial* first Piola-Kirchhoff (PK1) stress that is determined from the trial elastic deformation gradient $\mathbf{F}_q^{KL,Etr}$. This trial PK1 stress doesn't satisfy the zero transverse normal stress condition of the Kirchhoff-Love model. We, therefore, need its in-plane projection, which we obtain by calculating the corresponding trial Cauchy stress $\boldsymbol{\sigma} = \frac{1}{\det(\mathbf{F}^{KL})}\mathbf{P}(\mathbf{F}^{KL})^{\intercal}$ and its in-plane components (see Sec. 4.5). It is from this trial in-plane PK1 that the weakened PK1 stress $\tilde{\mathbf{P}}$ is evaluated with damage c (see Sec. 4.4.1).The remaining term $\mathbf{f}_{\mathbf{i}}^{\text{III},n}$ is the generalized shearing/compression force. We refer readers to Guo et al. (2018) for details.

3. Grid to particle transfer: Particle positions are then updated using

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1}, \quad p \in \Gamma^{(\text{I})} \cup \Gamma^{(\text{II})}, \tag{4.9}$$

and

$$\mathbf{x}_p^{n+1} = \sum_{s\in\Gamma^{(\text{II})}} \mathcal{W}_s^{\text{NURBS}} \mathbf{x}_s^{n+1}, \quad p \in \Gamma^{(\text{III})}, \tag{4.10}$$

where $\mathcal{W}_s^{\text{NURBS}}$ is the NURBS interpolation function from a Type-II particle p to a Type-III particle s. For the vicinity of cracks, we instead use MLS as the representation for representing the crack geometry (see Sec. 4.6). We also provide an algorithm to stitch multiple NURBS patches with C^k continuity in Sec. 4.8 such that shells with different physical properties can be coupled.

4. Update particle's deformation gradient and plasticity: Type-I particles' deformation gradient is updated as usual. For Type-III and Type-IV particles, Return mapping is used to calculate the corrected plastic strain satisfying the plasticity yield condition: We use the return mapping method of Guo et al. (2018) for Type-III particles to model shearing/compression resistance but adopt the associative flow rule of Ambati et al. (2018) on Type-IV particles to model bending resistance with fracture.

5. Damage update and crack extraction: The damage state c and history c^{his} of each quadrature point are updated (*cf.* Sec. 4.4.1), where c^{his} is then used to extract crack geometry at a given timestep which we describe in Sec. 4.4.2.

4.4 Co-dimensional fracture

Sec. 2.3 describe how to propagate cracks using the local phase field by weakening the Cauchy stress. This model works well for brittle object fracture simulation even though the crack path is thick for two reasons. On the one hand, the brittle object has little deformation due to the material's high stiffness. On the other hand, we render the fragments by projecting them back to material space, which has no deformation at all. However, this is not the case for thin shell object fracture simulation as we render the fragments in their world space. As we are simulating thin shells with varying material stiffness, the excessive crack path in the low stiffness material causes undesirable over-stretching artefacts.

In this section, we present a detailed account of our approach for simulating fractures on thin shells. First, the damage in the domain is calculated using the phase field method (Sec. 2.3). However, the damage phase value is updated in a different way to enable material healing. Then, a consistent crack is explicitly computed to separate the fragments. Finally, we represent geometry in the crack-influenced region with MLS approximation to achieve arbitrary branching cutting.

4.4.1 Calculating stress and damage

Here we describe how we compute the damage c and how we use it to compute the weakened PK1 stress $\tilde{\mathbf{P}}$ for Eq. 4.8. Our damage calculation is based on the local phase field method Cervera and Chiumenti (2006), but we enable material healing to avoid over-stretching artifacts.

Without damage, the weakened PK1 stress $\tilde{\mathbf{P}} \equiv \mathbf{P}$ that is stored on Type-IV particles will be equivalent to the trial PK1 in Eq. 4.8. To measure in-plane stress, we require the Cauchy stress $\boldsymbol{\sigma} = \frac{1}{\det(\mathbf{F}^{KL})}\mathbf{P}(\mathbf{F}^{KL})^T$ which is a descriptor of force density in world space. Upon calculating this descriptor, we remove normal components to leave only the in-plane components as described in Sec. 4.5, which is necessary to satisfy the Kirchhoff-Love zero transverse normal stress condition. The effective (in-plane) stress is then determined by $\overline{\sigma} = \sigma_1$ via singular value decomposition $\boldsymbol{\sigma} = \mathbf{U}\Sigma\mathbf{V}^{\mathsf{T}}$

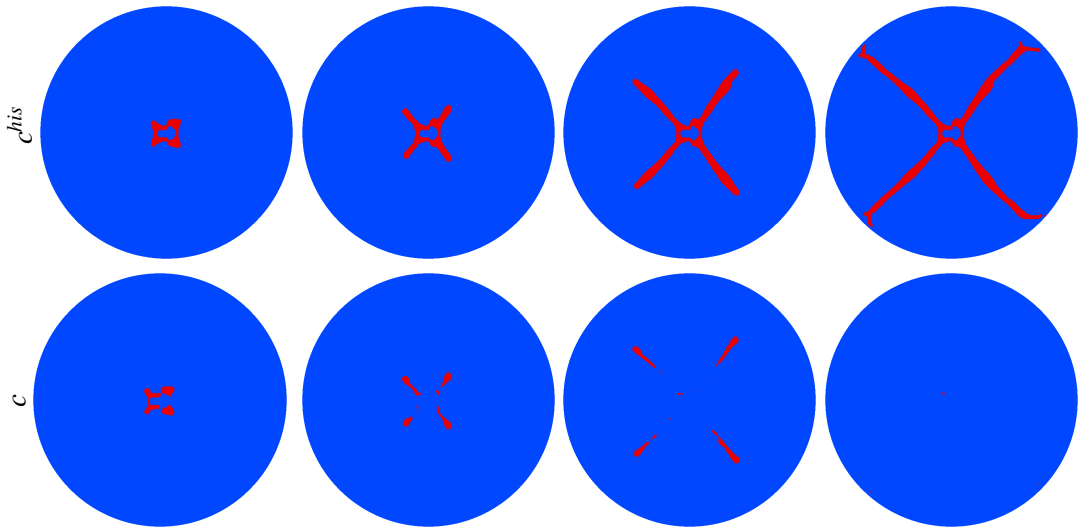


Figure 4.5: The evolution of phase c(bottom row) and its largest value in history c^{his}(top row). c^{his}'s area gradually expands, indicating the propagation of cracks, which is used to extract a consistent crack path. c vanishes when the elastic force is unloaded where fragments are separated. This effectively avoids over-stretching artifacts.

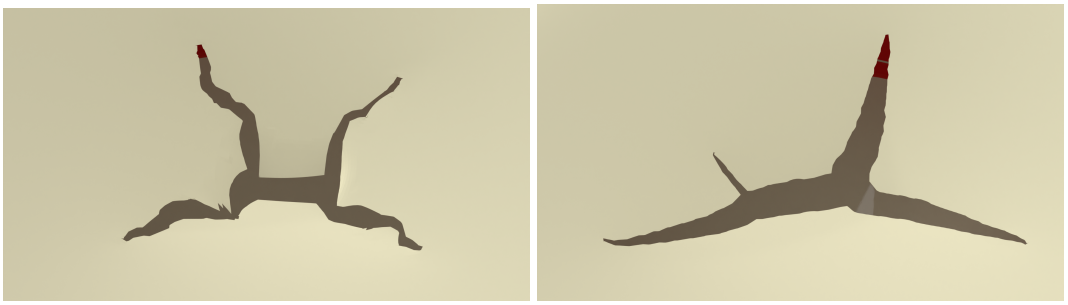
with $\text{diag}(\Sigma) = (\sigma_1, \sigma_2, \sigma_3)$ such that $\sigma_1 \geq \sigma_2 \geq \sigma_3$. We estimate the damage value $\bar{c}$ as in Sec. 2.3.

Damage is updated as $c^{n+1} = \bar{c}$, offering a simple update-rule when compared with conventional fully elasto-dynamic phase-field methods that use $c^{n+1} = \max(\bar{c}, c^n)$ to prevent damaged material healing (see e.g., Wolper et al. (2019); Cervera and Chiumenti (2006); Homel and Herbold (2017)). We have found that the conventional approach suffers from over-stretching artifacts in damaged regions as shown in Fig. 4.6. We avoid such artefacts via our explicit crack representation that provides exact (rather than smeared) locations of strong discontinuity. Moreover, we also store and update the largest value of damage over time $c^{\text{his}} = \max(c^{\text{his}}, c^{n+1})$, which is then used to extract crack geometry as described in Sec. 4.4.2. The weakened (Cauchy) stress is finally evaluated via rankine strain-softening to arrive at

$$\tilde{\mathbf{P}}(\mathbf{P}, c) = \frac{\partial \psi}{\partial \mathbf{F}^{KL}}(\mathbf{F}_q^{KL,Etr}, c) = \det(\mathbf{F}^{KL})\boldsymbol{\sigma}'^{-\mathsf{T}},$$

as the weakened PK1 stress that we use in Eq. 4.8 where $\boldsymbol{\sigma}'$ is the weakened Cauchy stress tensor that satisfies the zero transverse (normal) stress condition.

An example of the evolution of phase is shown in Fig. 4.5. Our use of c^{his} for extracting the crack region simplifies the problem of propagating smeared crack paths in a fully elastodynamic MPM setting. Moreover, our approach, which is based on



(a) Overstretched (b) Ours

Figure 4.6: Extreme over-stretching produced when cutting a thin shell by the original phase field method (Fig. 4.6a) and our proposed method (Fig. 4.6b).

material healing, is a simplified model of temporal singular crack-tip stress, where computed damage merely serves to prescribe the local region within which to extend the crack at the current point in time. Persistence of strong discontinuities in the material is then embodied by the explicit crack surface itself rather than storing permanent damage on the particles, which has the benefit of alleviating over-stretching artefacts that are common with smeared crack representations due to the permanent weakening of internal forces via stress.

4.4.2 Consistent crack extraction

In this section, we propose a consistent crack extraction algorithm based on Voronoi tessellation. We improve the crack extraction algorithm in Sec. 3.3 such that a discrete crack path extends over time. We first implement the algorithm in 2D manifold so that it can extract thin crack lines in the shell's material space. The method overview is shown in Fig. 4.7, which summarises crack extraction from the damaged particles with $c^{\text{his}} = 1$ as follows: (a) Damaged particles composing the domain boundary are extracted and then (b) used to constructed a Voronoi diagram as the Voronoi 'seeds' before (c) extracting the medial axis produced by seeds that lie on opposing sides of the domain boundary as crack path. This results in an inconsistent path when the damaged area thickens while expanding: we thus revise the method such that the crack path gradually extends in a consistent manner (see below, Fig. 4.8). The set notations used in this section can be found in Tab. 4.2.

Computing a consistent crack path: We use the superscript notation "p" and "c" to denote the previous extraction and the current extraction. At each extraction, we first identify fully damaged particle boundaries as $\mathcal{D}$. Due to the evolution of the phase

Set	Description
$\mathcal{D}$	Fully-damaged particle boundary.
$\mathcal{DL}$	Extracted lines using $\mathcal{D}$.
$\mathcal{L}$	Historical lines.
$\mathcal{LS}$	Voronoi seeds that define historical lines.
$\mathcal{NS}$	Voronoi seeds that define new lines.
$\mathcal{TV}$	Terminal vertices of crack lines.
$\mathcal{TS}$	Voronoi seeds that define terminal lines.
$\mathcal{N}$	Neighbours of seeds.

Table 4.2: A summary of the set notation that is used to extract consistent cracks.

field, $\mathcal{D}^p$ (blue points in Fig. 4.8(a)) is not a precise subset of $\mathcal{D}^c$ (brown points in Fig. 4.8(a)) as the smeared crack will thicken while simultaneously propagating forward. As a result, the extracted crack lines $\mathcal{DL}^p$ are not a subset of $\mathcal{DL}^c$. To keep the geometry of the previous extraction, we add $\mathcal{LS}$ to $\mathcal{D}^c$ as the Voronoi tessellation seeds, and term it as $\mathcal{D}^{'}$. After computing its medial axis $D'L$ (dashed line in Fig. 4.8(a)), we only keep (1) new lines whose defining seeds are from $\mathcal{D}^c$ (in orange box in Fig. 4.8(a)); and (2) the subset of historical lines $\mathcal{L}$ (in blue box in Fig. 4.8(a)) . The two lines (1)(2) produced by the above process are usually disconnected and needs to be connected for the crack to correctly propagate.

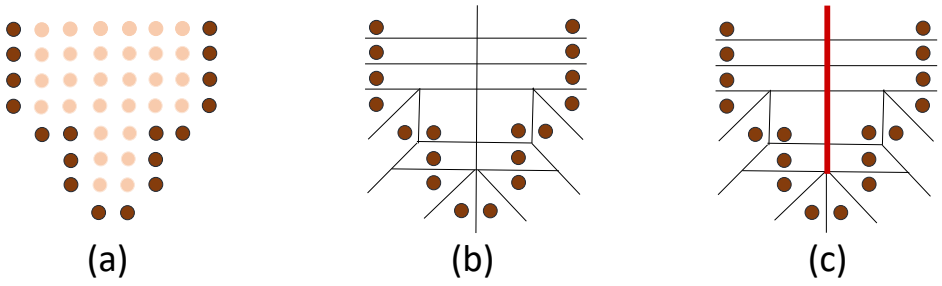


Figure 4.7: Overview of the 2D crack extraction algorithm. (a) Extracting the boundary particles of the damaged region, (b) computing the Voronoi diagram, and (c) extracting the crack that corresponds to the medial axis within the damaged area.

Extend crack tips: In most cases, the crack tends to propagate along the crack tip, where new lines are added to the previous extracted lines. We extract the terminal vertices $\mathcal{TV}$ (black dot in Fig. 4.8(b)) that is connected to the terminal line of $\mathcal{L}$ (purple

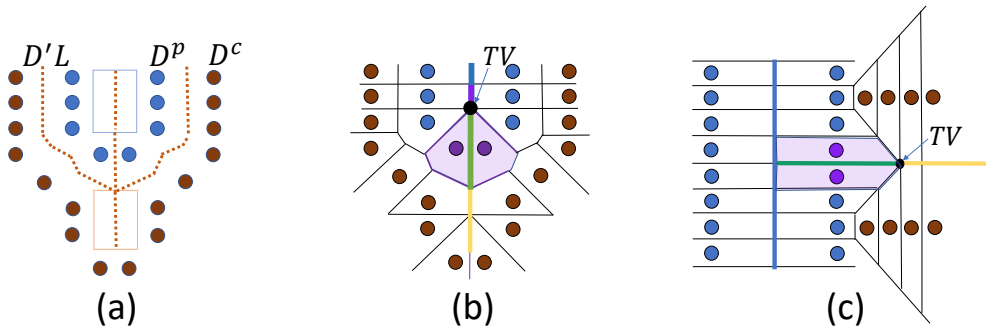


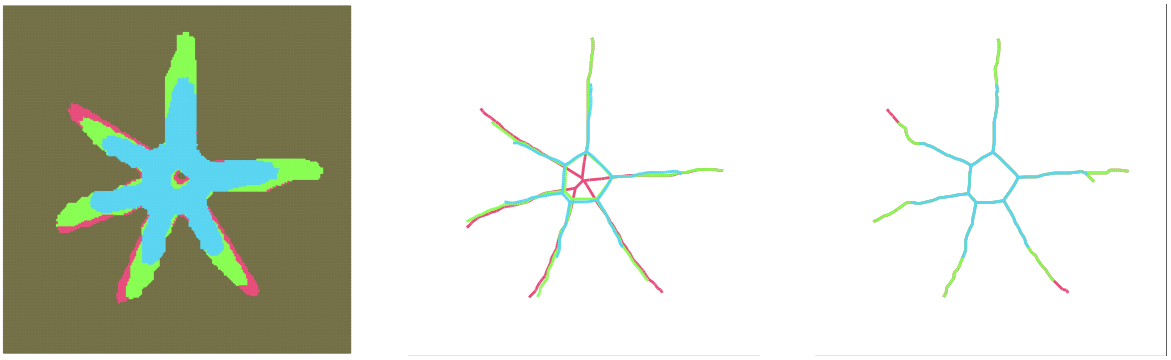
Figure 4.8: (a) Extending the crack in a consistent manner. (b) Connecting the previous crack and the extended crack line. (c) A branching/merging crack connected to the previous crack (see text for the details).

line in Fig. 4.8(b)). Among the cells that are connected to $\mathcal{TV}$, we seek those that are connected to the newly added terminal line (yellow line in Fig. 4.8(b)). This cell's boundary is the bridge connecting previous crack tips to newly extracted lines (green line in Fig. 4.8(b)). This boundary is added into $\mathcal{L}$ and the seeds that produce it are added into $\mathcal{LS}$ for the next extraction.

Extend a branch to the stem: A branching crack may be produced at the stem of previous crack lines as a result of a crack-merge or bifurcation. After extending crack tips (yellow line in Fig. 4.8(c)), we identify new terminal vertices $\mathcal{TV}$ (black dot in Fig. 4.8(c)). Among the cells connected to $\mathcal{TV}$, we check if any of them are connected with a previous line (blue line in Fig. 4.8(c)). If one is discovered, we traverse along the cell boundary that connects the previous line and $\mathcal{TV}$ and add these edges to $\mathcal{L}$. The corresponding seed pairs that form the edges (purple dots in Fig. 4.8(c)) are also found and added to $\mathcal{LS}$.

After extending the crack tips and connecting branches, the extracted crack path is guaranteed to be consistent with the previous extraction without gaps. The crack path can then be used to introduce strong discontinuity to the simulation.

Comparison: In Fig. 4.9 we show an example where we compare the proposed consistent method with that of Sec. 3.3. Simply computing the medial axis using the boundary of the damaged region in each stage (see Fig. 4.9(a)) will result in inconsistent crack paths (see Fig. 4.9(b)). The improved method can produce a crack that propagates overtime in a consistent manner as shown in Fig. 4.9(c).



(a) Phase field. (b) Inconsistent crack in Sec. 3.3. (c) Consistent crack.

Figure 4.9: Comparison of our 2D consistent crack extraction method with that of Sec. 3.3 : (a) A phase field evolution in three stages; cyan, green and pink. (b) Extracted inconsistent crack paths using the method of Sec. 3.3, where the medial axis are computed independently with the damages at each stage. (c) Extracted consistent crack path using our method. Note that part of the cracks in later stages extends those in the earlier stages.

