Fast Remeshing-Free Methods for Complex Cutting and Fracture Simulation

Submitted in partial fulfillment of the requirements
of the degree of

Doctor of Philosophy

by

Avirup Mandal
(Roll No. 163070008)

Supervisors:

Prof. Parag Chaudhuri
Prof. Subhasis Chaudhuri

Department of Electrical Engineering
INDIAN INSTITUTE OF TECHNOLOGY BOMBAY
2023

Only I have left to say,

More is thy due than more than all can pay.[1]

Dedicated to my beloved parents — Satyanarayan Mandal and Anuva Mandal.

[1] William Shakespeare, Macbeth

Approval Sheet

This thesis/~~dissertation/report~~ entitled **Fast Remeshing-Free Methods for Complex Cutting and Fracture Simulation** by Mr. Avirup Mandal is approved for the degree of **Doctor of Philosophy (PhD)**.

Examiners

Prof. Subodh Kumar, Department of Computer Science and Engineering, IIT Delhi

Prof. Abhishek Gupta, Department of Mechanical Engineering, IIT Bombay

Supervisor (s)

Prof. Subhasis Chaudhuri, Department of Electrical Engineering, IIT Bombay

Prof. Parag Chaudhuri, Department of Computer Science and Engineering, IIT Bombay

Chairperson

Prof. Neela Nataraj, Department of Mathematics, IIT Bombay

Date : 25-7-23

Place : Mumbai

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Date: 25-07-2023

Avirup Mandal

Roll No. 163070008

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY, INDIA

CERTIFICATE OF COURSE WORK

This is to certify that **Avirup Mandal** (Roll No. 163070008) was admitted to the candidacy of M.Tech. + Ph.D. dual degree on 14 July 2016, after successfully completing all the courses required for the M.Tech. + Ph.D. dual programme. The details of the course work done are given below.

S.No	Course Code	Course Name	Credits
1	EE 601	Statistical Signal Analysis	6
2	EE 635	Applied Linear Algebra	6
3	EE 603	Digital Signal Processing and its Applications	6
4	EE 703	Digital Message Transmission	6
5	CS 725	Foundations of Machine Learning	6
6	EE 608	Adaptive Signal Processing	6
7	EE 702	Computer Vision	6
8	EE 708	Information Theory & Coding	6
9	IE 601	Optimization Techniques	8
10	CE 712	Digital Image Processing of Remotely Sensed Data	6
11	EE 771	Recent Topics in Analytical Signal Processing	6
12	IE 605	Engineering Statistics	8
13	CS 775	Advanced Computer Graphics	6
14	EE 694	Seminar	4
15	EE 797	MTP I Stage Project	48
16	CS 675	Computer Graphics	AU
17	IE 643	Deep Learning - Theory and Practice	AU
18	EE 734	Advanced Probability and Random Processes for Engineers	AU
19	EE 610	Image Processing	AU
20	CS 736	Medical Image Computing	AU
21	EE 792	Communication Skills - II	PP
22	HS 791	Communication Skills - I	PP
23	GC 101	Gender in the workplace	PP
		Total Credits	**134**

TO MY DEAR READER

First my fear; then my curtsy; last my speech. My fear is your displeasure; my curt'sy, my duty; and my speech, to beg your pardons. If you look for a good speech now, you undo me: for what I have to say is of mine own making; and what indeed I should say will, I doubt, prove mine own marring. But to the purpose, and so to the venture.

William Shakespeare, The Second Part of Henry the Fourth

Abstract

Simulating complex cuts and fractures robustly and accurately benefits a broad spectrum of researchers from a wide variety of disciplines ranging from rendering realistic and spectacular animations in computer graphics and interactive techniques to material strength analysis in industrial design and mechanical engineering. We start our thesis by proposing an Improved stable extended Finite Element (Is-XFEM) method to simulate robust cutting at an interactive rate. However, the Is-XFEM solution suffers from increasing computational costs due to the introduction of additional degrees of freedom in the system dynamics. To tackle that, we develop a graph-based Finite Element Method (FEM) model that reformulates the hyper-elastic strain energy for fracture simulation and thus adds negligible computational overhead over a regular FEM. Our algorithm models fracture on the graph induced in a volumetric mesh with tetrahedral elements. We relabel the edges of the graph using a computed damage variable to initialize and propagate the fracture. Following that, we extend graph-based FEM to simulate dynamic fracture in anisotropic materials. We further enhance this model by developing novel probabilistic damage mechanics for modelling materials with impurities using a random graph-based formulation. We demonstrate how this formulation can be used by artists for directing and controlling fracture. Finally, we combine graph-based FEM with a Galerkin multigrid method to run fracture and cutting simulation at a real-time, interactive rate even for high-resolution meshes. We thoroughly verify all of our proposed algorithms and techniques by performing extensive standard laboratory experiments, comparing to state-of-the-art methods, presenting theoretical analysis, simulating multiple demos for a breath of materials and conducting user studies to evaluate the functionality and appeal of our methods.

Contents

List of Tables

List of Figures

List of Abbreviations

ANOVA	Analysis of Variance
AWGN	Additive white Gaussian noise
BEM	Boundary Element Method
CMOD	Crack Mouth Opening Displacement
CUDA	Compute Unified Device Architecture
FEM	Finite Element Method
GPU	Graphics Processing Unit
Gra-FEA	Graph-based Finite Element Analysis
GUI	Graphical User Interface
Is-XFEM	Improved stable Extended Finite Element Method
MRF	Markov Random Field
SOTA	State of The Art
XFEM	Extended Finite Element Method

List of Symbols

$\mathbf{\Psi}$	Hyper-elastic energy density
$\mathbf{u}$	Displacement field
$\mathbf{F}$	Deformation gradient
$\mathbf{F}_e$	Elastic part of deformation gradient
$\mathbf{F}_p$	Plastic part of deformation gradient
$\mathbf{R}$	Rotation matrix of polar decomposition
$\mathbf{S}$	Stretch matrix of polar decomposition
$\mathbf{C}$	Right Cauchy-Green tensor
$\mathbf{E}_{FS}$	Lagrangian finite strain tensor
$\mathbf{Y}$	Young's modulus
ν	Poission's ratio
$\mathbf{E}$	Elasticity tensor
ε	Cauchy linear strain tensor
σ	Linear stess tensor
σ_{thres}	Stess threshold
$\boldsymbol{\sigma}_c^e$	Cartesian components of stress tensor of a tetrahedron
$\boldsymbol{\sigma}_d^e$	Normal stress tensor along edges of a tetrahedron
$\mathbf{P}$	Piola-Kirchhoff stress tensor
I_C, II_C, III_C/J	Isotropic invariants
IV_C, V_C	Anisotropic invariants
I_4, I_5	Lower-order anisotropic invariants

μ, λ	Lamé parameters
n_v	Total number of nodes in object mesh
n_{tet}	Total number of tetrahedra in object mesh
$\mathbf{N}_i$	Shape function of a tetrahedron
ξ	Position of any point inside a tetrahedron
$\mathbf{m}_e$	Mass matrix of a tetrahedron
$\mathbf{h}_e$	Damping matrix of a tetrahedron
$\mathbf{k}_e$	Tangent stiffness matrix of a tetrahedron
$\mathbf{f}_e^{int}$	Internal force vector of a tetrahedron
$\mathbf{f}_e^{ext}$	External force vector of a tetrahedron
$\mathbf{t}_e$	Reference body force of a tetrahedron
$\mathbf{s}_e$	Specific elastic element force of a tetrahedron
$\mathbf{I}_n$	Identity matrix of size $n \times n$
ρ_0	Rest density
V_e	Volume of a tetrahedron
$\boldsymbol{\zeta}$	Damaged edge variable matrix
H_s	Heaviside function
Φ^j	Signed distance function of a point w.r.t. cut j
ψ_i^j	Enrichment of node i for cut j
γ	Linear interpolant of Heaviside enrichment
χ	Characteristic function of i^{th} subdomain
$w_{i,j}$	Weights at the quadrature points i of a tetrahedron for cut j
$\mathbf{T}_p$	Pre-conditioner matrix for Conjugate Gradient solver
$v_{i,j}$	Volume ratio of node i due to cut j
$d_{i,j}$	Distance between node i and node j
$\hat{\mathbf{d}}_{i,j}$	Unit vector from node i and node j

$\mathbf{T}$	Transformation matrix
$\mathbf{L}_d^e$	Original length matrix of the edges of a tetrahedron
$\mathbf{l}_d^e$	Increased length matrix of the edges of a tetrahedron
χ^e	Degradation function due to fracture
$\mathscr{M}_c$	Computational mesh
$\mathscr{M}_v$	Visualization mesh
G	Graph
d_i	Degree of node i
$p_{i,j}$	Probability of breaking an edge between node i and node j
h	Strictly increasing function
$\mathscr{N}$	Gaussian distribution
α_k	Impurity value at node k
$\mathbf{\Omega}$	Mesh
$\mathbf{\Omega}_l$	Mesh at l^{th} level grid hierarchy for Galerkin multigrid
$\mathbf{U}_l/\mathbf{R}_l/\mathbf{P}_l$	Projection matrix for Galerkin multigrid at level l
$\mathbf{r}_l$	Residual vector for Galerkin multigrid at level l
$\mathbf{e}_l$	Error vector for Galerkin multigrid at level l
$\mathbf{A}$	System matrix of a simulation
$\mathbf{A}_l$	System matrix for Galerkin multigrid at level l
$\mathbf{I}_p^{VF}$	Penalty force between vertex and face
$\mathbf{I}_p^{EE}$	Penalty force between two edges
G_d	Damping kernel
$:$	Generalized dot product between two matrices

Chapter 1

Introduction

1.1 Motivation

Nearly every object, natural or manmade, can undergo geometric alteration. This is an all-pervading phenomenon we are all accustomed to. Generally, two kinds of geometric changes can be found in abundance around us — deformation and fracture. The deformation of an object can be purely elastic or may contain plastic characteristics. Deformation can produce a significant geometric variation in the object mesh but it does not introduce any new discontinuity inside the mesh.

A fracture occurs when new discontinuities are initiated inside a deforming object. To be more specific, when the deformation limit of an object is reached and its material integrity fails, it fractures. Every day we come across several instances of fracture, ranging from the shattering of a brittle statue (see Figure 1.2 (left)) to the tearing of a soft and ductile loaf of bread (see Figure 1.1 (left)). Many dynamic fracture algorithms are developed in structural mechanics

and computer graphics to recreate such real-world phenomena. However, simulating intricate fracture patterns and crack propagation often requires complex mathematical formulation. Incorporating fractures for both brittle and ductile material in a single framework needs even more sophisticated analysis. A material is brittle if, when subjected to stress, it fractures with little elastic deformation and without significant plastic deformation. On the contrary, ductile material goes through significant deformation, elastic or plastic, before getting fractured.