4.5 In-plane stress

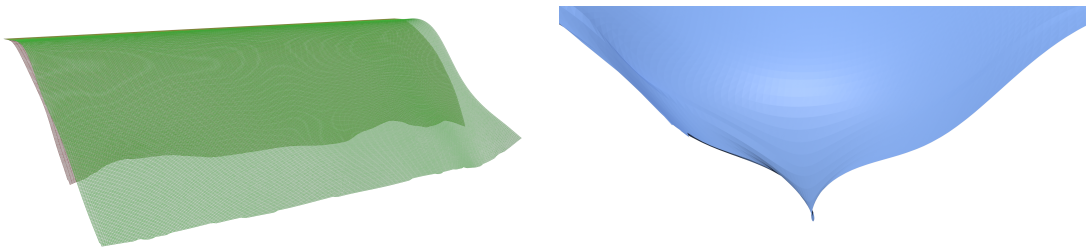
In this section, we describe our approach to stably compute the stress that satisfies the zero transverse stress condition. Satisfying this condition $\boldsymbol{\sigma}\mathbf{g}_3^{KL} = \vec{0}$ means that terms of the stress related to $\mathbf{g}_3^{KL}$ are discarded to keep only those corresponding to the mid-surface tangent plane. Jiang et al. (2017) satisfy this condition using a 2D deformation gradient to then compute a projected 3D stress, while Guo et al. (2018) use the components of the deformation gradient related to the lamina strain to define a plane strain. These methods generally work well but suffer from severe volume-loss or undesirable separation of particles when a crack is introduced as shown in Fig. 4.10. We propose a novel plane-stress formulation satisfying the zero transverse stress condition without artefacts upon fracture.

Using an expression of second-order tensors in curvilinear coordinates, the Cauchy stress can be re-written as

$$\boldsymbol{\sigma} = \sum_{i=1}^{3}\sum_{j=1}^{3} \sigma^{ij}\mathbf{g}_i^{KL} \otimes \mathbf{g}_j^{KL}, \tag{4.11}$$

using the scaled deformed basis vectors $\mathbf{g}_i^{KL}$ to identify the lamina and fiber components of this stress. To solve for the unknown contravariant components σ^{ij}, we construct the following system

$$\mathbf{G}^{KL}\boldsymbol{\varsigma} = \mathrm{vec}(\boldsymbol{\sigma}). \tag{4.12}$$



(a) Ours (green) and Jiang et al. (2017)(red) (b) Guo et al. (2018).

Figure 4.10: Deformation of the shell after being torn: we compare our in-plane stress model with Jiang et al. (2017) and Guo et al. (2018). Jiang et al. (2017) (red) suffers from volume loss while Guo et al. (2018) (blue) experiences material separation near the cracked region. Our in-plane model (green) is robust and stable even in regions near the crack. The original shell's area is 1.0m^2. After cracking, the shell's surface shrinks to 0.668m^2 in Jiang et al. (2017) while our method resolves to 0.994m^2.

where $\mathbf{G}^{KL} \in \mathbb{R}^{9\times 9}$ is the covariant basis matrix, each of whose row $\text{vec}(\mathbf{g}_i^{KL} \otimes \mathbf{g}_j^{KL})^T \in \mathbb{R}^{1\times 9}$ is the tensor-product of two basis vectors in vectorized form with $1 \leq i, j \leq 3$. The term $\boldsymbol{\varsigma} \in \mathbb{R}^{9\times 1}$ is the vector of unknown contravariant components to give, e.g.,

$$\text{vec}(\mathbf{g}_1^{KL} \otimes \mathbf{g}_1^{KL})^T \cdot \boldsymbol{\sigma}^{11} = \sigma_{11} \tag{4.13}$$

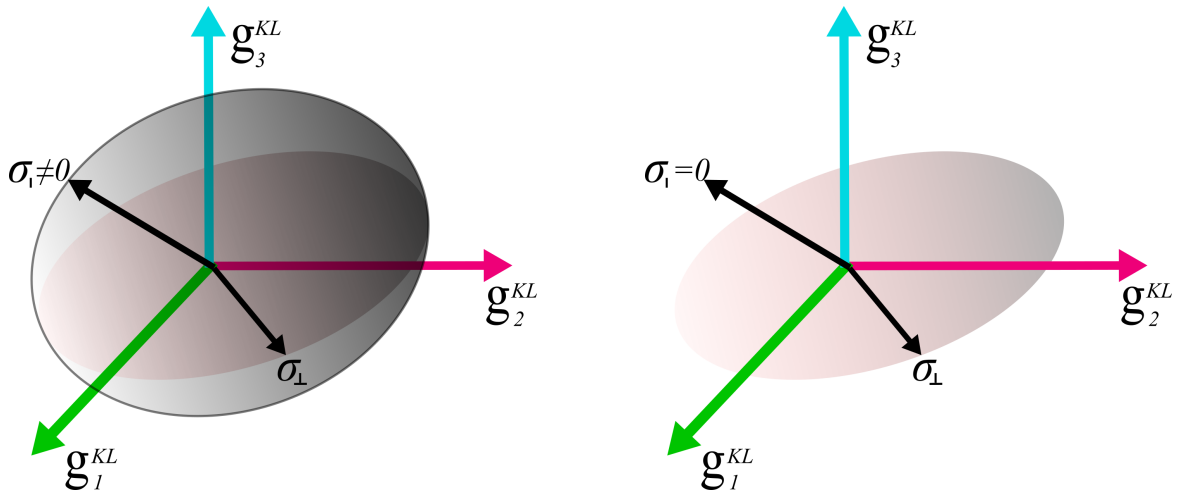
as the equation of the first row. Thus, solving Eq. 4.12 for $\boldsymbol{\varsigma}$ gives all components σ^{ij} from which we compute the Cauchy stress satisfying the zero transverse stress condition by

$$\boldsymbol{\sigma}_\perp = \sum_{\alpha=1}^{2}\sum_{\beta=1}^{2} \sigma^{\alpha\beta}\mathbf{g}_\alpha^{KL} \otimes \mathbf{g}_\beta^{KL} \in \mathbb{R}^{3\times 3}, \tag{4.14}$$

which we use as the Kirchhoff-Love bending stress. In contrast with Jiang et al. (2017), our approach operates directly on the 3D stress tensor, which is a simpler and more robust method: Jiang et al. (2017) first construct a 2D (planer) deformation gradient $\mathbf{F} \in \mathbb{R}^{2\times 2}$ of the mid-surface followed by a calculation the corresponding 2D stress that is then extrapolated to 3D. The use of a 2D $\mathbf{F}$ has the potential for underestimating the volume ratio $J = \det(\mathbf{F})$ of the true deformation gradient $\mathbf{F} \in \mathbb{R}^{3\times 3}$.

4.6 Resolving discontinuities in material space

In this section, we describe our MLS interpolation of field quantities, which we use when a cut is made with the thin shell. The method will provide a scheme to interpolate the quantities of the type-II and type-IV particles near the crack(s). Such an interpolation is done in parameter space ω.



(a) Cauchy stress tensor. (b) Plane-stress tensor after removing normal component.

Figure 4.11: The plane-stress tensor can be computed by removing the normal component from the original Cauchy stress tensor. (a) The original Cauchy stress tensor, where stress in any direction, $\boldsymbol{\sigma}_{|}$, can be decomposed using three covariant basis vectors $\mathbf{g}_1^{KL}$, $\mathbf{g}_2^{KL}$ and $\mathbf{g}_3^{KL}$. The grey shaded area represents the 3D stress, and the red shaded area represents the stress component in the plane (lamina) defined by $\mathbf{g}_1^{KL}$ and $\mathbf{g}_2^{KL}$. (b) The stress component in the fiber direction is removed. Only plane-stress component is kept. Any stress which is not in the plane is 0 now, i.e., $\boldsymbol{\sigma}_{|} = 0$.

The NURBS shape functions are designed to maintain smooth continuity of the interpolated field quantities across patches, which can limit our ability to accurately represent discontinuities due to cracks. We can overcome this limitation by utilizing the MLS approximation of the field quantities that are stored at particles located near the crack, as demonstrated in Fig. 4.12. A Type-II and Type-IV particle are together considered separated by the crack if their connecting line segment intersects this crack in parameter space. Information is then transferred between a Type-II and Type-IV only if they reside on the same side of the crack. Particles far from the crack shall use standard NURBS shape functions.

Our MLS approximation will estimate field variables (like deformed covariant basis vectors) at a quadrature particle $q(\xi) \in \Gamma^{(\mathrm{IV})}$ from the $\Gamma^{(\mathrm{II})}$ control points in its support domain by

$$u^h(\xi) = \sum_{i=1}^{|C^{(q)}|} \mathcal{W}_i^{\mathrm{MLS}}(\xi) u_i, \tag{4.15}$$

where u^h is the approximation of the unknown field quantity at $\xi \in \mathbb{R}^2$ using known values u_i at the control point $C_i^{(q)}$ that is in the support domain centered at $q(\xi)$. We also have $\mathcal{W}_i^{\mathrm{MLS}}$ as the shape function associated with the i'th control point, which is

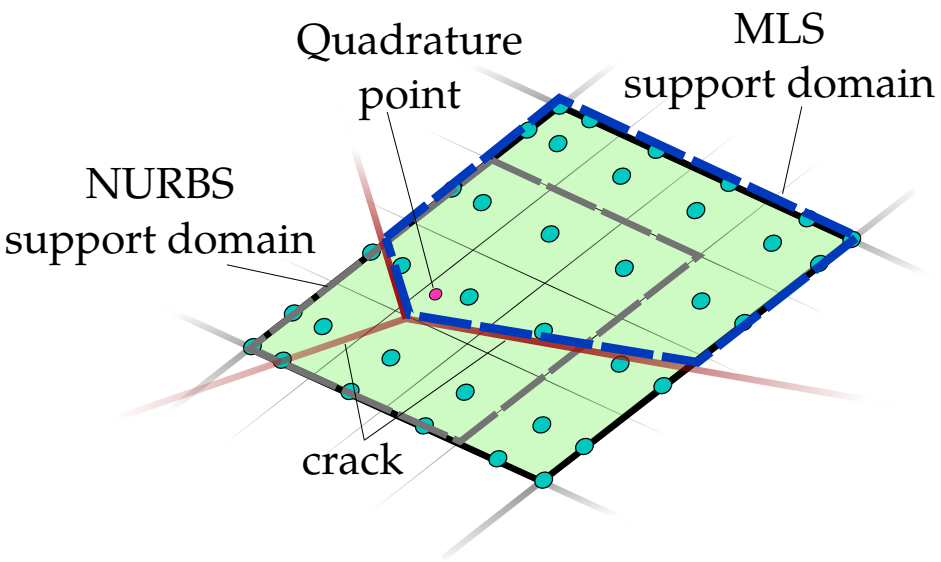


Figure 4.12: Approximating field quantities of a quadrature point near the crack: The red lines are extracted crack paths, and the turquoise blue points depict the control points (Type-II particles). A single quadrature point is shown here in pink, which is affected by the crack. Without the crack, this point would use a rectangular support radius (grey). The purple dashed line represents the (typically larger) custom support radius of the quadrature point that is used with the crack for MLS interpolation.

given by

$$\mathcal{W}_i^{\mathrm{MLS}}(\xi) = \mathbf{p}^{\intercal}(\xi)(\mathbf{A}^{-1}(\xi)\mathbf{B}(\xi))_i, \tag{4.16}$$

where $\mathbf{p}(\xi) = [1, \xi_1, \xi_2, \xi_1^2, \xi_1\xi_2, \xi_2^2]^{\intercal} \in \mathbb{R}^6$ is the polynomial basis vector; and $\mathbf{A} \in \mathbb{R}^{6\times 6}$ is given by

$$\mathbf{A}(\xi) = \sum_{k=1}^{|C^{(q)}|} W(\xi - \xi^{(k)})\mathbf{p}(\xi^{(k)})\mathbf{p}^{\intercal}(\xi^{(k)}), \tag{4.17}$$

where $\xi^{(k)} \in \mathbb{R}^2$ are the parameter coordinates of the kth control point, and $W(\xi - \xi^{(k)})$ is the weight function of MLS

$$W(\xi - \xi^{(k)}) = \begin{cases} 2/3 - 4r^2 + 4r^3 & r \leq 0.5 \\ 4/3 - 4r + 4r^3 - 4/3r^3 & 0.5 \leq r \leq 1.0 \\ 0 & r > 1.0 \end{cases}, \tag{4.18}$$

which is cubic in our case with $r = \|\xi - \xi^{(k)}\|/d_X^{(k)}$ where $d_X^{(k)}$ is the support domain size of the kth control point. The matrix $\mathbf{B} \in \mathbb{R}^{\mathrm{m}\times|\mathrm{C}^{(\mathrm{q})}|}$ is defined as

$$\mathbf{B} = \left[\begin{array}{ccc} W(\xi - \xi^{(1)})\mathbf{p}(\xi^{(1)}) & \ldots & W(\xi - \xi^{(|C(q)|)})\mathbf{p}(\xi^{(|C(q)|)}) \end{array} \right]. \tag{4.19}$$

We adopt a rectangular support domain, which means that the total weighting is calculated as a multiplication of weight along each direction of the parameter space $W(\xi_1 - \xi_1^{(k)})W(\xi_2 - \xi_2^{(k)})$.

We determine $d_X^{(k)}$ based on the type of control-point. For a NURBS control-point, this is equivalent to setting $d_X^{(k)} = 1.5l$, where l is the cell-size of a NURBS patch. In the case of an MLS control-point, we dynamically select $d_X^{(k)}$ depending on the complexity of the crack: Starting from $d_X^{(k)} = 1.5l$, we count the number of neighbouring control-points (on the same side of the explicit crack described in Sec. 4.4.2) to then expand the list farther away from the crack if the current control-point has less than nine neighbours[1]. Throughout our experiments, we found this to require a support domains size of no more than approximately $d_X^{(k)} = 4.5l$, which may also be set as a fixed constant.

First and second order derivatives of the MLS shape function in Eq. 4.15 are also required to compute the deformation gradient of quadrature points. The first order derivative of $\mathcal{W}_i^{\text{MLS}}$ wrt ξ_α is:

$$\mathcal{W}_i^{\text{MLS}}{}_{,\alpha} = \mathbf{p}_{,\alpha}^T(\mathbf{A}^{-1}\mathbf{B})_i + \mathbf{p}^T(\mathbf{A}_{,\alpha}^{-1}\mathbf{B})_i + \mathbf{p}^T(\xi)(\mathbf{A}^{-1}\mathbf{B}_{,\alpha})_i, \tag{4.20}$$

where $\mathbf{A}_{,\alpha}^{-1} = -\mathbf{A}^{-1}\mathbf{A}_{,\alpha}\mathbf{A}^{-1}$. The second order derivative is calculated by repeated application of the product rule:

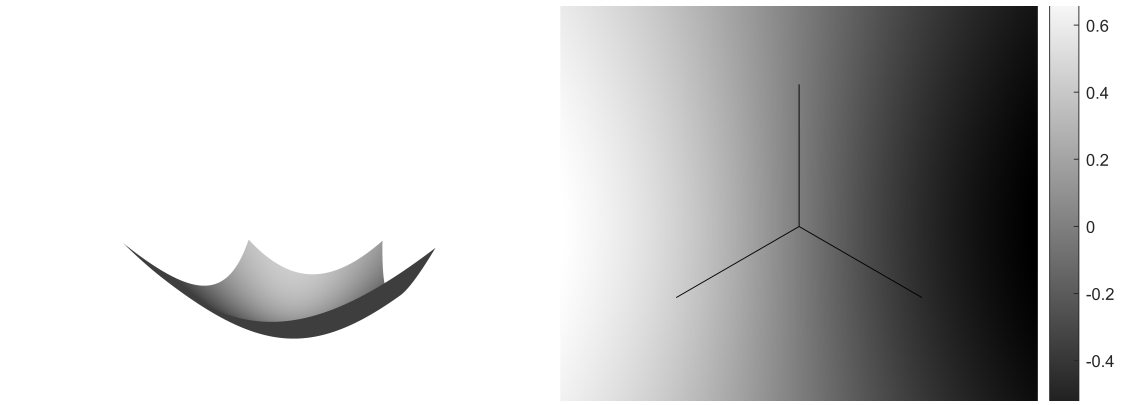
$$\begin{aligned}\mathcal{W}_i^{\text{MLS}}{}_{,\alpha\beta} =& \mathbf{p}_{,\alpha\beta}^T(\mathbf{A}^{-1}\mathbf{B})_i + \mathbf{p}_{,\alpha}^T(\mathbf{A}_{,\beta}^{-1}\mathbf{B})_i + \mathbf{p}_{,\alpha}^T(\mathbf{A}^{-1}\mathbf{B}_{,\beta})_i + \\ & \mathbf{p}_{,\beta}^T(\mathbf{A}_{,\alpha}^{-1}\mathbf{B})_i + \mathbf{p}^T(\mathbf{A}_{,\alpha\beta}^{-1}\mathbf{B})_i + \mathbf{p}^T(\mathbf{A}_{,\alpha}^{-1}\mathbf{B}_{,\beta})_i + \\ & \mathbf{p}_{,\beta}^T(\mathbf{A}^{-1}\mathbf{B}_{,\alpha})_i + \mathbf{p}^T(\mathbf{A}_{,\beta}^{-1}\mathbf{B}_\alpha)_i + \mathbf{p}^T(\mathbf{A}^{-1}\mathbf{B}_{,\alpha\beta})_i\end{aligned} \tag{4.21}$$

where $\mathbf{A}_{,\alpha\beta}^{-1} = \mathbf{A}^{-1}\mathbf{A}_{,\beta}\mathbf{A}^{-1}\mathbf{A}_{,\alpha}\mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{A}_{,\alpha\beta}\mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{A}_{,\alpha}\mathbf{A}^{-1}\mathbf{A}_{,\beta}\mathbf{A}^{-1}$.

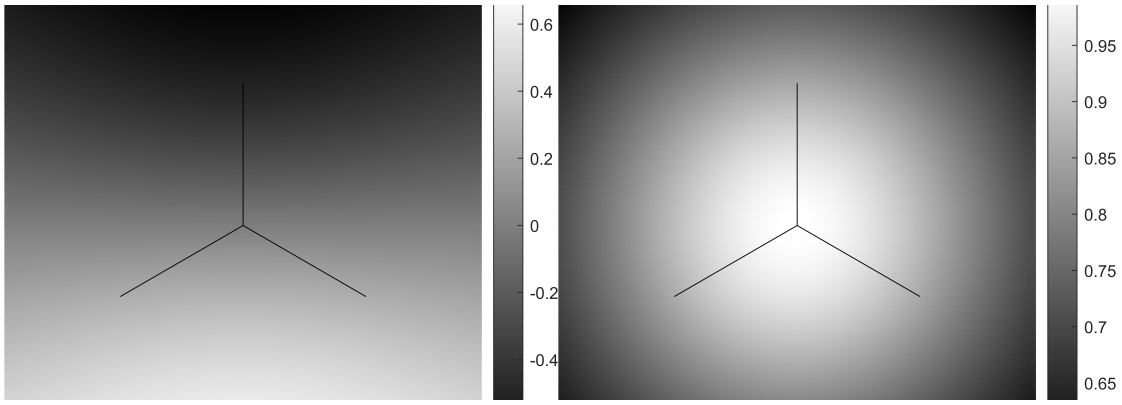
4.7 Resolving discontinuities in world space

Conventional MPM builds a single background grid on which derivatives are computed to alleviate the need for a Lagrangian mesh, where contacts are handled automatically. However, using a single grid tends to cause cohesion artifacts when simulating the contacts between multiple objects (e.g.,from fracture) since each grid node will receive information from particles belonging to different fragments. For shells, this can result in cracked fragments failing to separate as the averaged velocity of separating nodes will cancel out each other, or gluing artefacts between shells and external objects

[1]Nine neighbours because we are using a quadratic polynomial basis vector. Otherwise sixteen for cubic Li et al. (2020).



(a) The mesh. (b) x-component.



(c) y-component. (d) z-component.

Figure 4.13: Our MLS formulation can achieve a smooth transition of the normal vector across the regions where we use NURBS shape function region and those where we apply our MLS shape function, which is in the vicinity of the crack(black lines).

(defined by Type-I particles $\in \Gamma^{(\mathrm{I})}$). The multi-velocity field method we described in Here we describe a solution to resolve the cohesion artifacts and successfully partition material when there is a crack.

4.7.1 Grid-based partitioning

To apply material space fractures in world space, we create multiple ”phantom” copies for each grid node **i** in the vicinity of a crack, based on the number of cracks within its influence radius. A copy represents an instance of the node to which field quantities (mass, velocity and elastic force) are transferred. We determine if a grid node is in the vicinity of the crack by checking whether any particle in its influence radius is taking part in the MLS shape function used for constructing the crack geometry (*cf.* Sec. 4.6). Then, the number of disconnected segments in the vicinity of the grid node is counted

based on Sun et al. (2021) and a phantom node is produced per count. With this solution, we can simulate continuum shells with multiple branching cracks while resolving contacts with other (e.g.,deformable) objects as shown in Sec. 4.10.

4.7.2 Guaranteeing separation by contact forces

Here we present a contact algorithm, using phantom nodes for cohesion-free resolution of inter-object/fragment contact at grid nodes in the vicinity of a crack. Given the surface normal vectors at two phantom copies a and b of grid node $\mathbf{i}$, we define the contact normal direction at this node by

$$\mathbf{n}_\mathbf{i} = \begin{cases} (\mathbf{n}_\mathbf{i}^a - \mathbf{n}_\mathbf{i}^b)/\left\|\mathbf{n}_\mathbf{i}^a - \mathbf{n}_\mathbf{i}^b\right\| & a \text{ is solid, and } b \text{ is solid} \\ \mathbf{n}_\mathbf{i}^a/\left\|\mathbf{n}_\mathbf{i}^a\right\| & a \text{ is solid, and } b \text{ is shell} \\ -\mathbf{n}_\mathbf{i}^b/\left\|\mathbf{n}_\mathbf{i}^b\right\| & a \text{ is shell, and } b \text{ is solid} \\ (\mathbf{x}_{\text{com}}^b - \mathbf{x}_{\text{com}}^a)/\left\|\mathbf{x}_{\text{com}}^b - \mathbf{x}_{\text{com}}^a\right\| & a \text{ is shell, and } b \text{ is shell} \end{cases} \tag{4.22}$$

where $\mathbf{n}_\mathbf{i}^a$, $\mathbf{n}_\mathbf{i}^b$, and $\mathbf{x}_{\text{com}}^a$, $\mathbf{x}_{\text{com}}^b$ are the normals and centres of mass (COM) for the segments that include a and b, respectively. The normal of a solid is determined by either its precomputed outward normal direction for rigid bodies or by the normalized density gradient (*cf.* Eq (11) in Homel and Herbold (2017)). The COM of a shell is computed as $\mathbf{x}_{\text{com}} = \frac{\sum_p m_p \mathbf{x}_p}{\sum_p m_p}$ where m_p and $\mathbf{x}_p$ are the mass and position of a particle p within the support radius of the node. The approach resembles the virtual node algorithm Molino et al. (2005b) and non-manifold level sets Mitchell et al. (2015) but it is distinguished by the fact that our (MPM) grid is initialized and destroyed every timestep. Thus we have no requirement to maintain additional information about the nodes besides their duplication based on the dynamic partitioning material particles.

4.7.2.0.1 Separation contact force We determine the adhesive contact condition at grid node $\mathbf{i}$ between the two fragments of a and b based on their relative velocity in the normal direction $v_\mathbf{i}^{a,b} = (\overline{\mathbf{v}}_\mathbf{i}^a - \overline{\mathbf{v}}_\mathbf{i}^b)^T \mathbf{n}_\mathbf{i}$, where $\overline{\mathbf{v}}_\mathbf{i}^a$ and $\overline{\mathbf{v}}_\mathbf{i}^b$ are the respective trial velocities. We assume the two are in separation when $v_\mathbf{i}^{a,b} \leq 0$ and in contact/penetration when $v_\mathbf{i}^{a,b} > 0$. The trial velocities $\overline{\mathbf{v}}_\mathbf{i}^a$ and $\overline{\mathbf{v}}_\mathbf{i}^b$ are updated in the latter case using the adhesive force, whose magnitude in the normal and tangent directions are computed by:

$$f_{\text{con},N}^a = -\frac{1}{\Delta t}\frac{m^a m^b}{m^a + m^b}[(\overline{\mathbf{v}}_\mathbf{i}^b - \overline{\mathbf{v}}_\mathbf{i}^a)^\intercal \mathbf{n}_\mathbf{i}] \tag{4.23}$$

$$f_{\text{con},T}^a = -\frac{1}{\Delta t}\frac{m^a m^b}{m^a + m^b}[(\overline{\mathbf{v}}_\mathbf{i}^b - \overline{\mathbf{v}}_\mathbf{i}^a)^\intercal \boldsymbol{\tau}_\mathbf{i}] \tag{4.24}$$

where Δt as the timestep size, and $\boldsymbol{\tau}_\mathbf{i} = (\overline{\mathbf{v}}_\mathbf{i}^b - \overline{\mathbf{v}}_\mathbf{i}^a)/\|\overline{\mathbf{v}}_\mathbf{i}^b - \overline{\mathbf{v}}_\mathbf{i}^a\| - \mathbf{n}_\mathbf{i}$ is the contact tangential vector. The above adhesive force eliminates the component of relative velocity that will cause penetration.

4.7.2.0.2 Sliding friction force When the tangent contact force computed by Eq. 4.24 is larger than the maximum static friction force, i.e., $\left|f_{\text{con},T}^a\right| > \mu \left|f_{\text{con},N}^a\right|$ where μ is the friction constant, the tangent force is replaced by the sliding force:

$$f_{con,T}^a = -\mu \left|f_{con,N}^a\right| \operatorname{sgn}[(\overline{\mathbf{v}}_\mathbf{i}^b - \overline{\mathbf{v}}_\mathbf{i}^a)^T \boldsymbol{\tau}_\mathbf{i}]. \tag{4.25}$$

4.7.2.0.3 Total contact force The total contact force applied to the phantom copy a of grid node $\mathbf{i}$ is thus

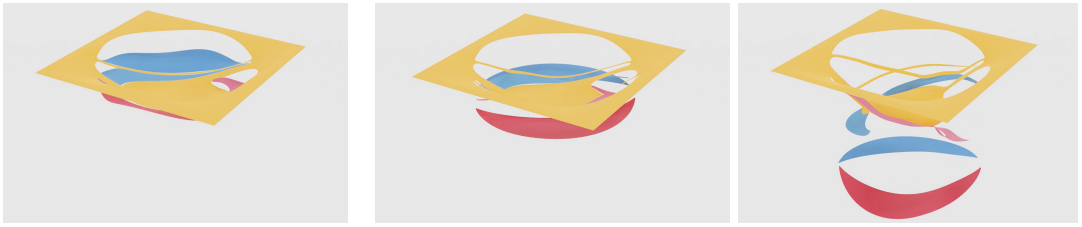
$$\mathbf{f}_{\mathbf{i},\text{con}}^a = -(f_{\text{con},N}^a \mathbf{n}_\mathbf{i} + f_{\text{con},T}^a \boldsymbol{\tau}_\mathbf{i}) \tag{4.26}$$

from which we compute $\mathbf{f}_{\mathbf{i},\text{con}}^b = -\mathbf{f}_{\mathbf{i},\text{con}}^a$ as the corresponding total force applied to the phantom copy b.

We enhance the method outlined in Eq. 4.23-(4.26) to manage three or more segments as done by Sun et al. (2021). More specifically, we iterate the momentum update process by computing the contact force between the two fragments with the largest relative velocity and applying the force to the nodes. This is repeated until all relative velocity residuals are lower than a threshold $\varepsilon = 10^{-4}$. A disadvantage of this method is that field quantities near the crack-tip are underestimated, because particles of the shell surrounding this explicit crack-tip are lumped into a single object at node $\mathbf{i}$. This is however not a significant issue in practice as only the particles near the crack-tip are affected. The estimation is accurate beyond this crack-tip region, where momentum conservation and non-penetration conditions are satisfied. We also refer readers to Sun et al. (2021) for further discussions.

Comparing with Homel and Herbold (2017), we compute the contact force between multiple objects iteratively, whereas their method will only consider two objects. Furthermore, their method of mass gradients for approximating contact normals does not generalize to thin geometries.

We show an example of applying our method for animating a pre-cut thin-shell in Fig. 4.14. Conventional MPM does not produce contact forces between adjacent particles and thus they suffer from cohesion artifacts (see Fig. 4.14(a)). Sun et al. (2021) compute the normal vectors of the objects and use them to apply contact force to the phantom nodes, but they do not handle co-dimensional objects. Simply using



(a) Conventional MPM method. (b) Sun et al. (2021). (c) Ours.

Figure 4.14: Our phantom method can effectively simulate the separation of co-dimensional objects. A SIGGRAPH logo is cut off from a piece of cloth and pulled by gravity for 2 seconds: (a) Conventional MPM method suffers from cohesion artifact. (b) Sun et al. (2021) adds phantom nodes near the cracked region which alleviates the cohesion artifact. (c) Our method correctly computes the outward normals and separates the thin shell fragments.

the normal vector of the surface results in cohesion as shown in Fig. 4.14(b). Our approach can compute the normal vectors even for the cut area of thin shells, resulting in the fastest separation as shown in Fig. 4.14(c).

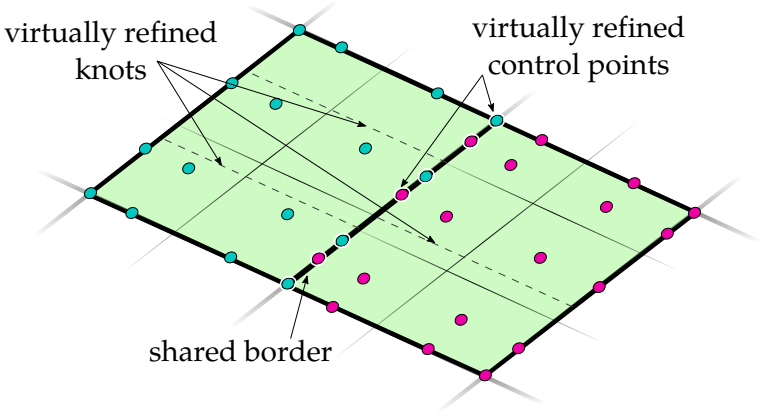


Figure 4.15: Patch coupling: We propose an extension to the virtual uncommon-knot insertion method of Coox et al. (2017) to ensure C^1 coupling between patches. The image shows two non-conforming patches (generally defining a complex surface) that have mismatching resolution to require C^1 and C^2 coupling for ensuring C^1 continuous velocity transfers with MPM grids and C^2 continuous tangent directions.

4.8 Coupling NURBS patches

Simulating objects defined by complex surface geometry that are discretized as multiple NURBS patches will require C^1 continuity (or H^2-regularity) at the borders between these patches for calculating deformation gradients of quadrature particles near

these borders. A coupling of conforming patches (those that have matching resolution in knot placement) can be done easily but non-conformity requires special treatment. Coox et al. (2017) recently propose a virtual uncommon-knot insertion method at the border between non-conforming patches. The method works well in isogeometric analysis with finite elements but induces discontinuous velocity fields when applied to MPM, leading to exacerbated stress at quadrature particles. We present a modification to the virtual uncommon-knot insertion method in this section, which we use to mitigate discontinuous velocities estimated at quadrature particles near borders between non-comforming patches.

4.8.1 C^0 continuity

Let $\mathbf{k}_\eta$ and $\mathbf{k}_\lambda$ be the knot-vectors, where the entries represent the knots along the border that is shared between two non-conforming patches with higher and lower resolution, respectively, as shown in Fig. 4.15. To ensure C^0 continuity between these two 1D *curves*, we will require a way to relate control points and knots of the two patches for smooth interpolation of velocity across the respective patches. To do so, the knot vectors are virtually refined until they are identical to give $\mathbf{k}'_\eta = \mathbf{k}'_\lambda = \mathbf{k}_\eta \cup \mathbf{k}_\lambda$ as the refined knot vectors. This means that the knots to be inserted in either vector will be

$$\tilde{\mathbf{k}}_* = \mathbf{k}'_* \setminus \mathbf{k}_* = \{\mathbf{k}'_{*,1}, \ldots, \mathbf{k}'_{*,k_*}\} \in \mathbb{R}^{\mathrm{k}_*} \tag{4.27}$$

where k_* is the number of knots inserted, and $*$ is used as stand-in for 'η' or 'λ'. A knot vector will thus have original dimensions $\mathbb{R}^{\mathrm{k}_* \times \mathrm{n}_* \times \mathrm{o}_*}$, where n_* is the number of control points and o_* is the polynomial order of the NURBS patch.

Insertion of knots also implies insertion of control points with matching resolution (like the form at the patch boundary shown in Fig. 4.4). We therefore also require a way of relating the original control point positions $\mathbf{c}_* \in \mathbb{R}^{3\mathrm{n}_*}$ that are specified according to the knots $\in \mathbf{k}_*$ and the new set positions $\mathbf{c}'_* \in \mathbb{R}^{3(\mathrm{n}_* + \mathrm{k}_*)}$ according to $\mathbf{k}'_*$.

A relationship between control points $\mathbf{c}_*$ and $\mathbf{c}'_*$ can be established as

$$\mathbf{c}'_* = \mathbf{U}_* \mathbf{c}_*, \tag{4.28}$$

where $\mathbf{U}_* \in \mathbb{R}^{(\mathrm{n}_* \mathrm{k}_*) \times \mathrm{n}_*}$ is determined solely from the original $\mathbf{k}_*$ and final knot vector $\mathbf{k}'_*$ (see Appendix D). The two NURBS curves (running the length of the border) express the same physical curve after knot insertion, where the virtually-refined new control points will be the same since the knot vectors $\mathbf{k}'_\eta = \mathbf{k}'_\lambda$ are identical to give $\mathbf{U}_\eta \mathbf{c}_\eta = \mathbf{U}_\lambda \mathbf{c}_\lambda$.

The positions of the original control points along the higher resolution curve are coupled to the lower resolution curve using

$$\mathbf{c}_\eta = \mathbf{U}_\eta^+ \mathbf{U}_\lambda \mathbf{c}_\lambda, \tag{4.29}$$

where $\mathbf{U}_\lambda^+$ is the Moore-Penrose pseudo-inverse of $\mathbf{U}_\lambda$. Such a linear system states that control points of the higher resolution curve can be determined by those of the lower resolution, which we can use during the MPM grid-to-particle transfer phase as follows: We first tag the control points of the high resolution curve; Then during the grid-to-particle transfer phase of MPM, we transfer velocity only to the control points of the lower resolution curve with positions $\in \mathbf{c}_\lambda$ after-which we advect them followed by updating the positions $\mathbf{c}_\eta$ using Eq. 4.29.

4.8.1.1 Continous velocity fields

The method outlined works well to update positions but violates the continuous assumption of the velocity field in MPM because positions are updated via Eq. 4.29 rather than directly from the MPM grid which provides a continuous velocity field. The effect is that stress values at quadrature points near the (higher resolution) patch border become excessively large due to a discontinuity between the position of the control points defining $\mathbf{c}_\eta$ and rest in the patch. We resolve this by updating the original control points that define $\mathbf{c}_\eta$ with a momentum-correction velocity, which is given by

$$(\mathbf{U}_\eta^+ \mathbf{U}_\lambda \mathbf{c}_\lambda)^{n+1} - \bar{\mathbf{c}}_\eta^{n+1} = \Delta t \check{\mathbf{v}}_\eta^{n+1}. \tag{4.30}$$

The term $\bar{\mathbf{c}}_\eta^{n+1} = \mathbf{c}_\eta^n + \Delta t \mathbf{v}_\eta^{n+1}$ is the newly advected positions after the grid-to-particle transfer of the current timestep, which we must be corrected to ensure continuity. To do so, we transfer $\check{\mathbf{v}}_\eta^{n+1}$ back to the grid once-more

$$\check{\mathbf{v}}_\mathbf{i}^{n+1} = \sum_p m_p w_{\mathbf{i}p}^n \check{\mathbf{v}}_{\eta,p}^{n+1} / m_\mathbf{i}^n \tag{4.31}$$

to calculate a correction velocity, using only the high resolution control points with positions $\in \mathbf{c}_\eta$. Although only these control points transfer $\bar{\mathbf{c}}_\eta^{n+1}$ to the grid once more, their neighbouring control points of the same (higher resolution) patch will also receive a corrected velocity from the grid by

$$\mathbf{v}_p^{n+1} = \mathbf{v}_p^{n+1} + \sum_\mathbf{i} w_{\mathbf{i}p} \check{\mathbf{v}}_\mathbf{i}^{n+1}, \tag{4.32}$$

which is used for their (second) advection.