Moreover, many real-world materials have anisotropic fibres along certain directions; this makes them stiffer in these preferred directions and softer in others. This imbalance of stiffness also influences crack propagation during fracture. In addition, the state or condition of the material affects the fracture as well. While some materials remain pure, others get corrupted over time due to environmental and human interventions, which in turn impose uncertainties on the fracture criteria. Because of this multi-pronged complex nature of the fracture, it, to date, remains one of the most challenging and important research topics in both structural mechanics and computer graphics.

Due to its ubiquitous nature, fracture simulation algorithms have massive applications in animation, gaming and VFX industries to emulate real-world phenomena. The entertainment world is a huge consumer of these techniques to recreate different kinds of scenarios e.g., ranging from pre-historic to futuristic societies, fantasy animals, weapons etc. Moreover, video games that involve massive fighting sequences like Counter-Strike [CounterStrike, 2023], Call of Duty [CallOfDuty, 2023] often use these technologies. Moreover, virtual simulators are often used to train and educate professionals in applications that involve high risk and high cost. Some of these implementations include simulator for pilot training, surgical framework for training doctors, war scenario simulator to test the sustainability of fighter jet airframes and frameworks to evaluate the material integrity of large structures like water dams. Computer graphics techniques are extensively used in scientific visualization also such as interactive pedagogical applications, high-spectral satellite imagery, data visualization and geological study for detecting glacier cracks to name a few.

Figure 1.1: Left: A piece of torn bread (Image courtesy: Shutterstock). Right: Simulated fracture using our proposed algorithm.

Figure 1.2: Left: A shattered replica of David (Image courtesy: LA Times). Right: Brittle fracture of a blue jade armadillo model.

1.2 Problem Statement

Our objective is to *develop fast, efficient and robust algorithms for cutting and fracture simulation of ductile and brittle materials* (Figure 1.1(right) & Figure 1.2(right)). To that end, we propose four remeshing-free fracture simulation methods in this thesis.

1. Improved stable eXtended Finite Element Method (Is-XFEM) [Chapter 3 & [Mandal et al., 2022a] & [Mandal. et al., 2023]].

2. Graph-based Finite Element Method [Chapter 4 & [Mandal et al., 2023]].

3. Random Graph-based Finite Element Method [Chapter 5 & [Mandal et al., 2022b]].

4. Graph-based Finite Element Method with Multigrid Galerkin [Chapter 6].

Techniques based on the remeshing-free eXtended Finite Element Method (XFEM) [Koschier et al., 2017] [Chitalu et al., 2020] are considered the state-of-the-art (SOTA) algorithms for mesh-based fracture simulation. XFEM-based algorithms can accurately predict crack propagation and are faster than the remeshing-based fracture simulation methods [O'Brien and Hodgins, 1999] [O'Brien et al., 2002] [Koschier et al., 2015] that require extensive remeshing at each time step. However, XFEM has several limitations like instability due to skewed volume elements and poor scalability while incorporating a large number of cracks. Next, we explain how our proposed fracture simulation algorithms address and solve each of these drawbacks while retaining the other key characteristics desired in a fracture simulation algorithm.

1.2.1 Key Characteristics of Our Fracture Simulation Algorithms

1.2.2 Accuracy

Fracture or cutting simulation algorithms can be broadly classified into two categories: geometry-based and physics-based. Even though geometry-based approaches are faster to simulate, physics-based methods are more accurate in simulating real-world material behaviours. All our proposed algorithms for fracture simulation are built on the Finite Element Method (FEM) and thus, are physically accurate. We evaluate the accuracy of Graph-based FEM in Section 4.5.5 by comparing it with real-world structural mechanics experiments.

1.2.3 Stable

Fracture comes with inherent stability issues as it produces multiple disjoint segments, consequently, increasing the overall entropy of the system. The stability of XFEM remains an open challenge for researchers to date. In Section 3.3.2, we demonstrate how Is-XFEM is comparatively more stable than XFEM. Graph-based FEM fares even better in terms of stability (see Section 4.5.6) at larger time steps.

1.2.4 Scalable

Each new crack in an object mesh introduces new degrees of freedom (dof), increasing the system matrix's size and computational complexity. Thus, Is-XFEM is particularly dependent on the number of crack surfaces and not scalable to applications which require extreme disintegration of an object mesh. It is one of the major limitations for both XFEM and Is-XFEM as explained in Section 3.4 and Section 2.2.2. Graph-based FEM overcomes this limitation by reformulating internal hyper-elastic strain energy for fracture and thus, avoids remeshing. Unlike Is-XFEM, it is capable of incorporating a large number of cracks without adding extra computational overhead (see Section 4.5.6).

1.2.5 Efficiency

The computational complexity of FEM is $O(NW^2)$ where N and W denote the number of nodes and bandwidth of the stiffness matrix respectively [Farmaga et al., 2011]. However, different applications demand different update rates. Keeping that in mind, we categorize the fracture simulation algorithms into two broad areas.

- Algorithms used for rendering realistic animations and visual effects, which do not require real-time processing. The final output of these methods tends to have high visual fidelity.
- Algorithms utilized in gaming and VR/AR applications which require real-time, interactive processing. In this case, for the sake of fast real-time simulation, visual fidelity and accuracy are compromised to a certain extent.

When parallelized on the GPU, Is-XFEM runs at an interactive rate for low-resolution meshes ($\sim$ 2-4k tetrahedra) with a few numbers of cuts ($\sim$ 5-10). However, it ceases to be interactive for high-resolution meshes or even when a low-resolution mesh contains a large number of cracks due to the scalability issue discussed earlier. Graph-based FEM focuses on rendering realistic fracture animation. Still, it runs much faster than state-of-the-art XFEM-based algorithms [Koschier et al., 2017] [Chitalu et al., 2020]. Our graph-based FEM combined with

Multigrid Galerkin (Chapter 6) is capable of running at an interactive rate even for very high-resolution meshes ($\sim$ 200-300k tetrahedra) despite containing numerous cracks ($\sim$ 15-20k).

1.2.6 Controllable

Fracture is a random phenomenon. Generally, cracks are initiated from the high-stress regions depending on a user-defined threshold. However, for a specific visual effect or animation, an artist may want to direct or control the pattern of crack propagation. But, there exists no robust yet easy algorithm to precisely predict the regions of crack initiation and then their propagation in the object mesh. In Chapter 5, we combine our Graph-based FEM method with a random graph technique to give the user/artist specific control over the initiation and propagation of crack surfaces (see Section 5.5.3).

1.3 Challenges

Listed below are the key challenges that we come across while developing our fracture simulation algorithms.

1.3.1 Multidisciplinary Subject

Fracture simulation spans multiple disciplines; from physics to structural mechanics, mechanical engineering to computer graphics. However, we believe that the research communities in mechanical and civil engineering contribute to most of the engineering-related fracture literature. But the primary focus of these two communities remains on simulating localized static cracks, validating the thermodynamical perspective of fracture in terms of free energy and evaluating mechanical properties for materials used in different structures, e.g., water dams and vehicle bodies. On the other hand, interactive frameworks require full dynamic simulations with distributed crack openings and realistic visualization. While mechanical and civil engineering communities strive for increasingly more accurate physical simulation at the cost of

extremely high simulation time, the graphics community can sometimes demand a trade-off between physical accuracy and computational complexity as long as visual plausibility is maintained.

Moreover, the mathematics of fracture simulation is different from what is regularly encountered in electrical and computer science disciplines. The mathematical theory of fracture requires knowledge of advanced tensor calculus, discrete differential geometry and random graph theory.

Thus, drawing connections from different disciplines for different aspects of fracture simulation necessitates a steep learning curve.

1.3.2 High Entropy

Fracture is an energy dissipative process which rapidly adds new degrees of freedom. Thus, it is hard to control the different output characteristics of fracture, e.g., the amount of debris, the microstructure of the fracture surface, stability of the simulation, volume and momentum conservation. Tuning multiple highly sensitive simulation parameters to get the desired output requires experience and a good intuitive understanding of the physical phenomena.

1.3.3 No Open-Source Code-base

Another significant challenge that we faced is that in computer graphics there is almost no open-source code base available for mesh-based fracture simulation, specifically for XFEM and remeshing-based algorithms. It makes comparison studies with state-of-the-art methods more challenging and time-consuming to perform. This brings us to the next challenge i.e., validation with real-world phenomena.

1.3.4 Validation with Real-World Phenomena

To evaluate the quantitative and qualitative accuracy of fracture simulation, real-world benchmark laboratory experiment setups are reproduced in the simulation environment. However, building the virtual equivalent of a real-world experiment requires careful mapping of multiple material parameters e.g., Young's modulus, Poisson's ratio, plasticity and fracture threshold of stress to the simulation environment. Moreover, these physical parameters of the real-world materials have to be scaled and normalized appropriately for the virtual simulator, which is a complex and onerous process.

1.4 Structure of the Thesis

Here is a brief overview of the chapters of the thesis.

In **Chapter 2**, we present a background study of the relevant literature and existing techniques related to our work.

Following that, in **Chapter 3** we develop an Improved stable XFEM [Mandal et al., 2022a] for accurately simulating cut openings in an object mesh. Is-XFEM is an improvement over the existing XFEM-based cutting algorithm [Koschier et al., 2017] in terms of stability. Moreover, as Is-XFEM does not require the remeshing of the original object mesh, it is much faster than the remeshing-based methods as well. However, even though Is-XFEM is more stable compared to regular XFEM [Wu and Li, 2015], a few stability issues persist. More importantly, Is-XFEM fares poorly in terms of scalability. With the introduction of new cracks, the system matrix size of Is-XFEM keeps getting increased, leading to higher computation time.

These limitations of Is-XFEM motivate us to develop a more stable and scalable fracture simulation algorithm. Our next work, outlined in **Chapter 4**, proposes a novel remeshing-free Graph-based Finite Element Method (Graph-based FEM) for fracture simulation. In Graph-based FEM [Mandal et al., 2023], when fracture sets in, the hyper-elastic strain energy density of the element is reformulated to simulate the independent movement of the disjoint pieces. Only the outer surface mesh is remeshed for visualization. However, the volumetric mesh

on which the dynamic physics simulation is performed remains the same as it initially was. Thus, unlike Is-XFEM, the system matrix size remains constant throughout the simulation for Graph-based FEM, irrespective of the number of discontinuities introduced due to fracture. This makes the Graph-based FEM highly scalable to severe geometric changes of object mesh consisting of a large number of crack planes. Moreover, as Graph-based FEM does not require any remeshing, it is both as accurate and stable as the underlying FEM formulation. Thus, Graph-based FEM retains the good characteristics of both FEM and Is-XFEM. Using, graph-based FEM we simulate the fracture of a wide range of materials such as brittle & ductile, isotropic & anisotropic. However, our Graph-based FEM has a few limitations. It does not take into account the effects of material impurity while simulating fracture. But impurity affects the fracture simulation significantly. Material impurities impose probabilistic threshold criteria on fracture threshold rather than a deterministic one.