4.8.2 C^1 continuity

Modelling the behaviour of objects that are characterised by *smooth geometry* requires that our NURBS surfaces (not shape functions) are C^1 continuous. We can easily extend the above derivation for C^0 coupling to C^1 (or even higher-order) coupling. The key factor is the number of control points used for the coupling, where only a single row/curve of control points where needed for C^0 coupling. For a general C^e coupling, $e+1$ rows/curves of control points are required to ensure high-order derivatives are continuous across the patch border. For example, two rows of control points near the border determine the continuous first-order derivative between two NURBS patches, with the requirement that control points near the border are collinear (tangent directions are the same) with those on the other side/patch.

Let $\mathbf{c}_{*1}$ be the control points in the interface, and let $\mathbf{c}_{*2}$ be the control points next to $\mathbf{c}_{*1}$ in the same patch. To enforce C^1 continuity, the C^0 continuity condition must first hold, which is $\mathbf{c}_{\eta 1}^{n+1} = \mathbf{U}_\eta^+ \mathbf{U}_\lambda \mathbf{c}_{\lambda 1}^{n+1}$. The colinearity condition must also hold, which is given by

$$\mathbf{D}_{\eta\lambda}\mathbf{U}_\lambda(\mathbf{c}_{\lambda 1}^{n+1} - \mathbf{c}_{\lambda 2}^{n+1}) = \mathbf{U}_\eta(\mathbf{c}_{\eta 1}^{n+1} - \mathbf{c}_{\eta 2}^{n+1}) \tag{4.33}$$

where $\mathbf{D}_{\eta\lambda}$ is a constant diagonal matrix that contains mesh-dependent constraints Coox et al. (2017). The control points with positions $\mathbf{c}_{\eta 2}$ are determined by

$$\mathbf{c}_{\eta 2}^{n+1} = \mathbf{U}_\eta^+ \mathbf{U}_\lambda \mathbf{c}_{\lambda 1}^{n+1} - \mathbf{U}_\eta^+ \mathbf{D}_{\eta\lambda}\mathbf{U}_\lambda(\mathbf{c}_{\lambda 1}^{n+1} - \mathbf{c}_{\lambda 2}^{n+1}) \tag{4.34}$$

from which the unknown velocity term of the C^1 coupling can be calculated as in Eq. 4.30 to Eq. 4.31, and the adjacent control points' position can be updated using Eq. 4.32.

4.9 Get physical rendering mesh

The NURBS surface is defined in both material space (2D) and world space (3D), where the former stores control points' coordinate in the patch, and the latter stores control points' position in the real world. We call the mesh in the world space "physical mesh". We use the physical mesh for rendering as it represents the actual middle surface of the shell. The physical mesh can be easily computed as a collection of polygons defined by the NURBS interpolation function without cracks. However, when a crack does pass through the NURBS patch, the physical mesh has to be updated ac-

cordingly because the mesh connectivity is changed, e.g., new vertices and edges are inserted.

In this section, we outline how we get the physical rendering mesh with cracks passing through it. An illustrative overview is also provided in Fig. 4.16.

4.9.1 Overview

Given NURBS patches, cracks, and positions of control points, we calculate the physical mesh of each patch separately for the sake of convenience and parallel performance. In the material space, we split the patch into **R**x**S** quad elements. Note that the resolution is independent of the resolution of the NURBS patch (Sec. 4.3.1). Therefore, we can get a higher-resolution rendering mesh from a relatively low-resolution simulation mesh. The patch thus has (**R**+1)x(**S**+1) vertices, i.e., yellow hollow points in Fig. 4.16. We then calculate the intersection points, i.e., purple hollow points, of the crack with the quad elements' edges. For each quad element who are interfacing with the crack, its edges are split into several line segments by the intersection points. Meanwhile, these points also crop the crack lines and leave partial crack segments inside the element. We then duplicate all crack line segments in the quad element as in Fig. 4.16.c. The crack line segments separate the quad element into several smaller elements that we can find iteratively. Then we calculate the world-space positions of all points. This step maps the 2D elements in the material-space into world-space polygons. They are the faces of the rendering mesh. We assemble all polygons together to get the final physical rendering mesh.

4.9.2 Find elements separated by cracks

We find elements separated by cracks iteratively using the quad element edge segments and crack line segments. As shown in Fig. 4.16.c, the quad element's edges, "Y3Y2", "Y2Y1" and "Y1Y0", are split by crack lines into eight segments, e.g., "Y3P4", "P4P3" etc. Note that we index these segments in an anti-clockwise direction. In this quad element, there are also four crack line segments. For each segment, we generate two lines with reversed directions, e.g., "P0G0" and "G0P0". All lines are shown with an arrow indicating the direction in Fig. 4.16.c. These lines will form the subelement's edges and they are stored in a set $\mathcal{SL}$.

We randomly select one line from $\mathcal{SL}$ as the starting edge, e.g., "AmBn", where (A,B) $\in$ (G,Y,P) and (m,n) $\in$ (0,1,2...). Then we search for candidate lines that start

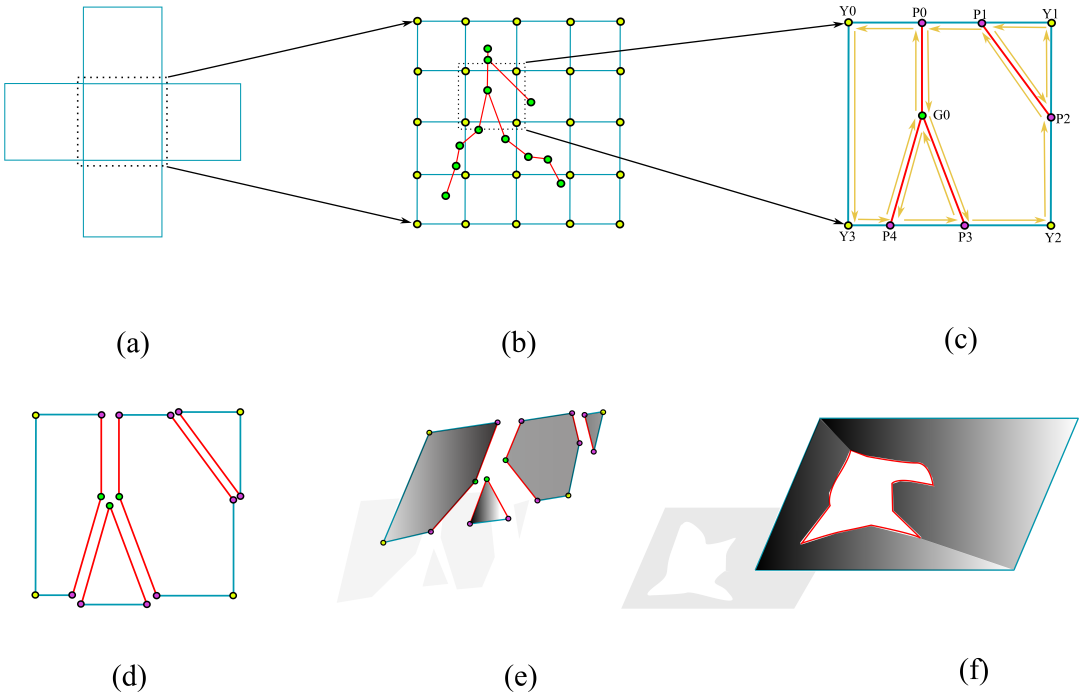


Figure 4.16: Get physical rendering mesh. (a) The inputs are a collection of NURBS patches, cracks and positions of control points (Type II). (b) The resolution of rendering mesh is specified (4x4 elements in this subfigure). The yellow hollow points are the vertices of elements. The green hollow points are crack lines' vertices. (c) The purple hollow points are the intersection points of crack lines with element edges. We duplicate crack line segments and split the element edges into line segments by purple points. (d) Polygons of the element separated by crack lines are found iteratively(see Sec. 4.9.2). (e) The world positions of polygon vertices are calculated based on their coordinate in the patch. (f) We assemble all polygons to get the final physical rendering mesh.

with point "Bn". If there is only a solo candidate line, then this line will be the next edge. If there are multiple candidate lines, we choose the one with the smallest clockwise angle between the starting edge and the candidate line. For example, there are two candidate lines after edge "P0G0", i.e., "G0P3" and "G0P4". $\angle P0G0P3$ is smaller than $\angle P0G0P4$. Once a line is selected as the subelement's edge, it is marked as "visited" in the set $\mathcal{SL}$. The last edge will be the line that ends with point "Am". We repeat the above steps until all lines in the set are visited. Therefore, all 2D polygons are found.

4.9.3 Calculate polygons vertices' world-space position

Sec. 4.9.2 find all 2D polygons in a quad element. They share edges and vertices. In the physical rendering mesh, polygons are separated by cracks, and no vertices are shared. To separate polygons, we need to give shared vertices (green and purple

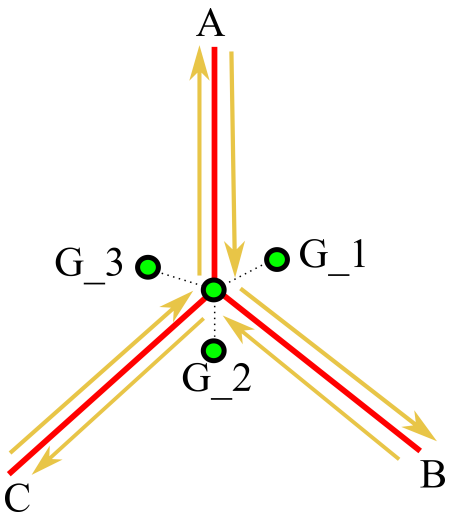
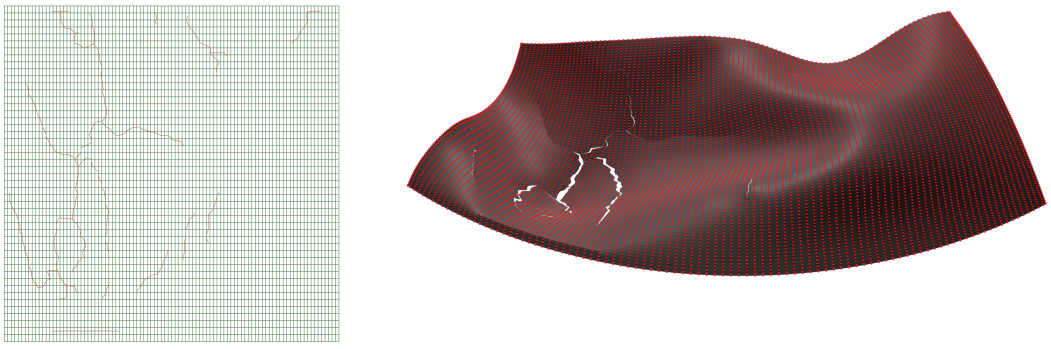


Figure 4.17: Point ”G” is shared by three polygons and three shadow points are created.

hollow points) a different coordinate in the material space. Therefore, their world-space position can be calculated using control points that are on the same side of the crack. As shown in Fig. 4.17, we duplicate shadow points according to the number of polygons that share this point. These shadow points are then deviated along the median line of corresponding polygon's adjacent edges by a small magnitude **Dv**. We typically set $\mathbf{Dv} = 0.00001$. It is small enough to avoid visible gaps but also sufficient to separate polygons.

We use MLS interpolation to calculate the world-space position of these points as they are affected by cracks(see Sec. 4.6). For points that are far away from the crack, we use conventional NURBS interpolation to calculate the position.



(a) The NURBS patch and crack. (b) Control points (red) and physical mesh.

Figure 4.18: Cracked NURBS patch and physical rendering mesh.

Algorithm 5: Get physical rendering mesh

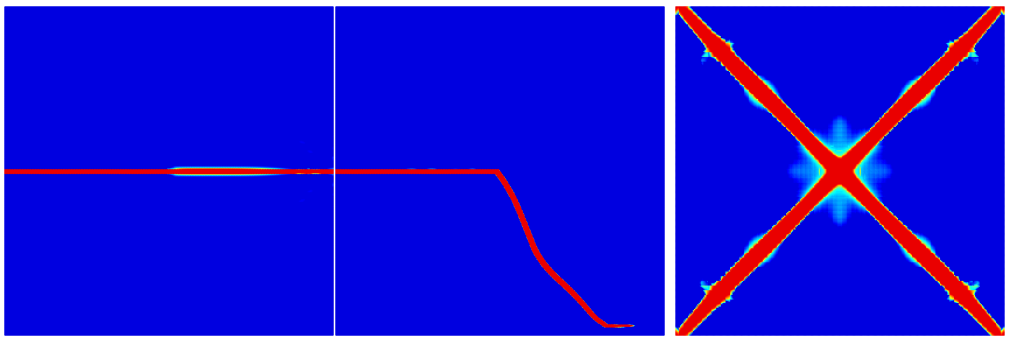
```
   input  : nurbsPatches, cracks, controlPointsPositions
   output: Physical rendering mesh made of face polygons
 1 vertices ← ∅
 2 polygons ← ∅
 3 for nurbsPatch, crack ∈ nurbsPatches, cracks do
 4     SetPatchMeshResolution (R,S)
 5     quads = CreateQuads ()
 6     vertices.Add (quads.vertices())
 7     vertices.Add (crack.vertices())
 8     for quad ∈ quads do
 9         intersectPoints = (quad.edges(), crack)
10         vertices.Add (intersectPoints)
11         lineSegments.Add (SplitEdgeIntoSegements (quad.edges(),intersectPoints))
12         lineSegments.Add (CropCrackLines (crack,intersectPoints))
13         lineSegments.Add (DuplicateReverseSegments (CropCrackLines (crack,intersectPoints)))
14         while !allLineSegmentsVisited() do
15             line = SelectOneUnvisitedLineSegment (lineSegments)
16             polygon.Add (line)
17             line.TagAsVisited ()
18             startPoint = line.start()
19             endPoint = line.end()
20             currentEndPoint = endPoint
21             nextLine ← ∅
22             while nextLine.end() != startPoint do
23                 for candidateLine ∈ lineSegments do
24                     if isUnvisited (candidateLine) && candidateLine.start() == currentEndPoint then
25                         nextLine = candidateLine
26                         polygon.Add (nextLine)
27                         currentEndPoint = nextLine.end()
28
29                     end
30                 end
31             end
32             polygons.Add (polygon)
33         end
34     end
35 end
36  /* Create shadow points and deviate them, see Sec. 4.9.3                                    */
37 SeparateSharedVertex (vertices)
38 CalWorldSpacePosition (vertices, controlPointsPositions)
39 return ExportMesh (vertices, polygons)
```

4.10 Results

In this section, we showcase various examples to demonstrate the versatility of our method for simulating tearing and fracture of thin materials represented as continuum shells. We list all timing information and simulation parameters in Tab. 4.3. All simulations were run on an Intel Core i9-7920X processor (2.90 GHz, 24 cores) with 62.6GB RAM.

4.10.1 Mode tests

We first evaluate the physical accuracy of our method w.r.t the propagation of damage under sustained loading by using three benchmark tests. The most-elementary tests of a fracture simulator are the loading mode tests. Fig. 4.19 shows our propagation



(a) Mode-I. (b) Mode-II. (c) Simply supported plate.

Figure 4.19: Benchmark test results that we use to verify our method by reproducing three experiments of Kiendl et al. (2016).

results under (a) Mode-I and (b) Mode-II loading. Our simulator produces propagation results that remain faithful the expected behaviour of the material with the given loading conditions. We also evaluate our method's ability to resolve fracture within a simply-supported plate under uniform pressure loading as shown in Fig. 4.19 (c). Our method is able to successively initiate four cracks near the center and which then propagate towards the corners. The results are identical to Fig.12 in Kiendl et al. (2016).

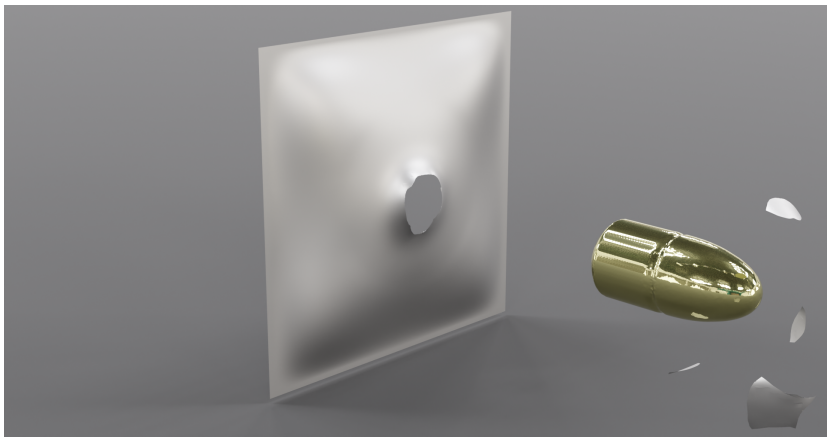


Figure 4.20: A high-speed bullet hits a piece of thin metal sheet and leaves a permanent deformation.

4.10.2 Elasto-plasticity

In Fig. 4.20, we show the result of an simulation, involving plastic-deformation for reproducing permanent-denting behaviour that is common in ductile fracture. For this test, we run a simulation where a high-speed bullet pierces a thin metal sheet, leaving a clean cut as fragment debris fly in the direction of travel. We are naturally able to

Figure 4.21: A sheet is pre-cut with an Archimedean Spiral curve. Our MLS formulation enables exact cutting with complex crack geometry.

incorporate the effect of plasticity in the shell.

4.10.3 Cohesion-free contacts and prescribed cut paths

Using several examples (e.g.,Fig. 4.21 and Fig. 4.14 (c)), we show that our method is able to simulate proper contact forces between crack faces and without cohesion artifacts, which we do by making use of phantom nodes. Fragments will experience contact and friction force along crack faces to ensure clean separation when a crack is introduced. Fig. 4.21 also demonstrates the versatility and controllable-aspect of our method with an example where a thin shell is cut along a prescribed path. Prescribing more than one path is also possible as shown in Fig. 4.14 (c). This pre-cutting exemplifies our ability to incorporate user-control, which is desirable for artistic design and manipulation of simulation results.

4.10.4 Consistent crack paths with branching

Fig. 4.23(a) shows that our method can robustly extract a consistent non-manifold crack path in a time-varying phase field. This overcomes the key problem of sensitivity (to input data) when computing the medial axes that we use to define crack faces. As we fundamentally rely on phase-fields, our system is able to produce damage effects with bifurcation and merging which we use to extract and apply explicit crack paths on thin shells for modelling strong discontinuities. Fig. 4.1 also shows complex crack

patterns generated by our method.

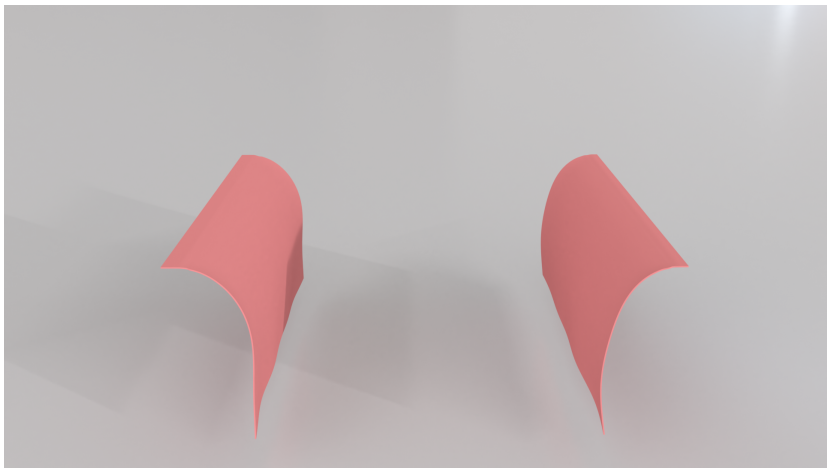


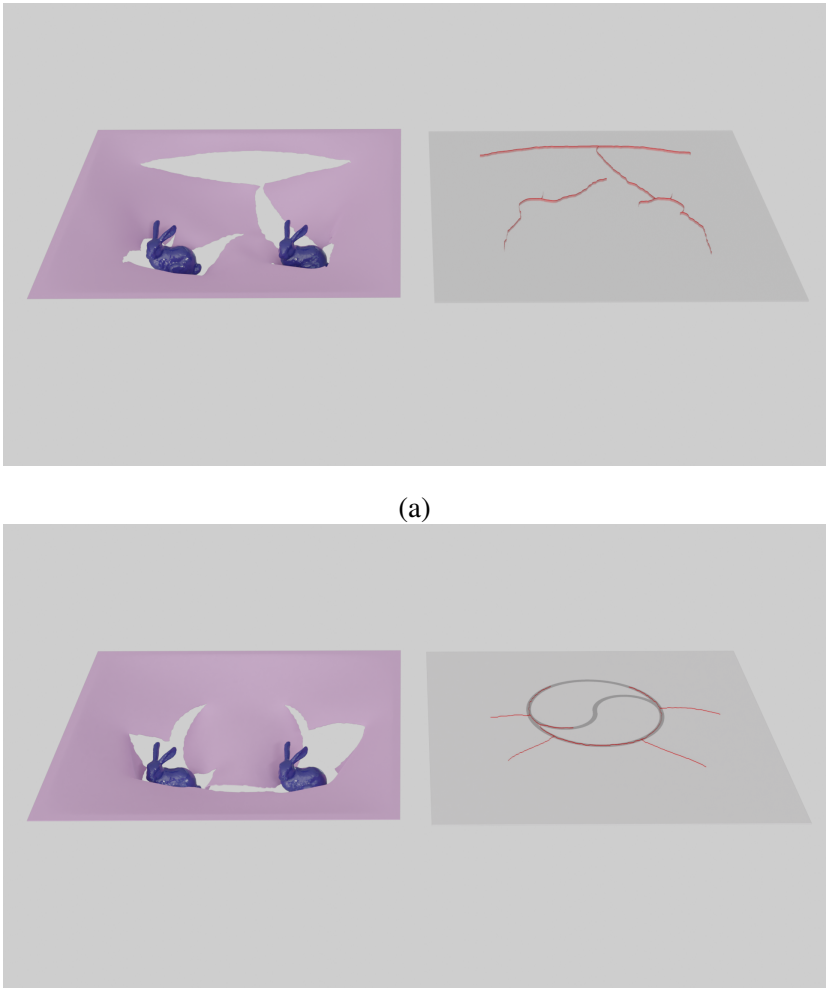
Figure 4.22: We tear a piece of thin rubber with a short guided gap on the top and bottom. The crack starts from the gap and gradually expands to the center.

4.10.5 Influencing fracture with material parameters

Our method likewise facilitates simulating objects with diverse properties since each particle can carry its own material parameters for characterizing behavior. Weakened or reinforced regions can be simply represented as materials with different reference stress thresholds. We can also propagate cracks with complex material coupling scenarios as in Fig. 4.23(b), which shows a crack initiating in a region where the material is softer and propagating to the rest of the domain. Note that the crack doesn't follow the weakened region strictly. Instead, secondary cracks appear in the unweakened region after crack initiation. A more complex example of a prespecified cut is also shown in Fig. 4.21.

4.10.6 Stitching NURBS patches

Robust enforcement of continuity across adjacent NURBS patches is a challenge, which we have addressed in our method. Fig. 4.24 shows the geometric smoothness of a piece of cloth consisting of two NURBS patches. When the two patches are not properly coupled as described in Sec. 4.8, the connection is only preserved through the MPM particles. An example is shown in Fig. 4.24 (b) where the cloth with two (uncoupled) patches drapes over a sphere: The cloth can be clearly seen to exhibit folds at the interface of the two patches since there is no bending resistance. Our approach allows for proper smoothness to enable accurate estimation of surface normal vectors for



(a)

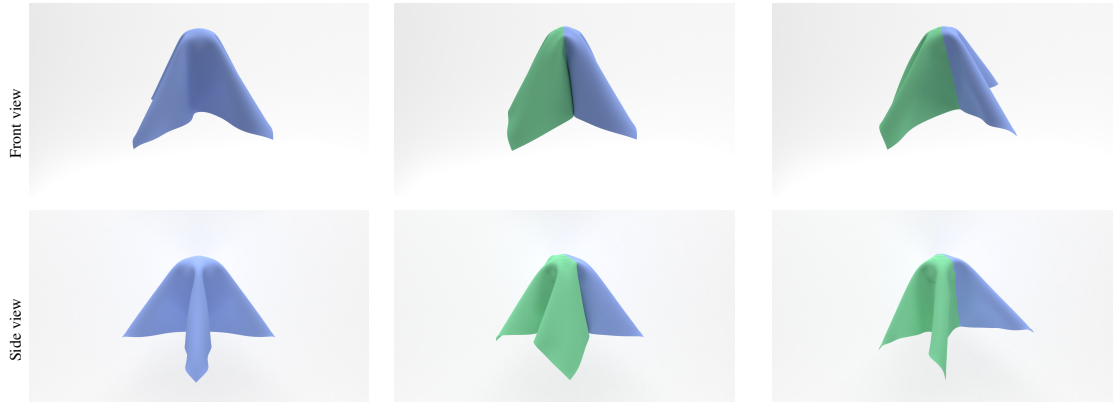
(b)

Figure 4.23: (a) Two rigid bunnies hit a piece of cloth fixed in its four edges. The phase field method produces complex branching and merging damaged region. Our method extracts a consistent crack path which is essential to separate cloth fragments. (b) We use a smaller reference stress threshold to denote weakened material in the central region(shaded grey area). The crack favors the weakened region during its propagation.

computing bending resistance across the patch interface (see Fig. 4.24(a),(c)). Another example is shown in Fig. 4.25, to demonstrate the buckling effect on a cylinder shape. Without the NURBS coupling, the patch interface appears unsmooth and discontinuous, whereas it will be sufficiently smooth to exhibit proper buckling when using our NURBS patch coupling strategy.

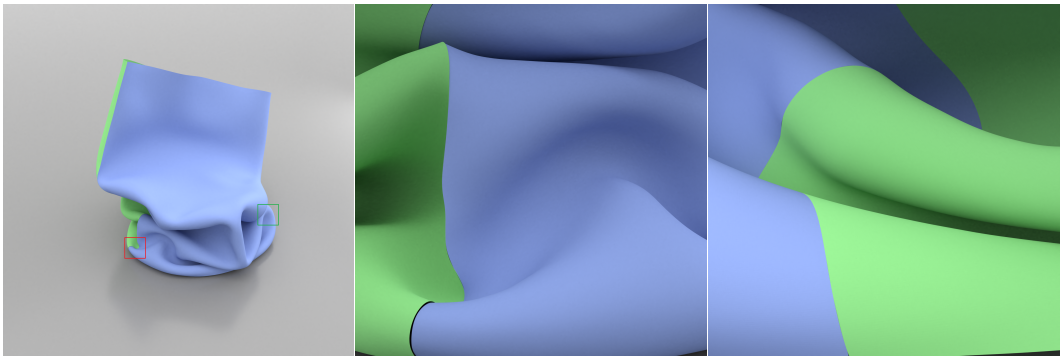
Our coupling method also supports discontinuities passing through the coupling interface. In Fig. 4.26, the garment is torn by force applied on the bottom. Complex tears with bifurcations are produced leading to three fragments in the end. Fig. 4.27 shows a related example in which an adorned mannequin is inflated at rest pose, resulting in

the tearing of its outfit with several tears. We also show a more complex scenario in Fig. 4.1 where a car composed of 12 NURBS patches with diverse materials is crashed by a frontal impactor and then subsequently by a tree. The top 10 patches are coupled with C^1 continuity such that they yield a smooth transition between coupling regions.



(a) One patch. (b) Two patches without coupling. (c) Two patches with C^1 coupling.

Figure 4.24: Effectiveness of our coupling method. (a) A piece of cloth drops onto a sphere. Four corners hang naturally. Only one NURBS patch is used. (b) This cloth is simulated using two non-conforming NURBS patches without coupling. There are 100 and 125 knots in the interface region, respectively. These two patches tend to "glue" together because there is no bending resistance in the interface. (c) We enforce C^1 continuity across two patches. The cloth's four corners hang naturally as (a).



(a) (b) (c)

Figure 4.25: (a) A cylinder composed of two NURBS patches is dropped to the ground. These patches connect without coupling on the left side, and with C^1 coupling on the right side. (b) Zoom-in view of the red squared region in (a), (c) Zoom-in view of the green squared region in (a).

Figure 4.26: A garment is fixed on its shoulder strap and is pulled from its bottom. It is composed of eight NURBS patches which are stitched with C^1 continuity. These patches are also shown with different colors on the right. Branching cracks are generated and pass between patch borders.

4.10.7 Couple with a rigid-body solver

Our local phase-field formulation can likewise be coupled to a rigid body simulator. In Fig. 4.29, we replicate the thin glass breakage test in Chapter 3, where the glass is hit by a spherical marble. The crack initiates from the contact region to form multiple circular patterns outwards.

4.11 Conclusion and Discussion

We have presented an extension to the hybrid Langrangian/Eulerian formulation of thin shells to include controllable fracture. Thin shells are modelled with the Kirchhoff-Love model of elasticity (with a novel in-plane stress evaluation), which is discretized using a NURBS surface based representation. A moving-least squares (MLS) interpolation scheme is then proposed for capturing strong discontinuities in velocity fields with the aide of explicit crack paths. These crack paths are updated according to a stress-dependent evolution of phase-fields that resolve crack initiation and propagation automatically with branching and merging effects. Unlike Chapter 3, this approach provides a consistent approximation of the crack, which also prevents the formation of tiny debris due to changes in past crack geometries. A method for resolving contacts along crack faces in MPM was also presented, based on defining contact normals correctly by considering the co-dimensional nature of thin shells. Finally, we also presented an approach to coupling non-conforming NURBS patches to allow for simulat-

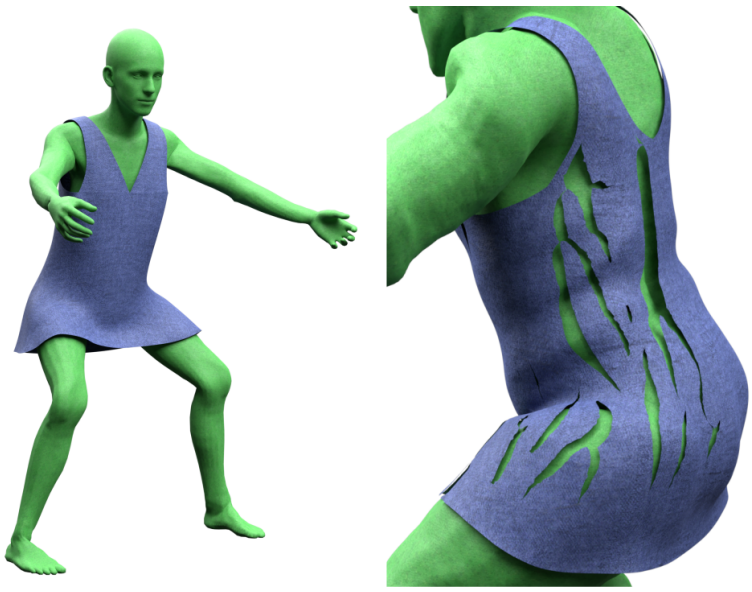


Figure 4.27: Inflatable mannequin: The body of a mannequin dressed in blue is inflated, resulting in the tearing of its outfit (dress). This expansion of the body generates multiple tears within the (eight) NURBS patches, where discontinuities also pass through the interface between these patches. The outfit is modelled as in Fig. 4.26.

Table 4.3: Performance summary. Columns: material density ρ (kg/m^3); Young's modulus E (J/m^2); Poisson's ratio ν; Reference stress σ^f (J/m^2); MPM grid size h (m); MPM time-step size Δt (s); Total particle number in $\Gamma^{(\mathrm{I})}$, $\Gamma^{(\mathrm{II})}$ and $\Gamma^{(\mathrm{III})}$ $(N_{(\mathrm{I}),(\mathrm{II}),(\mathrm{III})})$; Number of Gaussian quadrature points $\in \Gamma^{(\mathrm{IV})}$ $(N_{(\mathrm{IV})})$; Total number of time-steps $(N_{\Delta t})$; Average computation time per time-step in seconds $(s/\Delta t)$. The total time of Fig. 4.29 includes 0.5 hours of rigid body simulation time.

Scene	ρ	E	ν	σ_f	h	Δt	$N_{(\mathrm{I}),(\mathrm{II}),(\mathrm{III})}$	$N_{(\mathrm{IV})}$	$N_{\Delta t}$	$s/\Delta t$
Fig. 4.1	–	–	–	–	1.0E-2	5.0E-6	170K	630K	12000	2.73
Body	500	3.2E8	0.3	6.5E6	–	–	–	–	–	
Window	300	2.8E9	0.3	8.2E6	–	–	–	–	–	
Fig. 4.6	500	3.2E5	0.3	4.5E4	1.5E-2	2.0E-4	23K	160K	4000	1.29
Fig. 4.29	4.0E3	3.2E10	0.18	2.2E7	1.0E-2	1.0E-6	60K	1.25M	4300	4.02
Fig. 4.10	100	3.2E3	0.3	1.5E2	1.0E-2	5.0E-5	20K	160K	3500	1.08
Fig. 4.20	1.0E3	3.2E5	0.33	8.0E4	1.0E-2	2.0E-4	26k	160K	4500	1.57
Fig. 4.23	100	2.8E5	0.3	5.5E4/8.0E3	1.0E-2	1.0E-4	74K	500K	4000	3.00/3.02
Fig. 4.26	20	3.2E3	0.3	5.0E2	9.0E-3	2.0E-4	103K	502K	6000	2.80
Fig. 4.27	20	3.2E3	0.3	5.0E2	9.0E-3	2.0E-4	183K	502K	3400	4.10
Fig. 4.21	1000	4.2E5	0.3	–	1.0E-2	2.0E-4	19K	160K	8000	1.23
Fig. 4.14	1000	6.0E5	0.3	–	5.0E-3	2.0E-4	40K	640K	3000	3.52
Fig. 4.24	200	3.2E5	0.3	–	1.5E-2	3.0E-4	20K	312K	4600	1.38
Fig. 4.24 (b) & (c)	200	3.2E5	0.3	–	1.5E-2	3.0E-4	23K	352K	4600	1.46
Fig. 4.25	100	2.8E5	0.3	–	1.5E-2	3.0E-4	27K	368K	5200	1.72

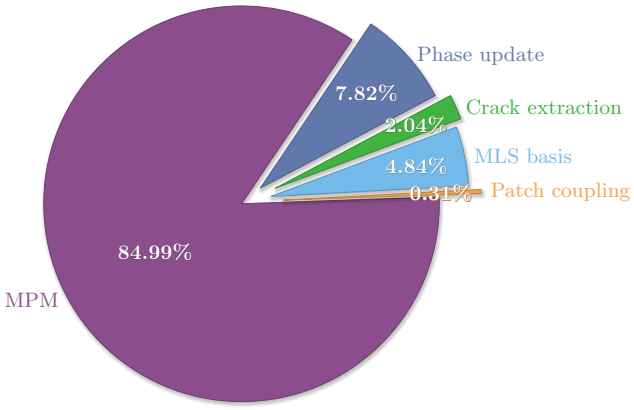


Figure 4.28: Example breakdown of total simulation time using the result shown in Fig. 4.26. A summary of the task-portions is as follows: The 'MPM' portion corresponds to steps 1-5 in Sec. 4.3.2; 'Phase update' is the calculation of phase values at quadrature points (Sec. 4.4.1); 'Crack extraction' corresponds to Sec. 4.4.2; 'MLS basis' involves computing our quadrature points' basis/shape functions and their derivatives while using MLS in crack-influenced regions (Sec. 4.6); Finally, 'Patch coupling' is the task of stitching adjacent NURBS patches as described in Sec. 4.8.

ing on more complex geometries by using a virtual uncommon-knot insertion method that does not introduce additional DOFs nor parameters. The presented method is shown to be capable of simulating challenging scenarios of fracturing thin shells to open new ways toward artist-directed control of this fracture.