To address these drawbacks, in **Chapter 5** we extend our original Graph-based FEM with random graph formulation. To model material impurity, a random graph technique is augmented with our regular Graph-based FEM [Mandal et al., 2022b]. The random graph method models the impurity using a probabilistic approach. This, in turn, makes an object weaker or stronger in different places depending on the deposition of impurity. Building on the random graph-based impurity modelling we develop an interactive framework using which an artist can embed specific fracture patterns on an object mesh, thus, giving the user more control over crack propagation. This framework is very useful for those applications which require to design of specific fracture patterns e.g., fracture of crystalloid materials. However, even though our Graph-based FEM method along with random graph formulation is much faster than the existing techniques, it does not run at an interactive rate for high-resolution meshes.

In **Chapter 6** we focus on improving Graph-based FEM to run at an interactive rate. To that end, our graph-based FEM is augmented with a Galerkin Multigrid method [Xian et al., 2019]. Galerkin Multigrid method is capable of running high-resolution meshes at an interactive rate and Graph-based FEM adds negligible extra computational overhead for fracture simulation on top of the underlying method. Thus, these two methods together are capable of rendering fracture and cutting at an interactive rate. To validate our claim, we develop a virtual sculpting application using these two methods together.

Finally, we conclude the thesis by giving an overview of all these techniques and a few

avenues for interesting future works.

Chapter 2

Background

2.1 Physics-based Simulation of Deformable Objects

Solid objects that we can touch, feel or see, are of two kinds by and large — rigid and deformable. When an external force is applied, rigid bodies do not exhibit any visible change in shape unless a fracture occurs. Deformable bodies, on the other hand, may contort into a completely different configuration without breaking.

Algorithms for simulating deformation using mathematical models can be broadly classified into two categories — geometric and physics-based. The crucial difference that sets the two methods apart lies in their response to the external force. In the case of the geometric approach, the object mesh does not develop any internal/reaction force to the external stimuli. Though the mathematical model is simpler, this method does not exhibit physically accurate behaviour. Among the existing geometric methods, mass-spring-based polygonal mesh deformation is explored in [Gunn, 2006] [Dachille et al., 1999]. Object surface manipulation using B-splines is

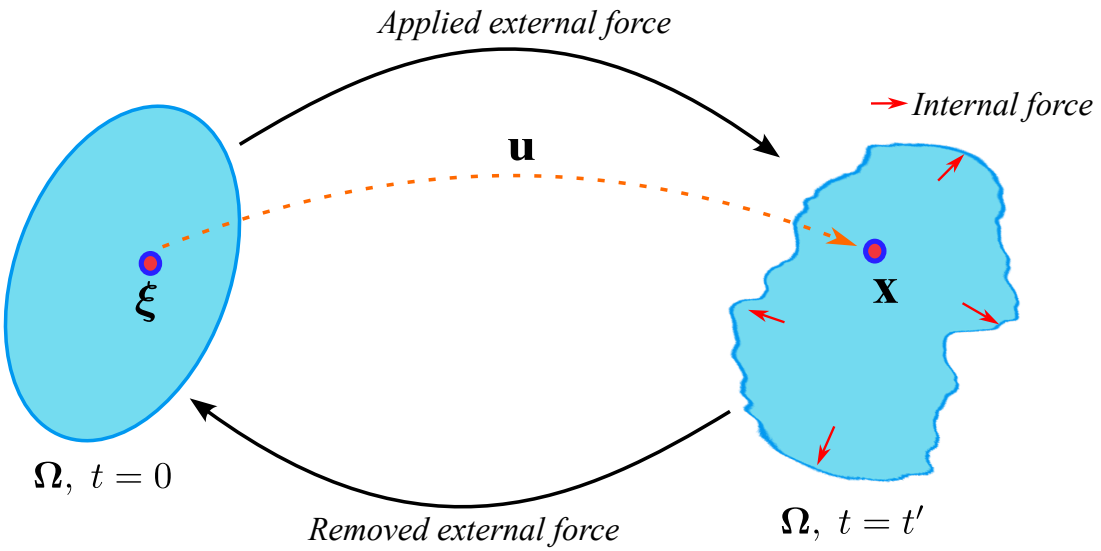


Figure 2.1: Deformable object: A displacement function maps a point on the object in material coordinates on the left and to world coordinates on the right.

presented in [Gao and Gibson, 2006]. [Jagnow and Dorsey, 2002] developed a method to perform virtual sculpting using displacement maps. However, both the B-spline and displacement map-based methods are limited to the interaction with 2D surfaces embedded into 3D space and do not explore the deformation of volumetric 3D objects.

On the other hand, physics-based deformation approaches use a mathematically inspired *hyper-elasticity* energy model. As depicted in Figure 2.1 when an object mesh deviates from its rest shape due to external load, hyper-elastic energy, $\mathbf{\Psi}$, gets developed which in turn produces internal force. This internal force then makes the object bounce back to its initial rest shape once the load is removed. This is the crux of a physically accurate hyper-elastic deformation model. Some objects do not return back to their rest shape even when the external force is removed. This is due to plasticity.

2.1.1 Hyper-elastic Materials

An elastic material is a linear material. It implies that stress varies linearly with respect to strain (see Figure 2.2). Objects that exhibit very small deformation such as paper and wood can be classified as elastic materials.

A hyper-elastic material [Erleben et al., 2005a], unlike an elastic material, is designed to

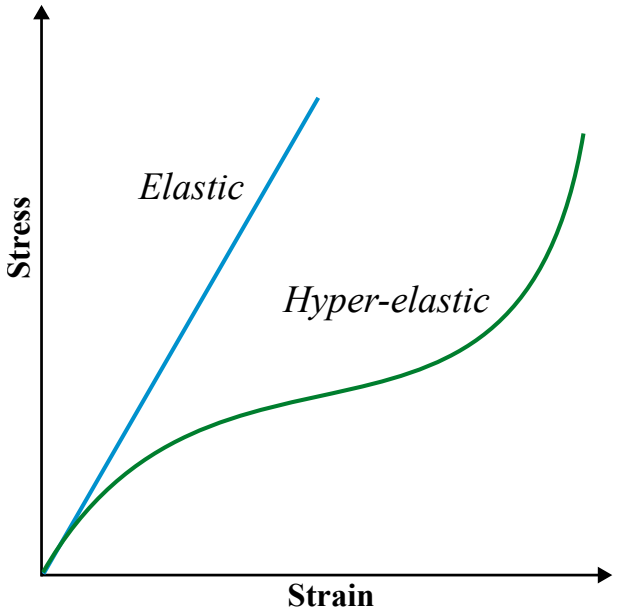


Figure 2.2: Stress-strain curve for elastic and hyper-elastic material.

model materials in which the deformation can be extremely large. A hyper-elastic material is a type of constitutive model for which the stress–strain relationship deviates from the linear form. The stress–strain relationship is usually derived from a strain energy density function [Kim and Eberle, 2022] [Sifakis and Barbic, 2012]. In these materials, the applied force induces displacement which is encoded by the strain energy density function. Hyper-elastic strain energies can be expressed using different invariants. Before defining these invariants, we first define some quantities frequently used in hyper-elasticity modelling.

2.1.1.1 Important Quantities

Let's consider a mesh with n_v node in a 3D space.

- $\underline{\mathbf{u}}$ — Displacement field of a deforming object compared to its rest shape, i.e., it captures the difference between current and rest positions for each point of the mesh. During FEM computation, $\mathbf{u}$ is discretized into a single vector of size $3n_v$.
- $\underline{\mathbf{F} = \mathbf{I} + \nabla \mathbf{u}}$ — Deformation gradient i.e., it a function that maps an object from its undeformed configuration to deformed configuration and $\mathbf{I}$ denotes identity matrix of size $n_v \times n_v$.
- $\underline{\mathbf{C} = \mathbf{F}^T \mathbf{F}}$ — Right Cauchy-Green deformation tensor. It can be physically understood as

the change in the square of the relative position vector in an undeformed body to that of a deformed one.

- $\underline{\mathbf{E}_{FS} = \frac{1}{2}\left(\mathbf{F}^T\mathbf{F} - \mathbf{I}\right)}$ — Lagrangian finite strain tensor. It is a higher-order strain measure that denotes the gradients in terms of the original configuration.

- $\underline{\varepsilon = \frac{1}{2}\left(\mathbf{F} + \mathbf{F}^T\right) - \mathbf{I}}$ — Cauchy linear strain tensor. It is a lower-order approximation of the Lagrangian finite strain tensor.

- $\underline{\mathbf{P}(\mathbf{F}) = \frac{\partial \boldsymbol{\Psi}}{\partial \mathbf{F}}}$ — Piola-Kirchhoff stress tensor.

We now define the invariants for hyper-elastic energy density and explain their physical significance.

2.1.1.2 Isotropic Invariants

- $\underline{I_C = \text{trace}(\mathbf{C})}$ — Relative change in the lengths of 1D elements of mesh, e.g., edges of a tetrahedron.

- $\underline{II_C = \mathbf{C} : \mathbf{C}}$ — Relative change in the areas of 2D elements of mesh, e.g., faces of a tetrahedron. The double-contraction, denotes with a ':', is a generalized version of the dot product for two matrices. An example is as follows

$$\mathbf{A} : \mathbf{B} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = a_{11}b_{11} + a_{12}b_{12} + a_{21}b_{21} + a_{22}b_{22}$$

- $\underline{III_C \equiv J = \det(\mathbf{C})}$ — Relative change in the volume of a FEM mesh element.

2.1.1.3 Anisotropic Invariants

- $\underline{IV_C = \mathbf{a}^T\mathbf{C}\mathbf{b}}$ — Represents the square of the fibre stretch. The parameters **a** and **b** are anisotropic fibre directions.

- $\underline{V_C = \mathbf{a}^T \mathbf{C}^T \mathbf{C}\mathbf{b}}$ — Represents the fourth order of the fibre stretch.

  Both IV_C and V_C invariants are based on $\mathbf{C}$, which square away the sign information of deformation gradient $\mathbf{F}$. The negative sign of $\det(\mathbf{F})$ denotes the inversion of an element and vice-versa. Thus both these invariants have sign-discarding, inversion-oblivious problems.

- $\underline{I_4 = \mathbf{a}^T \mathbf{S}\mathbf{a}}$ — A lower order sign preserving variant of IV_C. $\mathbf{S}$ is a stretch matrix obtained from the polar decomposition of $\mathbf{F}$.

- $\underline{I_5 = \mathbf{a}^T \mathbf{C}\mathbf{a}}$ — Same as IV_C with $a = b$.