Our method is formulated to maximise user-control while obtaining visually plausible results by making a few but notable simplifying assumptions about the evolution of damage. Our elastic constitutive law is dependent on the *history* of deformation but only in the case of sustained external force-loading since we evaluate damage c from the strain at the current timestep. This is in contrast to conventional fully elasto-dynamic MPM methods (e.g.,Wolper et al. (2019, 2020); Cervera and Chiumenti (2006); Homel and Herbold (2017); Borden et al. (2016); Kakouris and Triantafyllou (2017)), where c represents the maximum strain over time to create a permanent stress weakening effect. By using c^{his} (*cf.* Sec. 4.4.1), we effectively cast the evolution of damage into an evolution of actual crack paths, which embody the history of this damage over time and with-which material is strongly separated (i.e., rather than weakly separated as with smeared cracks). The advantage of our simplification is that overstretching of material due to weakening is avoided through healing. This ensures that we only compute damage at the crack tip, which effectively models the existence of *singular crack-tip stress* for evolving our explicit crack paths in the spirit of linear elastic fracture mechanics Chitalu et al. (2020); Richardson et al. (2011); Hahn and

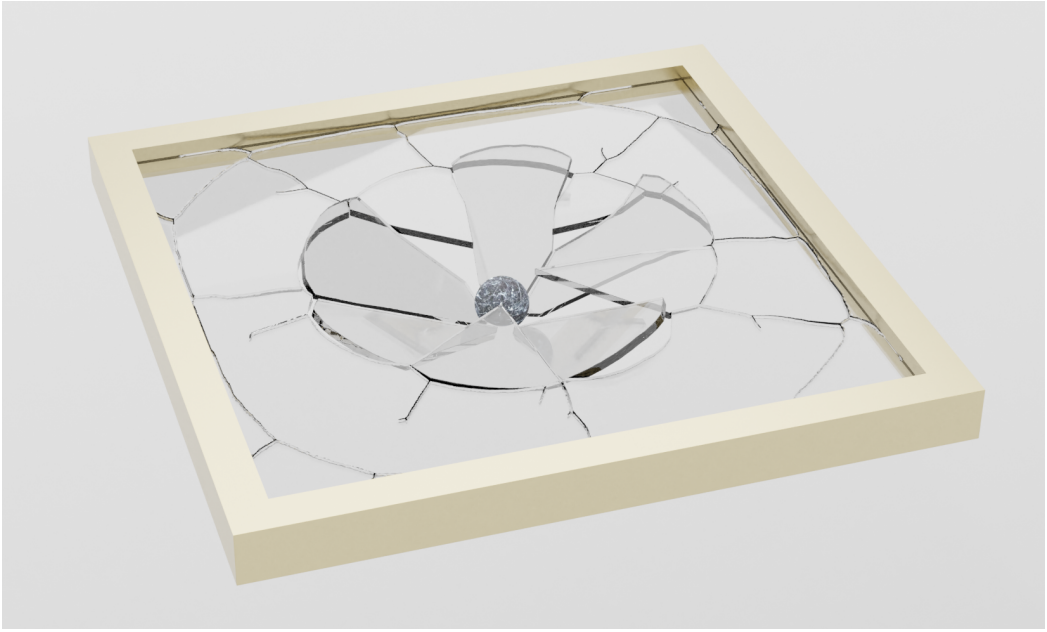


Figure 4.29: A ball hits a thin glass to generate realistic angular crack patterns. Due to the restriction of tiny timestep size, we use a rigid body solver to simulate the dynamics of fragments as Fan et al. (2022).

Wojtan (2015, 2016). In contrast, conventional fully elasto-dynamic MPM methods that use of maximum phase values fall into the dilemma of expanding crack regions forward while simultaneously avoiding thickening in perpendicular normal directions, which we overcome.

We have adopted the MLS method in our formulation for its several advantages, including its ability to satisfy the consistency condition (i.e., that a p-th order MLS approximation can reproduce exactly polynomials of less than or equal to order p) which ensures exact reproduction of NURBS basis functions up-to the given order. Interpolation error due to the use of MLS is therefore practically non-existent, meaning that general crack propagation behavior is unaffected by using MLS basis functions and visual results also remain consistent with the use of NURBS basis function (see also Fig. 4.14).

Comparing with , e.g.,Kaufmann et al. (2009b), we find that the enrichment function of XFEM is available with only C^0 continuity while our continuum shell representation needs at least C^1 continuity. Second, XFEM typically requires a sufficiently large number of quadrature points in one element for capturing discontinuities to mitigate aliasing artifacts. Conversely, the number of quadrature points in our method is independent of crack geometry, and we also avoid aliasing artifacts. Third, XFEM enrichment may fail to capture the crack when its path becomes overly complex (see

Fig. 12 in Kaufmann et al. (2009b)).

While we use a NURBS-based representation, our method is amenable to either NURBS- or Subdivision-based surfaces as their basis functions have the sought property of being twice differentiable, which is required to compute covariant basis vectors $\mathbf{g}_3^{KL}$ for evaluating the deformation gradient. Subdivision surfaces are common in graphics, but NURBS are also ideal for designs in manufacturing of cloth and other models including mechanical tools, vehicles as well as large transportation vessels.

Limitations and future work: As we rely on explicit integration, simulating stiff materials will inherently require extremely small timesteps to impact total simulation time. This issue can of-course be remedied by using implicit integration with larger timesteps. However, formulating an implicit integrator for continuum shell problems remains challenging with additional requirements on devising good preconditioners for a reliable iterative solver.

We successively update damage from in-plane stress with bending resistance but do not strictly account for shearing and/or compression during these updates since we consider stress only at Type-IV (quadrature) particles. Thus, severe shearing or compressive deformation of the continuum shell will only have indirect influence on the weakening of (bending) stress as shearing forces also penalize in-plane deformations. While this is a limitation in principal, our simulations still yield physically plausible results since the propagation of damage is primarily dependent on the in-plane tensile component of stress due to Kirchhoff-Love bending resistance.

Our algorithm facilitates the coupling of NURBS with arbitrary order continuity. However, the inherent tensor-product structure of NURBS surfaces presents challenges in creating intricate geometries, such as those of a bunny or an armadillo. The topology of NURBS patches may require minor adjustments when two patches do not align perfectly at their interface. Nonetheless, this can be largely mitigated through a global refinement of the NURBS at the interfaces. Alternatively, using subdivision surfaces can solve this issue, thanks to our MLS method's natural compatibility with subdivision surface.

We resolve the cohesion artefact by adding a contact force in the cracked region but cannot guarantee penetration-free. The contact force becomes excessively large due to the inaccurate normal estimation when the ratio of the shell fragment's mass to that of a 3D object is tiny. The large contact force can cause oscillation and even self-penetration. In rare cases, the simulation may terminate. However, we find that a

smaller timestep size can effectively resolve the penetration.

Our MPM framework seamlessly supports multiple materials. Currently, we can simulate the coupling of thin shells with rigid and ductile material. In the future, it is worth investigating the coupling of fluid. It is interesting to simulate the scene of shipwreck, e.g., the iceberg breaks the ship, and water flows inside through the hole.

Chapter 5

Conclusions

In this thesis, we have presented a method to simulate fracture in brittle objects and thin shell objects for visual effects. Central to our approach is the fusion of phase field method with the material point method. Objects are discretized using material points that carry information of velocity, displacement, stress, phase, etc. The crack initiation and propagation are handled implicitly by tracking the evolution of a scalar phase field, where complex branching and merging cracks can be simulated at no extra cost. Subsequently, we extract a non-manifold crack surface from the simulated damage phase. The crack surface provides a strong discontinuity which further facilitates the partition of material points and MPM's background grid. This can effectively avoid or alleviate sticky artifacts in MPM and yield a clean and sharp separation.

The main challenge when simulating brittle objects lies in the material's high stiffness, which imposes significant restrictions on the simulation's timestep size. In practice, it's nearly impossible to simulate real brittle materials in an elastodynamic manner. Therefore, we deviate from the conventional approach by opting for quasi-static simulations. A rigid body solver is responsible for collision detection and contact treatment as well as the dynamic motion of fragments, which saves computational cost. We can therefore use a tiny timestep size to simulate complex fracture patterns with our local phase field formulation.

The crack extraction algorithm is based on the Voronoi medial axis. It is simple yet effective in generating a non-manifold crack surface geometry. By using this explicit geometry representation, artificial welding artifact is greatly alleviated thanks to the partitioning of fragments in MPM's background grid. Moreover, it also helps to correctly resolve contact in the crack front.

For enhanced realism, we add brittle fracture patterns according to the simulated

damage phase field. Unlike other post-processing methods, our approach generates a physically correct fracture pattern without the need of artificial input, e.g., texture and displacement map.

As codimensional objects are typically represented as continuous shells, such as NURBS or subdivision surfaces, we incorporate the hybrid Lagrangian/Eulerian formulation to fracture simulation in the simplified Kirchhoff-Love shell model. We adopt the same local phase field method for crack initiation and propagation. However, the conventional 3D Cauchy stress tensor is incompatible with the Kirchhoff-Love's plane-stress assumption. We therefore design a new in-plane stress formulation from the stress tensor decomposition that avoids volume-loss or material separation artefacts in existing in-plane stress formulation.

In order to accurately represent the singularity of the crack front, it's crucial to have an explicit crack path. To achieve this, we improve the crack extraction algorithm to make it consistent all through the fracture simulation. Material points and background grid belonging to different fragments are then partitioned by this consistent crack using an improved phantom node method, which can handle arbitrary number of fragments contact at free. It is not only more accurate in fracture simulation but also avoids the over-stretching artefact resulting from the loss of force resistance in the damaged regions.

We also presented a novel approach to approximate discontinuous field quantities near cracks in NURBS surface using MLS formulation. By using only material points on the same side of the crack, quadrature points are decoupled from material points situated on the opposite side of the crack. We believe it creates a new avenue to model discontinuity in NURBS surface. Considering that complex objects are often composed of multiple NURBS patches, we propose a novel patch coupling method to stitch non-conforming patch while ensuring at least C^1 continuity. This approach smooths MPM background grid's velocity field after particle-to-grid transfer. It can effectively remove NURBS patch's small crease that may arise from direct displacement modifications in the coupling region.

While our hybrid formulation proves to be efficient in simulating brittle and thin shell fractures, it does have certain limitations. The crack extraction algorithm can be computationally demanding, especially in scenarios where cracks are extracted frequently, particularly in cases with a large number of material points. The crack surface may leave a large flat region in brittle object's thick damaged region though it can be mitigated by the fracture detail enrichment method as a post-processing step. For the

brittle fracture simulation, the quasi-static method is not suitable for ductile materials. Similarly, the thin shell fracture simulation method may not be well-suited for materials that are nearly brittle, as it requires extremely small timestep size. To address these challenges, we have explored the combination of both approaches to simulate brittle codimensional objects, yielding promising results, such as in the simulation of thin glass breakage in Fig. 4.29. Our thin shell formulation doesn't consider cracks caused by shearing and compression load. In the future, a unified phase criteria could be proposed to account for these effects. For example, the shearing/compression particles also carry a phase value. The damage phase is then calculated based on the phase of both bending particles and shearing/compression particles.

It is also worth finding an analytic solution for extracting the crack geometry. We have proposed a robust crack geometry extraction algorithm. However, it is a mesh-based method, which induces additional costs. The existing analytic approach only extends the crack surface in a single direction. It fails to handle crack merging and bifurcation phenomena that are common in the real world. One possible way is to convert the crack propagation problem into a minimization problem. The damaged volume stores crack energy (see Fig. 3.19) which should be released along its medial axis because real crack tends to split fragments with minimized energy. We found this method generally works well in manifold regions. In regions involving merging and bifurcation (especially the branching point), it generates a hole due to ambiguity.

The brittle fracture simulation can be further accelerated by adopting a generative method. Although the quasi-static method we proposed greatly saves computational cost, a huge amount of time is devoted to crack simulation, including crack initiation, propagation, and extraction. Therefore, replacing this step is crucial to the speedup of brittle fracture simulation. Recently, Huang and Kanai (2023) proposed a learning-based method that generates fragments using wGAN(Arjovsky et al., 2017). It can effectively generate similar fragments with significantly less time. Unfortunately, the method cannot generalize well in unseen shapes and materials. It does open a new avenue for future work.

Inspired by this work, we want to predict an explicit crack geometry directly from some learning-based approach rather than simulation. We will investigate the possibility of generating crack surfaces given the object's shape and impact location, duration, direction, and magnitude. Thanks to our crack extraction algorithm, we can readily generate the required training dataset.

Appendix A

Weak decoupling of CPIC

Compatible Particle-In-Cell (CPIC) (Hu et al., 2018) is a method providing discontinuous velocity fields in MPM simulations for rigid- to soft-body coupling. However, CPIC provides a weak decoupling of particles and grid nodes across a given surface. In this section, we present a short mathematical description of this weak decoupling, which affects linear deformation problems with small-scale discontinuities (at grid-cell resolution) lasting for prolonged periods.

Fig. A.1 depicts two material particles, p_1 and p_2, which both have mass m and are moving in opposite directions. Their respective velocities are $\mathbf{v}_{p_1} = (0, -v_y)$ and $\mathbf{v}_{p_2} = (0, v_y)$. We additionally place a horizontal (red) dashed line between these particles to represent a crack whose normal direction is $\mathbf{n} = (0, 1)$. We assume, for simplicity, that this crack is static since our problem of concern (brittle fracture) involves little-to-no deformation. Based on the location of the crack in Fig. A.1, particle p_1 is compatible with grid nodes g_1, g_2, g_3, g_4, g_5 and g_6, but it is incompatible with nodes g_7, g_8 and g_9. Conversely, particles p_2 is incompatible with grid nodes g_1, g_2, g_3, g_4, g_5 and g_6, but it is compatible with nodes g_7, g_8, g_9.

Particle to grid transfer: Here we describe the particle-to-grid (P2G) transfer as a pre-requisite for our proof of weak decoupling in CPIC during grid-to-particle transfers (G2P).

At a given time-step n, a node $g_{\mathbf{i}=1,\ldots,6}$ will receive a momentum contribution from particle p_1 according to

$$\begin{aligned} m_{\mathbf{i}}^n &= m w_{1\mathbf{i}} \\ (m\mathbf{v})_{\mathbf{i}}^{n,1} &= m\mathbf{v}_{p_1}^n w_{1\mathbf{i}}. \end{aligned} \tag{A.1}$$

The CPIC formulation will also allows this node to receive information from the in-

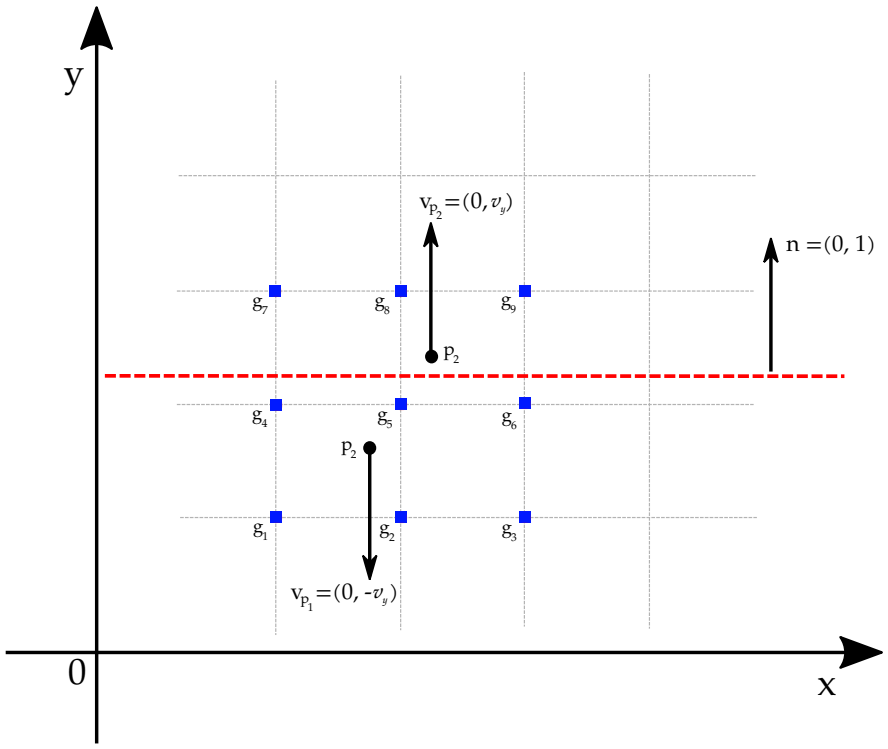


Figure A.1: Two material particles move in contrary direction with a crack in-between.

compatible particle p_2 whose velocity is adjusted according to the projection (*cf.* Eq. 25 in Hu et al. (2018))

$$\begin{aligned}\text{Proj}_r(\mathbf{v}^n_{p_2}, \mathbf{n}_{p_2}, \mathbf{x_i}) &= \mathbf{v}^n_r(\mathbf{x_i}) + \text{Proj}(\mathbf{v}^n_{p_2} - \mathbf{v}^n_r(\mathbf{x_i}), \mathbf{n}_{p_2}, B_r, \mu_r) \\ &= 0 + \text{Proj}(\mathbf{v}^n_{p_2}, \mathbf{n}_{p_2}, B_r, \mu_r) \\ &= \mathbf{v}^n_{p_2}, \end{aligned} \tag{A.2}$$

The subscript r in Eq. A.2 denotes the rigid-body parameters as in Hu et al. (2018), which, in our case, corresponds the crack surface. The mass and momentum of our node at timestep $n+1$ are also given by

$$m^{n+1}_{\mathbf{i}} = mw_{1\mathbf{i}} + mw_{2\mathbf{i}} \tag{A.3}$$

$$\begin{aligned}(m\mathbf{v})^{n+1}_{\mathbf{i}} &= m\mathbf{v}^n_{p_1} w_{1\mathbf{i}} + m\text{Proj}_r(\mathbf{v}^n_{p_2}, \mathbf{n}_{p_2}, \mathbf{x_i}) w_{2\mathbf{i}} \\ &\equiv m\mathbf{v}^n_{p_1} w_{1\mathbf{i}} + m\mathbf{v}^n_{p_2} w_{2\mathbf{i}}, \end{aligned} \tag{A.4}$$

with its velocity according to

$$\begin{aligned}\mathbf{v}^{n+1}_{\mathbf{i}} &= \frac{m\mathbf{v}^n_{p_1} w_{1\mathbf{i}} + m\mathbf{v}^n_{p_2} w_{2\mathbf{i}}}{mw_{1\mathbf{i}} + mw_{2\mathbf{i}}} \\ &= \frac{\mathbf{v}^n_{p_1} w_{1\mathbf{i}} + \mathbf{v}^n_{p_2} w_{2\mathbf{i}}}{w_{1\mathbf{i}} + w_{2\mathbf{i}}}. \end{aligned} \tag{A.5}$$

Grid to particle transfer: Having provided an overview of P2G steps in context of Fig. A.1, we now describe the G2P transfer step, to reveal the weak decoupling problem of CPIC.