2.1.1.4 Isotropic Strain Energy

Isotropic materials compress or stretch equally in all directions irrespective of the direction of applied force. A few popular isotropic hyper-elastic energy densities are mentioned here.

$$
\begin{aligned}
\boldsymbol{\Psi}_{StVK} &= \frac{\lambda}{8}(I_C - 3)^2 + \frac{\mu}{4}(II_C - I_C + 3), \text{ Saint–Venant Kirchhoff [Sin et al., 2013]} \\
\boldsymbol{\Psi}_{neo} &= \frac{\mu}{2}(I_C - 3) - \mu \log J + \frac{\lambda}{2}(\log J)^2, \text{ Neo-Hookean [Sin et al., 2013]} \\
\boldsymbol{\Psi}_{sneo} &= \frac{\mu}{2}(I_C - 3) + \frac{\lambda}{2}\left(J - 1 - \frac{3\mu}{4\lambda}\right)^2 - \frac{\mu}{2}\log(I_C + 1), \text{ Stable Neo-Hookean [Smith et al., 2018]}
\end{aligned}
\tag{2.1}
$$

where μ and λ are Lamé parameters.

2.1.1.5 Anisotropic Strain Energy

Many real-world materials have bundles of fibres in some preferred directions, making them stiffer in those directions and softer in others. As illustrated in Figure 2.3, when stretched or compressed, these materials resist more in the stiffer directions compared to softer directions. Xu et al. [2015] proposed a novel algorithm to model anisotropy using principle stretches of hyper-elastic strain energy density. Kim et al. [2019] developed a new anisotropic hyper-elastic energy density which is inversion-aware and contains no singularity. Inversion-aware implies that the energy is able to detect the phenomena such as an object getting inverted like a reflection or any vertex of an element passing through the outer surface of the element and coming out on

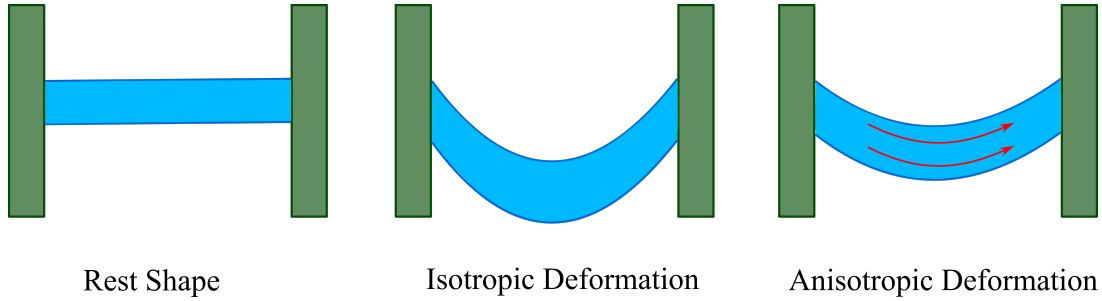


Figure 2.3: Anisotropic fibres make an object more resistant to deformation along one or more specific directions.

the other side. Moreover, the energy does not produce any singular or non-deterministic value of Piola-Kirchhoff stress.

$$\mathbf{\Psi}_{AA} = \frac{\mu}{2}\left(\sqrt{I_5} - \Pi(I_4)\right)^2 \tag{2.2}$$

where $I_4 = \mathbf{a}^T\mathbf{S}\mathbf{a}$, $I_5 = \mathbf{a}^T\mathbf{C}\mathbf{a}$ and Π is the signum function. The stretch matrix $\mathbf{S}$ is obtained from the polar decomposition of deformation gradient $\mathbf{F} = \mathbf{RS}$. More details on the invariants I_4 and I_5 are given later. In practice, we add the anisotropic hyper-elastic strain energy along with isotropic hyper-elastic strain energy and solve the system together to simulate anisotropy.

2.1.1.6 Plasticity

Plastic deformation occurs when an object undergoes a permanent change in shape without breaking as depicted in Figure 2.4. Thus, a deformed object retains a fraction of its topological change when even the external load is removed. Large plastic flow is rendered in works by Bargteil et al. [2007], Irving et al. [2004] and Stomakhin et al. [2012]. We follow the multiplicative plasticity model [Bargteil et al., 2007] for our work. In this model, the deformation gradient is split into an elastic and a plastic part

$$\mathbf{F} = \mathbf{F}_e\mathbf{F}_p \tag{2.3}$$

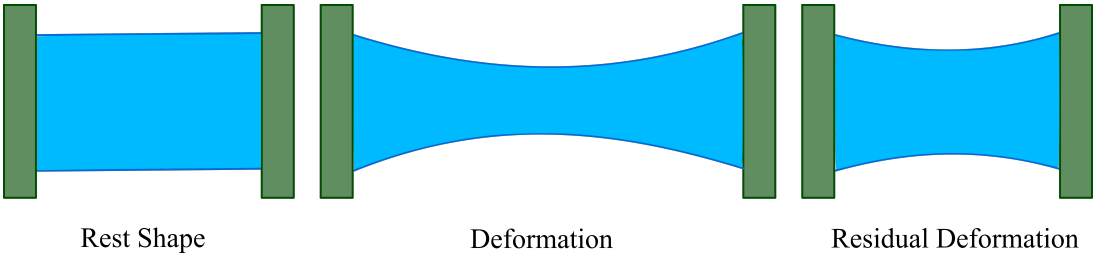


Figure 2.4: A plastic material does not return back to its rest shape even after the load is released.

where forcing $\det(\mathbf{F}_p) = 1$, the volume of the element is preserved. Starting from an initial identity matrix, $\mathbf{F}_p$ is updated as follows

$$\mathbf{F}_p \longleftarrow \mathbf{F}_p \cdot \mathbf{V} \left(\det(\mathbf{\Sigma})^{-\frac{1}{3}} \mathbf{\Sigma} \right)^{\beta} \mathbf{V}^T \tag{2.4}$$

where $\mathbf{U\Sigma V}^T$ is SVD of $\mathbf{F}_e$. The exponent β is an function of current stress ($\boldsymbol{\sigma}^e$), yield stress (σ^e_{thres}), flow rate (ν) and hardening parameter (δ, K)

$$\beta = \text{clamp} \left\{ \frac{\nu \left(||\boldsymbol{\sigma}^e||_2 - \sigma^e_{thres} - K\delta \right)}{||\boldsymbol{\sigma}^e||_2}, 0 \ldots 1 \right\} \tag{2.5}$$

where ν and K are user-defined parameters. The term $K\delta$ determines the work hardening or softening. The term δ is initialized with value zero and incremented in each time step by $\delta \longleftarrow \delta + \Delta t ||\boldsymbol{\sigma}^e||_2$.

2.1.2 Finite Element Method-based Deformation

There are many existing techniques to model physically accurate deformation, e.g. Finite Element Method, Material Point Method, Smooth Particle Hydrodynamics and Boundary Element Methods to name a few. However, the basic principles of all these methods remain the same while their method of discretization of the simulation space varies. In the case of mesh-based methods like FEM and BEM, the mesh is discretized using geometric elements, e.g. triangle/square in 2D, tetrahedron/hexahedron in 3D. On the contrary, meshless methods like MPM and SPH discretize the space using small particles both for 2D and 3D. In this thesis, even though we briefly explore other methods used for fracture simulation, our major focus lies on

FEM-based approaches.

2.1.2.1 Basics of FEM

Finite Element Method is one of the most popular mesh-based approaches to model object deformation. Müller and Gross [2004] proposed a co-rotational model with Cauchy linear strain for 3D deformable mesh simulation discretized using volumetric tetrahedral elements. In the co-rotational model, a deforming element is first projected back to its rest shape without the rotational part, deformation is then applied in the rest shape and finally projected back to the current position with the previously decoupled rotation matrix. Their method can handle artefacts produced due to large deformation and rotation. Smith et al. [2018] modelled extreme deformation of the flesh using the Neo-Hookean elastic energy. Using a set of lower-order invariants the authors later extended their work to include a larger class of isotropic elasticity models [Smith et al., 2019]. Please check [Kim and Eberle, 2022] [Smith et al., 2019] for a detailed explanation of the lower-order invariants. In our thesis, we do not make use of them. The basic theory of FEM is presented next.

Consider a three-dimensional domain $\Omega \in \mathbb{R}^3$ that is discretized into a mesh of n_{tet} tetrahedra. Moreover, let $\mathcal{N}$ be the set of n_v nodes shared by the tetrahedral mesh elements Δ_e. A displacement function, $\mathbf{u} : \Omega \times [0,\infty) \longrightarrow \mathbb{R}^3$, can be defined as a mapping from a material point $\xi \in \Omega$ to its deformed location $\mathbf{x} \in \Omega_t$ in the world space (see Figure 2.1). Ω_t represents the world space at time t. At a certain timestep $t \in [0,\infty)$, the displacement function can be represented as

$$\mathbf{u}(\xi,t) = \sum_{i \in \mathcal{N}} \mathbf{N}_i(\xi)\mathbf{u}_i(t), \tag{2.6}$$

where $\mathbf{N}_i(\xi)$ is the shape function and $\mathbf{u}_i$ is the displacement vector at the node i. The shape function, $\mathbf{N}_i(\xi)$ is a convex, linear interpolation function between the four nodes of the tetrahedral element. Kattan [2008] presents the detailed derivation and expression of the parameter. For the sake of brevity of notation, we drop the shape (ξ) and time (t) parameters for the shape and displacement functions in further discussion. Also, please note that the column-wise convention is used for all vectors and matrix variables throughout the thesis.

As discussed earlier when an object mesh is deformed from its rest position, a hyper-elastic strain energy density $\mathbf{\Psi}$, develops inside it. The strain energy density produces internal elastic force $\mathbf{f}^{int}$, which in turn tries to restore the rest shape of the mesh. Thus, the system dynamics of a deformed object can be written in Lagrange's form as

$$\mathbf{M}\ddot{\bar{\mathbf{u}}} + \mathbf{H}\dot{\bar{\mathbf{u}}} + \mathbf{f}^{int} = \mathbf{f}^{ext} \tag{2.7}$$

$$\bar{\mathbf{u}} = \left(\mathbf{u}_1^T \dots \mathbf{u}_{n_v}^T\right)^T \tag{2.8}$$

Here $\mathbf{M}$ and $\mathbf{H}$ are the mass and damping matrices of the full system. The external and internal force vectors are represented by $\mathbf{f}^{ext}$ and $\mathbf{f}^{int}$ respectively. Now Equation 2.7 can further be simplified as the sum of parameters from each individual tetrahedral element.

$$\sum_{e=1}^{n_{tet}} \mathbf{m}_e \ddot{\bar{\mathbf{u}}} + \sum_{e=1}^{n_{tet}} \mathbf{h}_e \dot{\bar{\mathbf{u}}} + \sum_{e=1}^{n_{tet}} \mathbf{f}_e^{int} = \sum_{e=1}^{n_{tet}} \mathbf{f}_e^{ext} \tag{2.9}$$

$$\mathbf{m}_e = \int_{\Delta_e} \rho_0 \mathbf{N}_e^T \mathbf{N}_e d\xi \tag{2.10}$$

$$\mathbf{f}_e^{ext} = \int_{\Delta_e} \mathbf{N}_e^T \mathbf{t}_e d\xi \tag{2.11}$$

$$\mathbf{f}_e^{int} = \int_{\Delta_e} \mathbf{s}_e(\bar{\mathbf{u}}, \xi) d\xi \tag{2.12}$$

$$\mathbf{N}_e = \left[\mathbf{N}_0\mathbf{I}_3\ \mathbf{N}_1\mathbf{I}_3\ \mathbf{N}_2\mathbf{I}_3\ \mathbf{N}_3\mathbf{I}_3\right]. \tag{2.13}$$

While $\mathbf{m}_e$ is the mass matrix of a tetrahedral element, $\mathbf{N}_e$, $\mathbf{t}_e$, ρ_0, $\mathbf{f}_e^{ext}$ and $\mathbf{f}_e^{int}$ represent element shape function vector, reference body force, rest density, external element force vector and

internal element force vector respectively. $\mathbf{I}_3$ refers to 3×3 identity matrix.