For a moment, let $\Delta t = 0$, which implies that the velocity of particle p_1 at time-step $n+1$ should be exactly the same as that of time-step n (*i.e.* $\mathbf{v}_{p_1}^{n+1} = \mathbf{v}_{p_1}^{n}$). However, this does not hold for our simple example in Fig. A.1 (*i.e.* the case of having two points within kernel radius and on opposite sides of a crack). The source of the issue is that a CPIC G2P transfer will gather contributions from compatible *and* incompatible nodes

$$\mathbf{v}_{p_1}^{n+1} = \sum_{j \in \mathbf{i}^{p_1^+}} w_{1j} \mathbf{v}_j^{n+1} + \sum_{j \in i^{p_1^-}} w_{1j} \mathbf{v}_{p_1}^{n}, \tag{A.6}$$

where j is the local grid node index w.r.t p_1, with the notation $^+$ and $^-$ denoting compatible and incompatible nodes, respectively. Furthermore, since $\sum_j w_{1j} = 1$, we can also rewrite $\mathbf{v}_{p_1}^{n}$ as

$$\mathbf{v}_{p_1}^{n} = \sum_j w_{1j} \mathbf{v}_{p_1}^{n} = \sum_{j \in \mathbf{i}^{p_1^+}} w_{1j} \mathbf{v}_{p_1}^{n} + \sum_{j \in \mathbf{i}^{p_1^-}} w_{1j} \mathbf{v}_{p_1}^{n}, \tag{A.7}$$

to verify that indeed $\mathbf{v}_{p_1}^{n+1} - \mathbf{v}_{p_1}^{n} = 0$. However,

$$\begin{aligned}
\mathbf{v}_{p_1}^{n+1} - \mathbf{v}_{p_1}^{n} &= \sum_{j \in \mathbf{i}^{p_1^+}} w_{1j} \left(\mathbf{v}_j^{n+1} - \mathbf{v}_{p_1}^{n} \right) \\
&= \sum_{j \in \mathbf{i}^{p_1^+}} w_{1j} \left(\frac{w_{1j} - w_{2j}}{w_{1j} + w_{2j}} - 1 \right) \mathbf{v}_{p_1}^{n} \\
&= \sum_{j \in \mathbf{i}^{p_1^+}} w_{1j} \frac{-2w_{2j}}{w_{1j} + w_{2j}} \mathbf{v}_{p_1}^{n} \\
&\neq 0
\end{aligned} \tag{A.8}$$

which demonstrates CPIC introduces a weak decoupling within kernel radius. Thus, CPIC will not capture discontinuous velocity fields at grid-cell resolution in the vicinity of the crack. Therefore, we propose a simple scheme to introduce strong decoupling near the crack in next section.

Appendix B

Gradient of damage time

The damage time at node $\mathbf{i}$ is approximated using the neighbourhood particles p by

$$\tau_{\mathbf{i}} = \frac{\sum_p w_{\mathbf{i}p}\tau_p}{\sum_p w_{\mathbf{i}p}}. \tag{B.1}$$

For a point $\mathbf{x}$ around point α, the damage time of $\mathbf{x}$ can be approximated using moving least squares

$$\tau = \sum_{\mathbf{i}} w_{\mathbf{i}\alpha} P^{\mathsf{T}}(\mathbf{x}-\mathbf{x}_\alpha) M^{-1}(\mathbf{x}_\alpha) P(\mathbf{x}_{\mathbf{i}}-\mathbf{x}_\alpha)\tau_{\mathbf{i}}, \tag{B.2}$$

where $w_{i\alpha}$ is the weight between point α and node $\mathbf{i}$, P is polynomial basis, M is a constant when a quadratic or cubic kernel is used (see Hu *et al.* (Hu et al., 2018) for details). It follows that the the gradient of α can be computed as the derivative of Eq. B.2 by

$$\begin{aligned}
\nabla\tau_\alpha &= \nabla\tau|_{\mathbf{x}=\mathbf{x}_\alpha} \\
&= \sum_{\mathbf{i}} w_{\mathbf{i}\alpha} \nabla P^{\mathsf{T}}(\mathbf{x}-\mathbf{x}_\alpha) M^{-1}(\mathbf{x}_\alpha) P(\mathbf{x}_{\mathbf{i}}-\mathbf{x}_\alpha)\tau_{\mathbf{i}}|_{\mathbf{x}=\mathbf{x}_\alpha} \quad , \\
&= \sum_{\mathbf{i}} w_{\mathbf{i}\alpha} M^{-1}(\mathbf{x}_\alpha)(\nabla P^{\mathsf{T}}(\mathbf{x}-\mathbf{x}_\alpha) P(\mathbf{x}_{\mathbf{i}}-\mathbf{x}_\alpha))\tau_{\mathbf{i}}|_{\mathbf{x}=\mathbf{x}_\alpha}
\end{aligned} \tag{B.3}$$

where we use a linear polynomial basis with

$$P(\mathbf{x}-\mathbf{x}_\alpha) = [1, (\mathbf{x}-\mathbf{x}_\alpha)^{\mathsf{T}}]^{\mathsf{T}}$$

to give

$$\nabla P^T(\mathbf{x}-\mathbf{x}_\alpha) P(\mathbf{x}_{\mathbf{i}}-\mathbf{x}_\alpha) = [\mathbf{x}_{\mathbf{i}}-\mathbf{x}_\alpha]^{\mathsf{T}},$$

from which we get Eq. 3.14.

Appendix C

Fracture energy and strain energy

Here we give the equations for total elastic- and fracture energy referenced in Sec. 3.8. Definitions are similar to Kakouris and Triantafyllou (Kakouris and Triantafyllou, 2017) with minor adjustments to account for our local formulation. Our elastic energy is

$$\Psi = \int_{\Omega} \left\{ (1-c)^2 \Psi^+ + \Psi^- \right\} dx, \tag{C.1}$$

where Ψ^+ and Ψ^- are the tensile and compressive strain energy, respectively. Our fracture energy is

$$\Psi_{\mathrm{f}} = \int_{\Omega} G_f \left[\frac{c^2}{4l_0} + l_0 \frac{\partial c}{\partial x_i} \frac{\partial c}{\partial x_j} \frac{\partial c}{\partial x_k} \right] dx. \tag{C.2}$$

where l_0 =2.5E-4 is a length scale factor whose value we choose to be the same as Kakouris and Triantafyllou (Kakouris and Triantafyllou, 2017). Our definition is slightly different since the parameter G_c is used to represent the energy release rate of crack surface. This parameter does not exist in our method so we use $G_c = G_f$ in our comparisons.

Appendix D

Calculation of patch coupling matrix

D.1 Insert one knot to B-spline

We now consider the simple case of inserting one new knot, $\overline{k} \in [k_m, k_{m+1})$, to the original knot vector $\mathbf{k}$ to form a new knot vector $\overline{\mathbf{k}}$ in B-spline. The physical curve before and after knot insertion should be the same geometrically, which implies:

$$\sum_{i=1}^{n} N_{i,p}(\xi)\mathbf{P}_i = \sum_{i=1}^{n+1} \overline{N}_{i,p}(\xi)\mathbf{Q}_{i+1} \tag{D.1}$$

where $N_{i,p}(\xi)$ is the basis function before insertion, $\overline{N}_{i,p}(\xi)$ is the one after insertion, p is the order, n is the number of control points, and $\mathbf{P}$ and $\mathbf{Q}$ are control points before and after knot insertion, respectively. Since a point is only controlled by $p+1$ nearby control points in the parametric space, we focus on these points:

$$\sum_{i=m-p}^{m} N_{i,p}(\xi)\mathbf{P}_i = \sum_{i=m-p}^{m+1} \overline{N}_{i,p}(\xi)\mathbf{Q}_{i+1} \tag{D.2}$$

$N_{i,p}(\xi)$ for $i = m-p, ..., m$ can be expressed as (*cf.* Eq (5.6) in Piegl and Tiller (1996)):

$$N_{i,p}(\xi) = \frac{\overline{k} - \overline{k}_i}{k_{i+p+1} - \overline{k}_i}\overline{N}_{i,p}(\xi) + \frac{\overline{k}_{i+p+2} - \overline{k}}{\overline{k}_{i+p+2} - \overline{k}_{i+1}}\overline{N}_{i+1,p}(\xi) \tag{D.3}$$

Substituting Eq. D.3 into Eq. D.2 and using knot vector $\mathbf{k}$ in place of $\overline{\mathbf{k}}$ yields:

$$\begin{aligned} \mathbf{Q}_{m-p} &= \mathbf{P}_{m-p} \\ \mathbf{Q}_i &= \alpha_i \mathbf{P}_i + (1-\alpha_i)\mathbf{P}_{i-1}, \quad m-p+1 \le i \le m \\ \mathbf{Q}_{m+1} &= \mathbf{P}_m \end{aligned} \tag{D.4}$$

where we set

$$\alpha_i = \frac{\overline{k} - k_i}{k_{i+p} - k_i} \tag{D.5}$$

Substituting Eq. D.4 into Eq. D.1 yields the formula for computing the positions of all new control points as:

$$\mathbf{Q}_i = \alpha_i \mathbf{P}_i + (1 - \alpha_i)\mathbf{P}_{i-1} \tag{D.6}$$

where

$$\alpha_i = \begin{cases} 1, & i \leq m - p \\ \frac{\overline{k} - k_i}{k_{i+p} - k_i}, & m - p + 1 \leq i \leq m \\ 0, & m + 1 \leq i \end{cases} \tag{D.7}$$

D.2 Insert multiple knots to NURBS

Inserting multiple knots to B-spline can be computed recursively using Eq. D.6. However, the weight of each control point is also needed if the spline is NURBS.

Let $\dot{\mathbf{k}} = [\overline{k}_1, \overline{k}_2, ..., \overline{k}_r]$ be the set of r knots inserted into the original knot vector $\mathbf{k}$ of NURBS curve. P and Q are the original control points and new control points, respectively. They can be related through the affine matrix U as

$$\begin{aligned} Q &= UP \\ &= (\overline{\mathbf{W}})^{-1}\mathbf{A}\mathbf{W}P \end{aligned} \tag{D.8}$$

where matrices $\mathbf{W}$ and $\overline{\mathbf{W}}$ are diagonal matrices containing NURBS weights before and after knots insertion, respectively:

$$\mathbf{W} = \begin{bmatrix} w_1 & & & \\ & w_2 & & \\ & & \cdots & \\ & & & w_n \end{bmatrix} \tag{D.9}$$

$$\overline{\mathbf{W}} = \begin{bmatrix} w_1 & & & \\ & w_2 & & \\ & & \cdots & \\ & & & w_{n+r} \end{bmatrix} \tag{D.10}$$

A is a transformation matrix which is defined recursively as $\mathbf{A} = \mathbf{A}^1\mathbf{A}^2...\mathbf{A}^r$. Each matrix represents one insertion of new knot k_j.

$$A^j = \begin{bmatrix} \alpha_1^j & 0 & & ... & & & 0 \\ 1-\alpha_2^j & \alpha_2^j & 0 & & ... & & 0 \\ 0 & 1-\alpha_3^j & \alpha_3^j & 0 & & & 0 \\ \vdots & & & \ddots & & & \vdots \\ 0 & ... & & & 0 & 1-\alpha_{n+j-1}^j & \alpha_{n+j-1}^j \\ 0 & & ... & & & 0 & 1-\alpha_{n+j}^j \end{bmatrix} \tag{D.11}$$

where $i = 1,2,...,n+j$ and $j = 1,2,...,r$.

Bibliography

Ambati, M., Kiendl, J., and De Lorenzis, L. (2018). Isogeometric kirchhoff–love shell formulation for elasto-plasticity. *Computer Methods in Applied Mechanics and Engineering*, 340:320–339.

Amiri, F., Millán, D., Shen, Y., Rabczuk, T., and Arroyo, M. (2014). Phase-field modeling of fracture in linear thin shells. *Theoretical and Applied Fracture Mechanics*, 69:102–109.

Andrade, H., Trevelyan, J., and Leonel, E. (2022). A nurbs-discontinuous and enriched isogeometric boundary element formulation for two-dimensional fatigue crack growth. *Engineering Analysis with Boundary Elements*, 134:259–281.

Areias, P., Rabczuk, T., and Msekh, M. (2016). Phase-field analysis of finite-strain plates and shells including element subdivision. *Computer Methods in Applied Mechanics and Engineering*, 312:322–350.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR.

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.

Baraff, D. and Witkin, A. (1998). Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 43–54.

Bargteil, A. W., Wojtan, C., Hodgins, J. K., and Turk, G. (2007). A finite element method for animating large viscoplastic flow. *ACM transactions on graphics (TOG)*, 26(3):16–es.

Borden, M. J., Hughes, T. J., Landis, C. M., Anvari, A., and Lee, I. J. (2016). A phase-field formulation for fracture in ductile materials: Finite deformation balance

law derivation, plastic degradation, and stress triaxiality effects. *Computer Methods in Applied Mechanics and Engineering*, 312:130–166. Phase Field Approaches to Fracture.

Bouclier, R., Passieux, J.-C., and Salaün, M. (2017). Development of a new, more regular, mortar method for the coupling of nurbs subdomains within a nurbs patch: Application to a non-intrusive local enrichment of nurbs patches. *Computer Methods in Applied Mechanics and Engineering*, 316:123–150.

Bourdin, B., Francfort, G. A., and Marigo, J.-J. (2000). Numerical experiments in revisited brittle fracture. *Journal of the Mechanics and Physics of Solids*, 48(4):797–826.

Brackbill, J. U., Kothe, D. B., and Ruppel, H. M. (1988). Flip: a low-dissipation, particle-in-cell method for fluid flow. *Computer Physics Communications*, 48(1):25–38.

Breen, D. E., House, D. H., and Wozny, M. J. (1994). Predicting the drape of woven cloth using interacting particles. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 365–372.

Bridson, R. (2007). Fast poisson disk sampling in arbitrary dimensions. *SIGGRAPH sketches*, 10:1.

Brivadis, E., Buffa, A., Wohlmuth, B., and Wunderlich, L. (2015). Isogeometric mortar methods. *Computer Methods in Applied Mechanics and Engineering*, 284:292–319.

Busaryev, O., Dey, T. K., and Wang, H. (2013). Adaptive fracture simulation of multi-layered thin plates. *ACM Trans. Graph.*, 32(4).

Cervera, M. and Chiumenti, M. (2006). Mesh objective tensile cracking via a local continuum damage model and a crack tracking technique. *Computer methods in applied mechanics and engineering*, 196(1-3):304–320.

Chau-Dinh, T., Zi, G., Lee, P.-S., Rabczuk, T., and Song, J.-H. (2012). Phantom-node method for shell models with arbitrary cracks. *Computers & Structures*, 92:242–256.

Chaudhury, A. and Godin, C. (2020). Skeletonization of plant point cloud data using stochastic optimization framework. *Frontiers in Plant Science*, 11:773.

Chen, L., Lingen, E. J., and de Borst, R. (2017). Adaptive hierarchical refinement of nurbs in cohesive fracture analysis. *International Journal for Numerical Methods in Engineering*, 112(13):2151–2173.

Chen, W., Zhu, F., Zhao, J., Li, S., and Wang, G. (2018). Peridynamics-based fracture animation for elastoplastic solids. *Computer Graphics Forum*, 37(1):112–124.

Chen, Y. and Kulasegaram, S. (2009). Numerical modelling of fracture of particulate composites using sph method. *Computational Materials Science*, 47(1):60–70.

Chen, Z., Yao, M., Feng, R., and Wang, H. (2014). Physics-inspired adaptive fracture refinement. *ACM Trans. Graph.*, 33(4):113:1–113:7.

Chitalu, F. M., Dubach, C., and Komura, T. (2018). Bulk-synchronous parallel simultaneous bvh traversal for collision detection on gpus. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 1–9.

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

Choo, J. and Sun, W. (2018). Coupled phase-field and plasticity modeling of geological materials: From brittle fracture to ductile flow. *Computer Methods in Applied Mechanics and Engineering*, 330:1 – 32.

Cleary, P. W. and Das, R. (2008). The potential for sph modelling of solid deformation and fracture. In *IUTAM Symposium on Theoretical, Computational and Modelling Aspects of Inelastic Media: Proceedings of the IUTAM Symposium held at Cape Town, South Africa, January 14–18, 2008*, pages 287–296. Springer.

Clyde, D., Teran, J., and Tamstorf, R. (2017). Simulation of nonlinear kirchhoff-love thin shells using subdivision finite elements. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA*, volume 17.

Coox, L., Greco, F., Atak, O., Vandepitte, D., and Desmet, W. (2017). A robust patch coupling method for nurbs-based isogeometric analysis of non-conforming multipatch surfaces. *Computer Methods in Applied Mechanics and Engineering*, 316:235–260.

Coumans, E. (2015). Bullet physics simulation. In *ACM SIGGRAPH 2015 Courses*, SIGGRAPH '15. Association for Computing Machinery.

Cox, M. G. (1972). The numerical evaluation of b-splines. *IMA Journal of Applied mathematics*, 10(2):134–149.

Culver, T., Keyser, J., and Manocha, D. (2004). Exact computation of the medial axis of a polyhedron. *Computer Aided Geometric Design*, 21(1):65–98.

Dieter, G. E. (1986). *Mechanical Metallurgy*. McGraw-Hill Book Company.

Domaradzki, J. and Martyn, T. (2018). Procedural fracture of shell objects.

Elek, P. and Jaramaz, S. (2008). Fragment size distribution in dynamic fragmentation: Geometric probability approach. *FME transactions*, 36(2):59–65.

Erdogan, F. and Sih, G. (1963). On the crack extension in plates under plane loading and transverse shear.