Specific elastic element force, $\mathbf{s}_e$, is defined as

$$\mathbf{s}_e = \frac{\partial \Psi^e}{\partial \mathbf{u}} \tag{2.14}$$

where Ψ^e is elemental hyper-elastic strain energy density. Parameter $\mathbf{h}_e = r_1 \mathbf{m}_e + r_2 \mathbf{k}_e + \tilde{\mathbf{h}}_e$ is the per element control damping matrix. Scalar quantities r_1 and r_2 control the level of 'mass' and 'stiffness' damping respectively while $\tilde{\mathbf{h}}_e$ is a user-defined additional damping matrix.

The gradient of $\mathbf{f}_e^{int}$ with respect to $\mathbf{u}$ is called tangent stiffness matrix $\mathbf{k}_e$.

$$\mathbf{k}_e = \int_{\Delta_e} \frac{\partial \mathbf{f}_e^{int}}{\partial \mathbf{u}} d\xi \tag{2.15}$$

The stiffness matrix of full system, $\mathbf{K}$, can be represented as

$$\mathbf{K} = \sum_{e=1}^{n_{tet}} \mathbf{k}_e \tag{2.16}$$

Interested readers can look into the review papers [Sifakis and Barbic, 2012] [Kim and Eberle, 2022] for further details.

2.1.2.2 Interactive FEM simulation

Most of the existing FEM-based deformation frameworks [Sin et al., 2013] [Smith et al., 2018] runs at real-time interactive rate ($\sim$ 30 fps) for low resolution meshes ($\sim$ 1000 tetrahedra). However, for high-resolution meshes, these methods cease to be interactive. Xian et al. [2019] developed a novel Galerkin multigrid method for fast simulation of deforming bodies. When parallelized on GPU, their algorithm is capable of simulating deformation at an interactive rate even for very high-resolution meshes containing more than six million tetrahedra. More recently, using an adaptive rigidification technique, Mercier-Aubin et al. [2022a] proposed an algorithm to speed up FEM simulation by a factor of $\times 5 - 10$ times.

2.1.3 What's Next?

Till now our discussion was revolving around the deformation of an object and its simulation strategies. The next obvious question that comes to mind is what happens if an object fails i.e. damage sets in. Our next section delves into that.

2.2 Simulation of Cutting and Fracture

Fracture occurs when an object disintegrates into multiple disjoint pieces under impact. Cutting denotes a specific type of fracture where the cut planes are pre-defined and smooth. Fracture is a notoriously difficult phenomenon to simulate. First, we present a brief theoretical explanation of fracture.

2.2.1 Theory of Fracture

Almost all fracture modelling approaches are rooted in Griffith's energy theory of fracture [Griffith, 1921]. The theory of Griffith for fracturing material is developed using balances of energy. According to his theory, any material fracture is a balance between a material's elastic potential stored by the material and the material's fracture energy, released as cracks emerge inside the object mesh. Mathematically, this can be put as below.

$$\mathscr{E}_{energy}(\mathbf{F},\Gamma) = \int_{\Delta_e} \mathbf{\Psi}^e(\mathbf{F}) d\mathbf{X} + \int_{\Gamma} \mathscr{G} d\mathbf{X} \tag{2.17}$$

where $\mathscr{E}_{energy}$ is the total energy in element Δ_e, $\mathbf{\Psi}^e$ is hyper-elastic strain energy density due to the deformation gradient, $\mathbf{F}$ and $\mathscr{G}$ is the Griffith critical energy release rate due to the discontinuities denoted by Γ.

When the shape of an object deviates from its original shape, strain and stress get developed inside the object mesh as depicted in Figure 2.5. In the figure, we notice the gradient of stress around a crack tip. The strain value gradually diminishes as it moves away from the tip.

From the zoomed-up version of Figure 2.5 (right), it is clear that the highest stress is accumulated around a point. The crack initiates from that particular point and then propagates inside the object along the direction of principal stress. If the present strain/stress value in an object

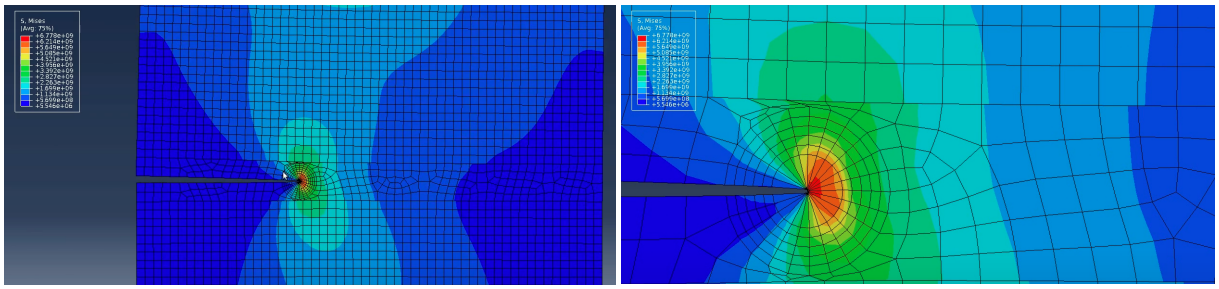


Figure 2.5: Contours of strain near the crack tip (left). Zoomed up strain profile near crack tip (right). (Courtesy to Youtube Tutorial, Last accessed - 2nd January 2023).

exceeds a critical threshold, the object fractures and the critical energy is released. The critical stress/strain condition for fracture simulation is widely known as the Rankine condition.

$$\sigma_e \geq \sigma_{thres} \tag{2.18}$$

where σ_e and σ_{thres} denote the stress in element Δ_e and critical stress threshold respectively.

As discussed in the last Chapter 1, material properties make an object brittle or ductile, impurities introduced due to environmental effects or human interventions change the material integrity and even anisotropic fibres cause an imbalance in the uniformity of material strength. All these factors influence crack propagation. In the next section, we explore how different research communities proposed solutions for them.

2.2.2 Mesh-based Fracture Simulation

The inception of fracture simulation in computer graphics goes back to the seminal work on visco-elastic fracture by Terzopoulos and Fleischer [1988]. Early approaches propose to model brittle fracture by removing the springs in a mass-spring system [Aoki et al., 2004] [Norton et al., 1991] [Hirota et al., 2000] [Hirota et al., 1998] depending on a stress-based yield threshold. But in such a system, where point masses are connected by springs, the sudden removal of

springs leads to significant visual artefacts. Moreover, visualization of the crack surface often requires a tetrahedral marching algorithm and is therefore computationally expensive [Hirota et al., 2000] [Hirota et al., 1998]. Later, finite element method-based solutions for simulating fracture have been more successful and different variations of those approaches are still widely used. The first breakthrough in FEM-based fracture came in the paper by O'Brien and Hodgins [1999]. In their work, the authors present a nodal stress-based analysis for brittle fracture that was later extended further to ductile fracture [O'Brien et al., 2002] [Müller and Gross, 2004]. Bao et al. [2007] also present a method to simulate both brittle and ductile fractures. These FEM-based fracture techniques rely on splitting the elements of a tetrahedral mesh, which satisfy the fracture conditions and then remesh the entire fractured mesh with the crack opening. However, these approaches suffer from various geometric difficulties like remeshing near crack tips — as a crack tip tends to be sharp and thin, very high-resolution remeshing is required to accurately represent it; generation of degenerate elements — due to remeshing a few angles of some new tetrahedra may become very small, and thus forming ill-conditioned basis matrices. An ill-conditioned basis matrix implies that the linear shape function of the tetrahedra (see Equation 2.13) becomes so skewed that the condition number of the matrix exceeds a threshold. In such a case, even a small deformation leads to instability. Subsequent FEM-based approaches employ different techniques to alleviate these problems, such as dynamic local mesh refinement to repair degenerate tetrahedra [Wicke et al., 2010], remeshing depending on gradient descent flow for finer fracture resolution [Chen et al., 2014], adaptive subdivision schemes for tetrahedral [Koschier et al., 2015] and triangular meshes [Pfaff et al., 2014] where remeshing is done around the high-stress areas to improve fracture resolution.

FEM-based solutions that work with a virtual node algorithm (VNA) [Molino et al., 2004] duplicate elements and add extra degrees of freedom to facilitate partial or full crack openings, instead of splitting the mesh elements. Later, VNA-based methods are improved to incorporate cutting at lower than mesh resolution [Sifakis et al., 2007] and to robustly handle intersections passing through a node, edge or face through an adaptive element duplication approach [Wang et al., 2015]. Originally developed in material science [Moës et al., 1999] [Areias and Belytschko, 2005] [Belytschko and Black, 1999] [Zi and Belytschko, 2003] [Zhu, 2012], XFEM has found its way in computer graphics simulation [Koschier et al., 2017] [Chitalu et al., 2020]. Koschier et al. [2017] presented a remeshing-free extended finite element method (XFEM) based algorithm to simulate dynamic pre-defined cuts in a 3D mesh. Like VNA, XFEM too adds

extra degrees of freedom using enrichment functions but the advantage of XFEM over VNA is while VNA duplicates the mesh element leading to erratic conservation of mass, XFEM splits the initial mass of an element into fragments. Chitalu et al. [2020] expanded the pre-defined cuts to crack generation and propagation. One major drawback of XFEM is that the system matrix can become singular, making the system unstable. Another version of XFEM called Improved stable XFEM (Is-XFEM) presented by Wu and Li [2015] in material science literature, improves system stability by avoiding the singular matrix generation.

Contrary to processing fracture in volume elements like FEM, boundary element methods (BEM) simulate cracks on a surface mesh. BEM was used in graphics for simulating elastostatic deformable objects [James and Pai, 1999]. It has been used recently for fracture simulation. Hahn and Wojtan [2015] and Zhu et al. [2015] used BEM with stress intensity factors (SIFs) along crack fronts to successfully simulate brittle fracture. Later Hahn and Wojtan [2016] expanded their work to develop an algorithm for fast approximation of BEM brittle fracture, alleviating the high computational cost for solving singular integrals. Currently, to the best of our knowledge, there exists no work on BEM-based ductile fracture in graphics.

Peridynamics methods solve the integral equations of continuum mechanics. Unlike discrete differential-based methods e.g., FEM or MPM, which need to impose necessary boundary conditions for simulating fracture, it is trivial to model discontinuities using integral-based peridynamics. In initial work using these ideas, Levine et al. [2015] approximated the behaviour of peridynamics using mass-spring systems to simulate brittle fracture. Later work [Chen et al., 2018] presented original peridynamics-based fracture algorithms to simulate brittle as well as ductile fracture. However, peridymanics incurs a high computational cost. Moreover, it is completely non-local dynamics i.e. the state of every point inside an object depends on the state of other points far away from it.

2.2.3 Meshless Fracture Simulation

Among the meshless methods, material point methods have gained considerable recognition in recent years. The first major breakthrough of MPM in computer graphics came through the seminal work in snow simulation by Stomakhin et al. [2013]. Later MPM found its application

in simulating a wide range of materials as such chocolate, lava, hybrid fluids and rubber, to name a few. Interested readers can look at [Jiang et al., 2016] for a detailed overview of different MPM methods. Most recently, MPM has shown promising results in simulating fracture due to its ability to handle extreme topological change. Hu et al. [2018a] proposed the Compatible Particle-in-Cell (CPIC) algorithm, which allows the simulation of sharp discontinuities inside a material and thus simulates dynamic material cutting. Later Wolper et al. [2019] introduced Continuum Damage Mechanics (CDM) with a variational energy-based formulation for crack evolution. This achieves a realistic dynamic fracture simulation for isotropic materials. Further study [Wolper et al., 2020] devoted to formulating non-local CDM to simulate anisotropic material fracture.

MPM methods reproduce realistic fracture phenomena but they are still not well suited to rendering sharp boundaries. They not only have difficulty in enforcing essential boundary conditions, but they also suffer from the disappearance of intricate crack patterns due to excessive smoothing, high computational cost and rigidity of fragmented elements. Further, MPM methods struggle to simulate the fracture of rigid objects.

2.3 Graph-based Finite Element Method

While simulating fractures in the material domain, classical FEM introduces degenerate elements due to remeshing and XFEM has several limitations like ambiguities of crack-tip enrichment in 3D, instabilities induced by large volume ratios in Heaviside enrichments, difficulties of handling branch enrichments in 3D. Moreover, in both of these methods, computational cost increases linearly with an increase in the number of cracks.

Work in finite element analysis literature by Reddy and Srinivasa [2015] proposed a solution based on classical FEM to show that for any hyper-elasticity problem, the magnitude of the nodal forces can be written completely in terms of strain energy density along the edges composing the elements, while force directions are along the unit vectors corresponding to the edges. Using this idea in subsequent work, Khodabakhshi et al. [2016] introduced Graph-based Finite Element Analysis (Gra-FEA) where a damage variable corresponding to the edges of composing elements is used. It is based on the strain threshold and thus can simulate the

fracture of an object by weakening the material. The advantage of using Gra-FEA is that it requires no remeshing while adding little computational overhead on FEM and it still retains all the advantages of FEM. The dependency of Gra-FEA on the underlying mesh structure is thoroughly studied in another work [Khodabakhshi et al., 2019]. Our work introduces Gra-FEA to the visual simulation of fracture for computer graphics. Existing literature referred to above, on Gra-FEA, only focuses on 2D materials consisting of triangular elements with linear strain energy density. We extend the idea to the 3D domain and our fracture algorithm can handle linear as well as non-linear strain energy density. The original Gra-FEA formulation is restricted to simulating fracture for brittle materials. Our algorithm is capable of simulating fracture for both brittle and ductile materials. We believe, to the best of our knowledge, that this is the first work in computer graphics to simulate the fracture of 3D brittle, as well as ductile objects, with linear or non-linear strain energy density using graph-based FEM.

2.4 What This Thesis Offers

Although, there exist several fracture simulation frameworks as discussed earlier, in every one of them the computational cost keeps on increasing as more cracks/discontinuities are introduced. We propose a graph-based FEM framework which incurs a negligible computational cost over regular FEM due to no remeshing. Thus, the run-time of graph-based FEM remains independent of the number of cracks initiated inside an object mesh. Moreover, the existing method renders fracture for a subset of brittle, ductile, anisotropic and impure materials. None of the frameworks combines all four types of material under a single algorithm. Our proposed graph-based FEM framework can handle fractures of all these varieties using a few simple extensions like the random graph. Finally, when parallelized on GPU with a Galerkin Multigrid method [Xian et al., 2019], our fracture simulation framework runs interactively in real-time. To sum up, in this thesis we present a physics-based, interactive FEM-based fracture simulation framework that can handle a wide range of materials e.g., brittle & ductile, isotropic & anisotropic, pure & impure.

Before delving into the details of our proposed methods, we quickly explain some standard real-world experiments that are frequently used to evaluate different parameters of a material,

e.g. failure threshold and material strength. Later, using our proposed algorithms, we simulate the fracture of various brittle and ductile materials. We compare the fracture simulation results with these benchmark experiments and demonstrate the accuracy of our methods in simulating fracture for different kinds of materials.

2.5 Benchmark Experiments in Fracture Simulation

Researchers developed multiple benchmark experiments to evaluate various properties of material fracture, e.g., brittleness, plastic flow and strength etc. To gauge the accuracy of our methods, we simulate three real-world experiments, 'Charpy Impact Test', 'Izod Impact Test' and 'Brazilian Test', using our framework. We briefly explain each one of them here.

2.5.1 Charpy Impact Test

The Charpy impact test is a standardized test which measures the impact energy of materials during fracture which is carried out by a pendulum. To carry out the test, a standard bar of a specific material is first supported at its two ends on an anvil as shown in Figure 2.6 (b). The bar contains a 45° angled V-notch in the middle of it. The bar is then struck on the opposite face of the notch by a swinging pendulum. The specimen is fractured and the height of the swing is measured to calculate the amount of energy absorbed in fracturing the specimen. Generally, during the experiment, the displacement of the fractured point on the V-notch is plotted against the load applied by the swinging pendulum to evaluate the ductility and strength of the material (see Figure 2.7).

2.5.2 Izod Impact Test

The setup of the Izod impact test is similar to the Charpy test. However, as shown in Figure 2.6(a), unlike Charpy, in the Izod test the standard specimen is held fixed at one end vertically and the swinging pendulum hits it on the free end. Here also, a similar load-displacement

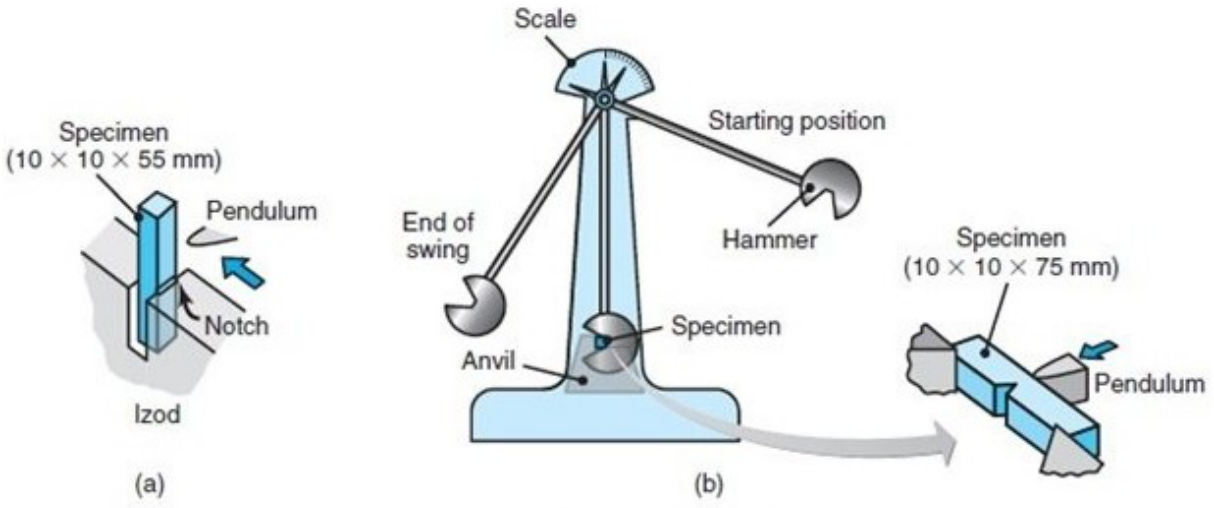


Figure 2.6: Impact Test setup for (a) Izod and (b) Charpy. Courtesy: Quora

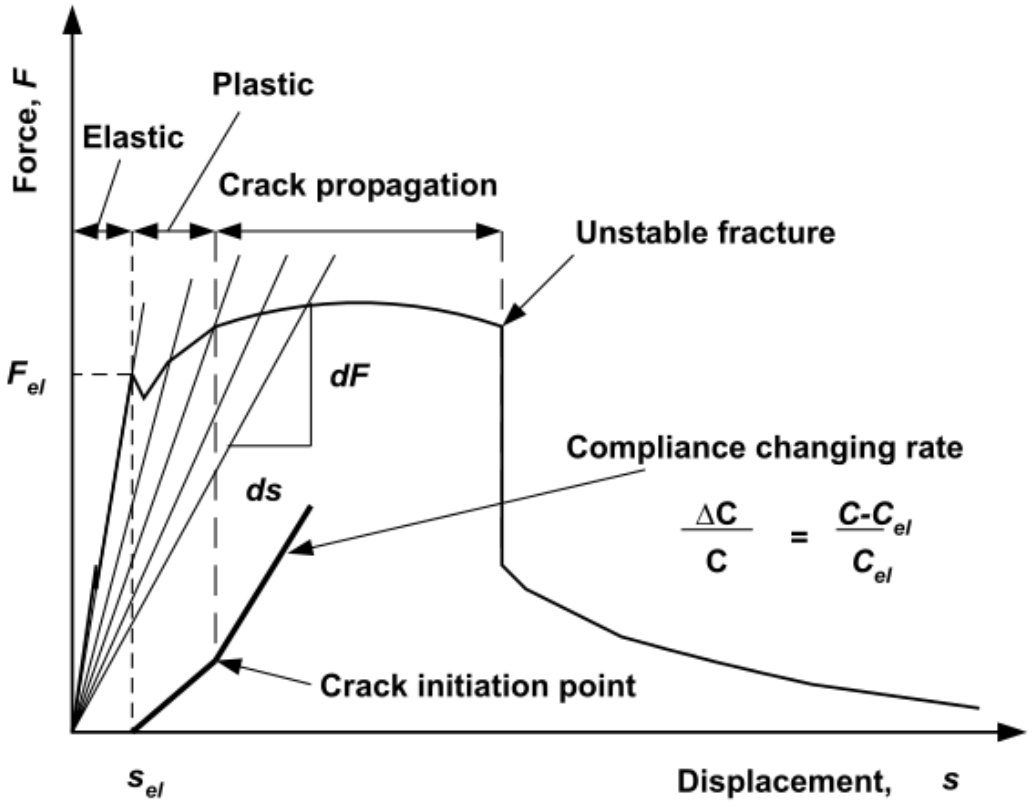


Figure 2.7: Load-Displacement curve of Charpy and Izod tests [Tronskar et al., 2002].

curve (see Figure 2.7) is plotted to evaluate the ductility and strength of the test specimen.