Etzion, M. and Rappoport, A. (2002). Computing voronoi skeletons of a 3-d polyhedron by space subdivision. *Computational Geometry*, 21(3):87–120.

Evans, M. W., Harlow, F. H., and Bromberg, E. (1957). The particle-in-cell method for hydrodynamic calculations. Technical report, LOS ALAMOS NATIONAL LAB NM.

Fan, L., Chitalu, F. M., and Komura, T. (2022). Simulating brittle fracture with material points. *ACM Trans. Graph.*, 41(5).

Francfort, G. A. and Marigo, J.-J. (1998). Revisiting brittle fracture as an energy minimization problem. *Journal of the Mechanics and Physics of Solids*, 46(8):1319–1342.

Glondu, L., Marchal, M., and Dumont, G. (2013). Real-time simulation of brittle fracture using modal analysis. *IEEE Transactions on Visualization and Computer Graphics*, 19(2):201–209.

Gou, Y. and Zhu, H. (2022). Automatic modelling of heterotypic tunnel structures via nurbs-based meshfree method. *Engineering Analysis with Boundary Elements*, 144:539–555.

Grinspun, E., Hirani, A. N., Desbrun, M., and Schröder, P. (2003). Discrete Shells. In Breen, D. and Lin, M., editors, *Symposium on Computer Animation*. The Eurographics Association.

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):1–15.

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. *Computer Modeling in Engineering & Sciences*, 113(4):389–410.

Gurson, A. L. (1977). Continuum Theory of Ductile Rupture by Void Nucleation and Growth: Part I—Yield Criteria and Flow Rules for Porous Ductile Media. *Journal of Engineering Materials and Technology*, 99(1):2–15.

Ha, Y. D. and Bobaru, F. (2011). Characteristics of dynamic brittle fracture captured with peridynamics. *Engineering Fracture Mechanics*, 78(6):1156–1168.

Hahn, D. (2017). *Brittle fracture simulation with boundary elements for computer graphics*. PhD thesis, Institute of Science and Technology Austria, Klosterneuburg, Austria.

Hahn, D. and Wojtan, C. (2015). High-resolution brittle fracture simulation with boundary elements. *ACM Trans. Graph.*, 34(4):151:1–151:12.

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

Harmon, D., Vouga, E., Tamstorf, R., and Grinspun, E. (2008). Robust treatment of simultaneous collisions. In *ACM SIGGRAPH 2008 papers*, pages 1–4.

Hellrung, J., Selle, A., Shek, A., Sifakis, E., and Teran, J. (2009). Geometric fracture modeling in bolt. In *SIGGRAPH 2009: Talks*, SIGGRAPH '09. Association for Computing Machinery.

Homel, M. A. and Herbold, E. B. (2017). Field-gradient partitioning for fracture and frictional contact in the material point method. *International Journal for Numerical Methods in Engineering*, 109(7):1013–1044.

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.

Hu, Y., Feng, G., Li, S., Sheng, W., and Zhang, C. (2020). Numerical modelling of ductile fracture in steel plates with non-ordinary state-based peridynamics. *Engineering Fracture Mechanics*, 225:106446.

Huang, H., Wu, S., Cohen-Or, D., Gong, M., Zhang, H., Li, G., and Chen, B. (2013). L1-medial skeleton of point cloud. *ACM Trans. Graph.*, 32(4):65–1.

Huang, Y. and Kanai, T. (2023). Deepfracture: A generative approach for predicting brittle fractures. *arXiv preprint arXiv:2310.13344*.

Irwin, G. R. (1957). Analysis of stresses and strains near the end of a crack traversing a plate. *J. appl. Mech.*

Jacobson, A., Panozzo, D., et al. (2018). libigl: A simple C++ geometry processing library. https://libigl.github.io/.

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.

Jiang, C., Schroeder, C., Selle, A., Teran, J., and Stomakhin, A. (2015). The affine particle-in-cell method. *ACM Transactions on Graphics (TOG)*, 34(4):1–10.

Kakouris, E. G. and Triantafyllou, S. P. (2017). Phase-field material point method for brittle fracture. *International Journal for Numerical Methods in Engineering*, 112(12):1750–1776.

Kakouris, E. G. and Triantafyllou, S. P. (2019). Phase-field material point method for dynamic brittle fracture with isotropic and anisotropic surface energy. *Computer Methods in Applied Mechanics and Engineering*, 357:112503.

Kaufmann, P., Martin, S., Botsch, M., Grinspun, E., and Gross, M. (2009a). Enrichment textures for detailed cutting of shells. *ACM Trans. Graph.*, 28(3):50:1–50:10.

Kaufmann, P., Martin, S., Botsch, M., Grinspun, E., and Gross, M. (2009b). Enrichment textures for detailed cutting of shells. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, New York, NY, USA. Association for Computing Machinery.

Kiendl, J., Ambati, M., De Lorenzis, L., Gomez, H., and Reali, A. (2016). Phase-field description of brittle fracture in plates and shells. *Computer Methods in Applied Mechanics and Engineering*, 312:374–394.

Kikis, G., Ambati, M., De Lorenzis, L., and Klinkel, S. (2021). Phase-field model of brittle fracture in reissner–mindlin plates and shells. *Computer Methods in Applied Mechanics and Engineering*, 373:113490.

Klár, G., Gast, T., Pradhana, A., Fu, C., Schroeder, C., Jiang, C., and Teran, J. (2016). Drucker-prager elastoplasticity for sand animation. *ACM Trans. Graph.*, 35(4).

Klein, R. (1989). *Concrete and Abstract Voronoi Diagrams*, volume 400 of *Lecture Notes in Computer Science*. Springer.

Koschier, D., Bender, J., and Thuerey, N. (2017). Robust extended finite elements for complex cutting of deformables. *ACM Trans. Graph.*, 36(4):55:1–55:13.

Koschier, D., Lipponer, S., and Bender, J. (2014). Adaptive tetrahedral meshes for brittle fracture simulation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '14, pages 57–66. Eurographics Association.

Kurlin, V. (2015). A one-dimensional homologically persistent skeleton of an unstructured point cloud in any metric space. In *Computer Graphics Forum*, volume 34, pages 253–262. Wiley Online Library.

Lemaitre, J. and Chaboche, J.-L. (1978). Aspect phénoménologique de la rupture par endommagement. *J Méc Appl*, 2(3).

Lévy, B. (2016). Robustness and efficiency of geometric programs. *Comput. Aided Des.*, 72(C):3–12.

Li, W., Nguyen-Thanh, N., Huang, J., and Zhou, K. (2020). Adaptive analysis of crack propagation in thin-shell structures via an isogeometric-meshfree moving least-squares approach. *Computer Methods in Applied Mechanics and Engineering*, 358:112613.

Liang, Y., Zhang, X., and Liu, Y. (2021). Extended material point method for the three-dimensional crack problems. *International Journal for Numerical Methods in Engineering*, 122(12):3044–3069.

Littlewood, D. (2015). Peri-dynamics. Technical report.

Lu, J. and Zheng, C. (2014). Dynamic cloth simulation by isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 268:475–493.

Lv, A., Zhu, Y., and Xian, C. (2022). Efficient cloth simulation based on the material point method. *Computer Animation and Virtual Worlds*, 33(3-4):e2073.

Ma, C. M. and Sonka, M. (1996). A fully parallel 3d thinning algorithm and its applications. *Computer vision and image understanding*, 64(3):420–433.

Ma, J., Bae, S. W., and Choi, S. (2012). 3d medial axis point approximation using nearest neighbors and the normal field. *The Visual Computer*, 28(1):7–19.

Mitchell, N., Aanjaneya, M., Setaluri, R., and Sifakis, E. (2015). Non-manifold level sets: A multivalued implicit surface representation with applications to self-collision processing. *ACM Trans. Graph.*, 34(6).

Molino, N., Bao, Z., and Fedkiw, R. (2005a). A virtual node algorithm for changing mesh topology during simulation. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05. ACM.

Molino, N., Bao, Z., and Fedkiw, R. (2005b). A virtual node algorithm for changing mesh topology during simulation. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, page 4–es, New York, NY, USA. Association for Computing Machinery.

Mororó, L., Poot, A., and van der Meer, F. (2022). Skeleton curve and phantom node method for the thick level set approach to fracture. *Engineering Fracture Mechanics*, 268:108443.

Mould, D. (2005). Image-guided fracture. In *Proceedings of Graphics Interface 2005*, pages 219–226.

Müller, M., Chentanez, N., and Kim, T.-Y. (2013). Real time dynamic fracture with volumetric approximate convex decompositions. *ACM Trans. Graph.*, 32(4).

Müller, M., McMillan, L., Dorsey, J., and Jagnow, R. (2001). Real-time simulation of deformation and fracture of stiff materials. In Magnenat-Thalmann, N. and Thalmann, D., editors, *Computer Animation and Simulation 2001*, pages 113–124. Springer Vienna.

Museth, K., Lait, J., Johanson, J., Budsberg, J., Henderson, R., Alden, M., Cucka, P., Hill, D., and Pearce, A. (2013). Openvdb: an open-source data structure and toolkit for high-resolution volumes. In *Acm siggraph 2013 courses*, pages 1–1.

Nairn, J. and Guo, Y. (2005). Material point method calculations with explicit cracks, fracture parameters, and crack propagation. *11th International Conference on Fracture 2005, ICF11*, 2.

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

Narain, R., Pfaff, T., and O'Brien, J. F. (2013). Folding and crumpling adaptive sheets. *ACM Transactions on Graphics (TOG)*, 32(4):1–8.

Narain, R., Samii, A., and O'brien, J. F. (2012). Adaptive anisotropic remeshing for cloth simulation. *ACM transactions on graphics (TOG)*, 31(6):1–10.

Nesme, M., Kry, P. G., Jeřábková, L., and Faure, F. (2009). Preserving topology and elasticity for embedded deformable models. *ACM Trans. Graph.*, 28(3).

Nguyen, V. P., Kerfriden, P., Brino, M., Bordas, S. P., and Bonisoli, E. (2014). Nitsche's method for two and three dimensional nurbs patch coupling. *Computational Mechanics*, 53:1163–1182.

Norton, A., Turk, G., Bacon, B., Gerth, J., and Sweeney, P. (1991). Animation of fracture by physical modeling. *The visual computer*, 7(4):210–219.

O'Brien, J. F. and Hodgins, J. K. (1999). Graphical modeling and animation of brittle fracture. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, page 137–146. ACM Press/Addison-Wesley Publishing Co.

OpenMP Architecture Review Board (2015). OpenMP 4.5.

Osher, S. and Fedkiw, R. P. (2001). Level set methods: an overview and some recent results. *Journal of Computational physics*, 169(2):463–502.

Paul, K., Zimmermann, C., Mandadapu, K. K., Hughes, T. J., Landis, C. M., and Sauer, R. A. (2020). An adaptive space-time phase field formulation for dynamic fracture of brittle shells based on lr nurbs. *Computational Mechanics*, 65(4):1039–1062.

Pfaff, T., Narain, R., De Joya, J. M., and O'Brien, J. F. (2014). Adaptive tearing and cracking of thin sheets. *ACM Transactions on Graphics (TOG)*, 33(4):1–9.

Piegl, L. and Tiller, W. (1996). *The NURBS book*. Springer Science & Business Media.

Proserpio, D., Ambati, M., De Lorenzis, L., and Kiendl, J. (2021). Phase-field simulation of ductile fracture in shell structures. *Computer Methods in Applied Mechanics and Engineering*, 385:114019.

Rabczuk, T., Areias, P., and Belytschko, T. (2007). A meshfree thin shell method for non-linear dynamic fracture. *International journal for numerical methods in engineering*, 72(5):524–548.

Rabczuk, T. and Ren, H. (2017). A peridynamics formulation for quasi-static fracture and contact in rock. *Engineering Geology*, 225:42–48.

Raghavachary, S. (2002). Fracture generation on polygonal meshes using voronoi polygons. SIGGRAPH '02, page 187. Association for Computing Machinery.

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. In Bertails-Descoubes, F., Coros, S., and Sueda, S., editors, *ACM/ Eurographics Symposium on Computer Animation*. ACM Siggraph.

Reddy, J. M. and Turkiyyah, G. M. (1995). Computation of 3d skeletons using a generalized delaunay triangulation technique. *Computer-Aided Design*, 27(9):677–694.

Richardson, C. L., Hegemann, J., Sifakis, E., Hellrung, J., and Teran, J. M. (2009). An xfem method for modeling geometrically elaborate crack propagation in brittle materials. *Institute for Computational and Applied Mathematics*, 90095:1555.

Richardson, C. L., Hegemann, J., Sifakis, E., Hellrung, J., and Teran, J. M. (2011). An xfem method for modeling geometrically elaborate crack propagation in brittle materials. *International Journal for Numerical Methods in Engineering*, 88(10):1042–1065.

Rycroft, C. (2009). Voro++: A three-dimensional voronoi cell library in c++. Technical report, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States).

Schvartzman, S. C. and Otaduy, M. A. (2014). Fracture animation based on high-dimensional voronoi diagrams. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '14, page 15–22. Association for Computing Machinery.

Sedighi, M., Yan, H., and Jivkov, A. P. (2022). Peridynamic modelling of clay erosion. *Géotechnique*, 72(6):510–521.

Sellán, S., Luong, J., Mattos Da Silva, L., Ramakrishnan, A., Yang, Y., and Jacobson, A. (2023). Breaking good: Fracture modes for realtime destruction. *ACM Transactions on Graphics*, 42(1):1–12.

Sherbrooke, E. C., Patrikalakis, N. M., and Brisson, E. (1995). Computation of the medial axis transform of 3-d polyhedra. In *Proceedings of the third ACM symposium on Solid modeling and applications*, pages 187–200.

Sifakis, E., Der, K. G., and Fedkiw, R. (2007). Arbitrary cutting of deformable tetrahedralized objects. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '07, page 73–80. Eurographics Association.

Sigalotti, L. D. G. and López, H. (2008). Adaptive kernel estimation and sph tensile instability. *Computers & Mathematics with Applications*, 55(1):23–50.

Sih, G. C. (1974). Strain-energy-density factor applied to mixed mode crack problems. *International Journal of fracture*, 10(3):305–321.

Steffen, M., Kirby, R. M., and Berzins, M. (2008). Analysis and reduction of quadrature errors in the material point method (mpm). *International journal for numerical methods in engineering*, 76(6):922–948.

Stomakhin, A., Schroeder, C., Chai, L., Teran, J., and Selle, A. (2013). A material point method for snow simulation. *ACM Trans. Graph.*, 32(4).

Su, J., Schroeder, C., and Fedkiw, R. (2009). Energy stability and fracture for frame rate rigid body simulations. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, page 155–164. Association for Computing Machinery.

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.

Sun, F., Wang, G., Zhang, L., Wang, R., Cao, T., and Ouyang, X. (2021). Material point method for the propagation of multiple branched cracks based on classical fracture mechanics. *Computer Methods in Applied Mechanics and Engineering*, 386:114116.

Tagliasacchi, A., Zhang, H., and Cohen-Or, D. (2009). Curve skeleton extraction from incomplete point cloud. In *ACM SIGGRAPH 2009 papers*, pages 1–9.

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

Thomaszewski, B., Wacker, M., and Straßer, W. (2005). A consistent bending model for cloth simulation with corotational subdivision finite elements.

Van der Meer, F. and Sluys, L. (2009). A phantom node formulation with mixed mode cohesive law for splitting in laminates. *International Journal of Fracture*, 158(2):107–124.

Vleugels, J. M., Overmars, M. H., et al. (1995). Approximating generalized voronoi diagrams in any dimension.

Wang, S., Ding, M., Gast, T. F., Zhu, L., Gagniere, S., Jiang, C., and Teran, J. M. (2019a). Simulation and visualization of ductile fracture with the material point method. *Proc. ACM Comput. Graph. Interact. Tech.*, 2(2).

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

Wang, Y., Bui, H. H., Nguyen, G. D., and Ranjith, P. (2019b). A new sph-based continuum framework with an embedded fracture process zone for modelling rock fracture. *International Journal of Solids and Structures*, 159:40–57.

Wang, Y., Tran, H. T., Nguyen, G. D., Ranjith, P. G., and Bui, H. H. (2020b). Simulation of mixed-mode fracture using sph particles with an embedded fracture process zone. *International Journal for Numerical and Analytical Methods in Geomechanics*, 44(10):1417–1445.

Wicke, M., Ritchie, D., Klingner, B. M., Burke, S., Shewchuk, J. R., and O'Brien, J. F. (2010). Dynamic local remeshing for elastoplastic simulation. In *ACM SIGGRAPH 2010 Papers*, SIGGRAPH '10, pages 49:1–49:11. ACM.

Wineman, A. (2005). Some results for generalized neo-hookean elastic materials. *International Journal of Non-Linear Mechanics*, 40(2-3):271–279.

Wojtan, C. and Turk, G. (2008). Fast viscoelastic behavior with thin features. In *ACM SIGGRAPH 2008 papers*, pages 1–8.

Wolper, J., Chen, Y., Li, M., Fang, Y., Qu, Z., Lu, J., Cheng, M., and Jiang, C. (2020). Anisompm: Animating anisotropic damage mechanics. *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 and Graphics*, 69:131 – 139.

Xue, T., Adriaenssens, S., and Mao, S. (2021). Mapped phase field method for brittle fracture. *Computer Methods in Applied Mechanics and Engineering*, 385:114046.

Yue, Y., Smith, B., Batty, C., Zheng, C., and Grinspun, E. (2015). Continuum foam: A material point method for shear-dependent flows. *ACM Trans. Graph.*, 34(5).

Zhang, Z., Qiu, Y., Hu, Z., Ye, H., Zhang, H., and Zheng, Y. (2022). Explicit phase-field total lagrangian material point method for the dynamic fracture of hyperelastic materials. *Computer Methods in Applied Mechanics and Engineering*, 398:115234.

Zhao, Z., Huang, K., Li, C., Wang, C., and Qin, H. (2020). A novel plastic phase-field method for ductile fracture with gpu optimization. In *Computer Graphics Forum*, volume 39, pages 105–117. Wiley Online Library.

Zhou, Q., Grinspun, E., Zorin, D., and Jacobson, A. (2016). Mesh arrangements for solid geometry. *ACM Trans. Graph*, 35(4).

Zhu, Y., Bridson, R., and Greif, C. (2015). Simulating rigid body fracture with surface meshes. *ACM Trans. Graph.*, 34(4):150:1–150:11.