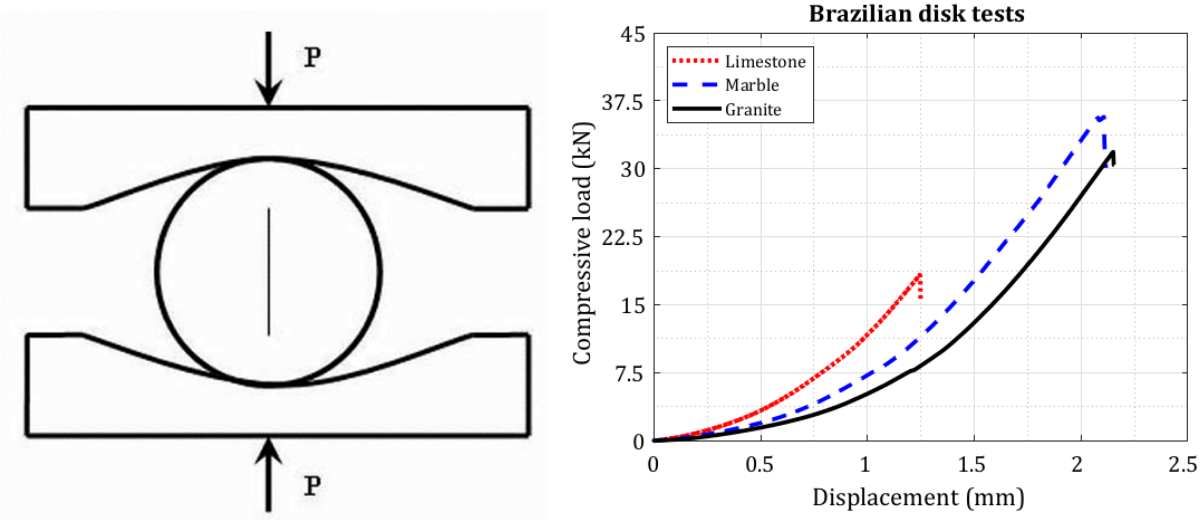


Figure 2.8: Brazilian Test setup (left). The load-Displacement curve of Brazilian test [Ghouli et al., 2021] (right).

2.5.3 Brazilian Test

The Brazilian test is designed to obtain the tensile strength of brittle materials such as concrete, rock, and rock-like materials in an indirect manner. As illustrated in Figure 2.8 (left), in this test, a thin circular disc is diametrically placed between two metal bearings, one on top and another at the bottom. The cylindrical structure is then compressed from both ends to its failure. In Figure 2.8 (left), 'P' depicts the applied load. Like the two experiments before, the Brazilian test also has its own distinct load-displacement characteristics curve shown in Figure 2.8 (right).

This page was intentionally left blank.

Chapter 3

Improved stable XFEM for Cutting of Deformable Objects

3.1 Introduction[1]

Our initial objective is to develop a mesh-based robust, fast and efficient algorithm for object cutting and fracture. We attempt to improve existing cutting algorithms that are stable and remeshing-free. In order to simulate cuts on mesh objects without remeshing, we take our inspiration from [Koschier et al., 2017] [Chitalu et al., 2020] to use XFEM. However, these XFEM methods are prone to instability due to the system matrix becoming singular. We enhance their model to an Improved stable eXtended Finite Element Method [Wu and Li, 2015]. To the best of our knowledge, we are the first to use the Is-XFEM method in any kind of interactive frame-

[1]Portions of this chapter are adapted from *Interactive Physics-Based Virtual Sculpting with Haptic Feedback* by Avirup Mandal, Parag Chaudhuri, and Subhasis Chaudhuri, published in ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D), 2022.

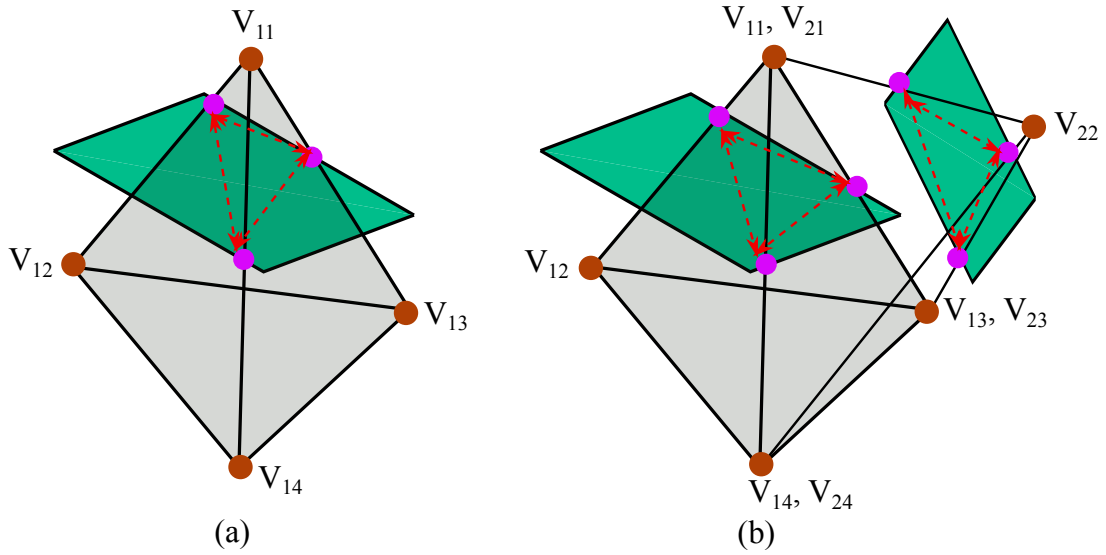


Figure 3.1: Red dotted lines indicate the cut path followed by the cut brush on the tetrahedral mesh. Extra degrees of freedom are added to the nodes (shown with red points). The pink points are the intersection points.

work.

Our interactive framework contains a virtual cutting tool/brush which is controlled by the user using a mouse or haptic device. When the brush collides with an object mesh, the cut occurs. Compared to real life a virtual brush can be thought of as a piece of sculpting equipment for chip stones or shape clay. More details about the framework are presented in Chapter 7. In this chapter, we focus on the technique of cutting a mesh using Is-XFEM.

3.2 Is-XFEM

The solid deformable object to be cut is represented as a tetrahedral mesh. The blade of the cutting brush is a triangular mesh. To begin the cutting process, we decide the path the cut brush follows on the tetrahedral mesh (red dotted lines in Figure 3.1). To determine the intersection path between the tetrahedral mesh and the cut brush mesh, we use the algorithm developed by Möller [1997]. Each tetrahedron consists of four triangles. Using a traversal algorithm [Baraff et al., 2003] we check if the cut boundary forms a closed loop. Only the closed-loop cut planes are considered. Subsequently, extra degrees of freedom (DOF) are added to the corresponding nodes (Figure 3.1).

If any tetrahedral element Δ_e has m numbers of cut planes, defined as $\Lambda_1, \Lambda_2 \ldots \Lambda_m$, a signed distance function [Koschier et al., 2017] of a point w.r.t. the j^{th} cut plane can be defined as

$$\Phi^j(\xi) = s(\xi) \inf_{\xi^* \in \Lambda_j} ||\xi - \xi^*|| \tag{3.1}$$

where $s : \mathbb{R}^3 \longrightarrow \{-1, 1\}$ denotes the sign of the distance and $\xi \in \mathbb{R}^3$. Better stability is achieved in our algorithm by using an improved enrichment function ψ_i^j as expressed in Equation 3.2. The enrichment formulation is first proposed by Wu and Li [2015]. The enrichment function ψ_i^j represents the enrichment or increase in the number of degrees of freedom of the node i due to the cut j.

$$\psi_i^j(\xi) = \begin{cases} 1, & \text{No cut} \\ H_s\left(\Phi^j(\xi)\right) - \left[k\gamma(\xi) + (1-k)H_s\left(\Phi^j(\xi_i)\right)\right], & \text{Cut} \end{cases} \tag{3.2}$$

where $0 < k \leq 1$ and H_s is the Heaviside function. Linear interpolant, γ, is defined as

$$\gamma(\xi) = \sum_{i=1}^{n_v} \mathbf{N}_i(\xi) H_s(\xi_i) \tag{3.3}$$

where $\mathbf{N}_i$ is the linear shape function of node i. The Is-XFEM enrichment function in Equation 3.2 reduces to XFEM for $k = 0$. We use a small value of $k = 0.1$ for our Is-XFEM simulation. Following the arguments presented in [Wu and Li, 2015], it can be proved that Is-XFEM improves the condition number of the system considerably, rendering the system more stable. Using the definition of ψ_i^j from Equation 3.2, displacement function from Equation 2.6 can be updated for Is-XFEM as

$$\mathbf{u}^*(\xi, t) = \sum_{i=1}^{n_v} \mathbf{N}_i(\xi)\mathbf{u}_i(t) + \sum_{j=1}^{m} \sum_{i=1}^{n_v^{enr}} \psi_i^j(\xi)\mathbf{N}_i(\xi)\mathbf{u}_i^j(t) \tag{3.4}$$

where $\mathbf{u}_i$ denotes the displacement of i^{th} node if no cut is present and $\mathbf{u}_i^j$ represents displacement of same i^{th} node when it gets enriched due to j^{th} cut. n_v^{enr} represents the number of enriched nodes. With this displacement function $\mathbf{u}^*(\xi, t)$, the per-element mass matrix from

Equation 2.10 and external force matrix from Equation 2.11 can be written as

$$\mathbf{m}_e^* = \int_{\Delta_e} \rho_0 \begin{bmatrix} \mathbf{N_e} & \mathbf{N_e^1} & \dots & \mathbf{N_e^m} \end{bmatrix}^T \begin{bmatrix} \mathbf{N_e} & \mathbf{N_e^1} & \dots & \mathbf{N_e^m} \end{bmatrix} d\xi \tag{3.5}$$

$$\mathbf{f}_e^{ext^*} = \int_{\Delta_e} \begin{bmatrix} \mathbf{N_e} & \mathbf{N_e^1} & \dots & \mathbf{N_e^m} \end{bmatrix}^T \mathbf{b} d\xi, \quad \mathbf{f}_e^{int^*} = \int_{\Delta_e} s_e^*(\overline{\mathbf{u}}, \xi) d\xi \tag{3.6}$$

$$\mathbf{N}_e^j = \begin{bmatrix} \psi_0^j \mathbf{N}_0 \mathbf{I}_3 & \psi_1^j \mathbf{N}_1 \mathbf{I}_3 & \psi_2^j \mathbf{N}_2 \mathbf{I}_3 & \psi_3^j \mathbf{N}_3 \mathbf{I}_3 \end{bmatrix} \tag{3.7}$$

where $\mathbf{m}_e^*$, s_e^*, $\mathbf{f}_e^{ext^*}$ and $\mathbf{f}_e^{int^*}$ are element mass matrix, elastic element force and external & internal element force vectors respectively for Is-XFEM. While $\mathbf{N}_e$ denotes the shape function without any cut present, $\mathbf{N}_e^j$ denotes the updated shape function of a tetrahedral element Δ_e corresponding to j^{th} cut. The same system dynamics for FEM deformable objects as defined in Equation 2.7 is still applicable for Is-XFEM with these updated parameters. The discontinuous integrands in Is-XFEM are solved using a highly accurate Gaussian quadrature rule [Müller et al., 2013] [Koschier et al., 2017]. This is described below.

3.2.1 Gaussian Quadrature Rule

The integrals that appear in Is-XFEM are solved using the surface and volume Gaussian quadrature integration rules. Regular Gaussian quadrature rule [Gustafson and Hagler, 1999] is not fit for solving the discontinuous integrals that originate from the Is-XFEM method. To solve the field equations over tetrahedral domains that are cut by implicit surfaces, we use a generalized and highly accurate surface and volume Gaussian quadrature integration presented in [Müller et al., 2013] [Koschier et al., 2017]. A visual depiction of the Gaussian quadrature rule for a cube mesh split by an arbitrary plane is presented in Figure 3.2. In this method, first, the volume integrals are converted to surface integrals by the divergence theorem. Following that, the surface integrals are converted to line integrals using Stokes' theorem. The motivation for doing so is that it is much easier to evaluate the 1-D line integrals compared to their 2-D or 3-D

counterparts. The mathematical details of the rule are presented next.

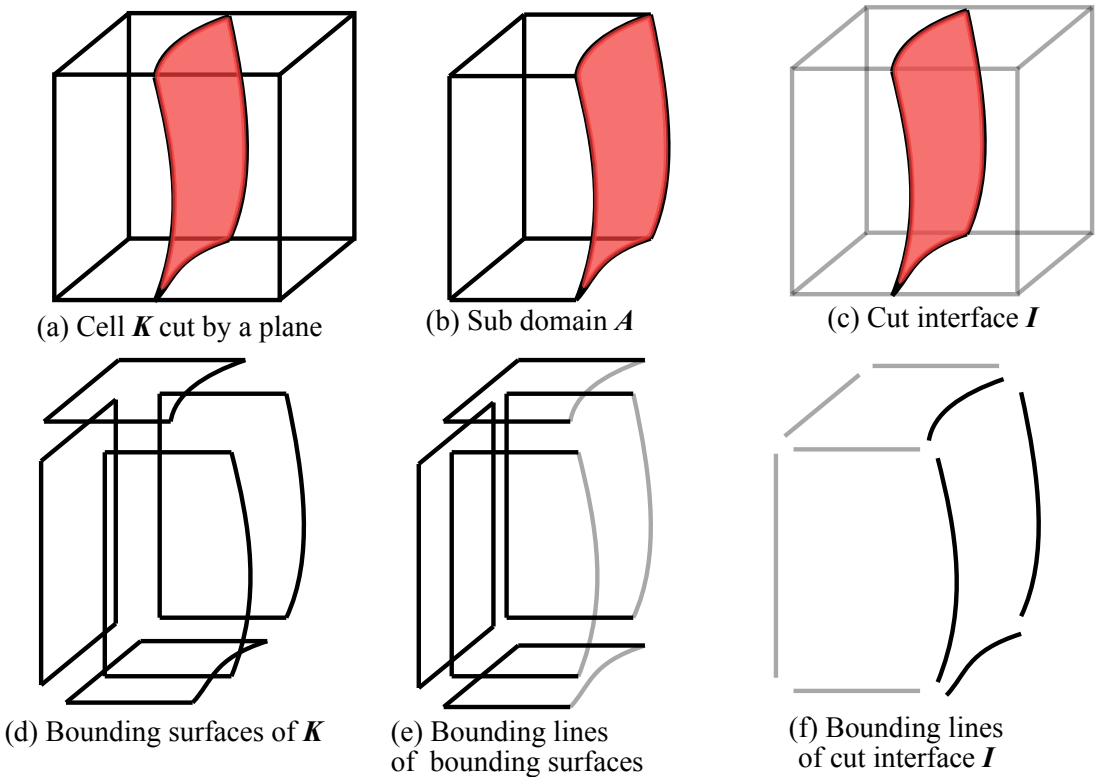


Figure 3.2: Visual depiction of Gaussian quadrature rule.

Let the cut plane, defined as $\mathscr{I}$, intersect a tetrahedral element Δ_e and split it into two distinct sub-domains. Let Δ_{e_i} denote the i^{th} sub-domain of Δ_e. Then by Gaussian quadrature integration rule of any function, h over that sub-domain can be written as

$$\int_{\Delta_{e_i}} h(\xi) d\xi = \int_{\Delta_e} \chi_i h(\xi)\, d\xi \approx \sum_{j=1}^{N} w_{i,j} h\left(\xi_j\right) \tag{3.8}$$

where ξ_j and $w_{i,j}$ are the quadrature points and weights corresponding to i^{th} sub-domain and j^{th} cut. χ_i is a characteristic function for i^{th} sub-domain, i.e., it returns one inside the sub-domain and zero everywhere else. Quadrature points of a tetrahedron are a set of predefined points, which can be calculated as per [Zhang et al., 2009]. The weights $w_{i,j}$ need to be calculated only once using a particular polynomial basis. The same weights can be reused to integrate any arbitrary bounded function.

3.2.1.1 Volume Quadrature

Let a set of polynomial integrands be $\mathscr{F} = \{f_j\}_{j=1,\ldots,M}$ and their anti-derivative be $\mathbf{f}_j$, where $\nabla \cdot \mathbf{f}_j = f_j \ \forall j$. Then using the divergence theorem, the quadrature rule to find out the weights for the tetrahedral volume element Δ_e can be written as

$$\begin{bmatrix} f_1(\xi_1) & \ldots & f_1(\xi_N) \\ \vdots & \ddots & \vdots \\ f_M(\xi_1) & \ldots & f_M(\xi_N) \end{bmatrix} \begin{bmatrix} w_{i,1} \\ \vdots \\ w_{i,N} \end{bmatrix} = \begin{bmatrix} \int_{\partial\Delta_e} \chi_i \mathbf{f}_1 \cdot \mathbf{n} d\xi \\ \vdots \\ \int_{\partial\Delta_e} \chi_i \mathbf{f}_M \cdot \mathbf{n} d\xi \end{bmatrix} \tag{3.9}$$

To solve the right-hand side of Equation 3.9 we need to construct a surface quadrature rule because $\partial\Delta_e$ consists of surface triangular elements and $\mathbf{n}$ denotes the normal to that surface.

3.2.1.2 Surface Quadrature

Following the arguments presented in [Müller et al., 2013], we construct the surface quadrature rule, which closely resembles volume quadrature except for the choice of the set of polynomial integrands.

The expression $\int_{\partial\Delta_e} \chi_i \mathbf{f}_j \cdot \mathbf{n} d\xi$ can be written as

$$\int_{\partial\Delta_e} \chi_i \mathbf{f}_j \cdot \mathbf{n} d\xi = \underbrace{\int_{\partial\Delta_{e_i} \setminus \mathscr{I}} \mathbf{f}_j \cdot \mathbf{n} d\xi}_{\text{I}} + \underbrace{\int_{\mathscr{I}} \mathbf{f}_j \cdot \mathbf{n} d\xi}_{\text{II}} \tag{3.10}$$

where $\partial\Delta_{e_i} \setminus \mathscr{I}$ are the surfaces of a tetrahedral element and $\mathscr{I}$ is the cut plane that intersects the tetrahedron.

Following similar arguments presented in volume quadrature, Gaussian quadrature rule can be constructed over the four surfaces of a tetrahedral element, $\partial\Delta_{e_i} \setminus \mathscr{I}$, to evaluate the first part of Equation 3.10. During evaluating volume quadrature, we notice that the right side of Equation 3.9 consists of functions integrated over two-dimensional area elements. Similarly, in surface Gaussian quadrature, the right-hand side consists of one-dimensional line elements which can be evaluated easily using the standard Gaussian quadrature rule.

In the second part of Equation 3.10, as the cut plane $\mathscr{I}$ can be any arbitrary nonlinear

surface, constructing an accurate surface is computationally expensive. This problem is solved by choosing a divergence-free basis of integrands $\mathscr{F}' = \{\mathbf{f}'_k\}_{k=1,\dots,K}$. Now the quadrature rule for the cut surface can be written as

$$\begin{bmatrix} \mathbf{f}'_1(\xi_1)\cdot\mathbf{n}_I(\xi_1) & \dots & \mathbf{f}'_1(\xi_N)\cdot\mathbf{n}_I(\xi_N) \\ \vdots & \ddots & \vdots \\ \mathbf{f}'_K(\xi_1)\cdot\mathbf{n}_I(\xi_1) & \dots & \mathbf{f}'_K(\xi_N)\cdot\mathbf{n}_I(\xi_N) \end{bmatrix} \begin{bmatrix} w_{i,1} \\ \vdots \\ w_{i,N} \end{bmatrix} = \begin{bmatrix} \int_{\mathscr{I}} \mathbf{f}'_1\cdot\mathbf{n}_I d\xi \\ \vdots \\ \int_{\mathscr{I}} \mathbf{f}'_K\cdot\mathbf{n}_I d\xi \end{bmatrix} \tag{3.11}$$

where $\mathbf{n}_I$ is the normal to the cut surfaces.

The right hand side of the Equation 3.11 can be rewritten as

$$\begin{aligned} \int_{\mathscr{I}} \mathbf{f}'_k\cdot\mathbf{n}_I d\xi &= \int_{\partial\Delta_{e_i}} \mathbf{f}'_k\cdot\mathbf{n} d\xi - \int_{\partial\Delta_{e_i}\setminus\mathscr{I}} \mathbf{f}'_k\cdot\mathbf{n} d\xi \\ &= \int_{\partial\Delta_{e_i}} \nabla\cdot\mathbf{f}'_k d\xi - \int_{\partial\Delta_{e_i}\setminus\mathscr{I}} \mathbf{f}'_k\cdot\mathbf{n} d\xi \\ &= 0 - \int_{\partial\Delta_{e_i}\setminus\mathscr{I}} \mathbf{f}'_k\cdot\mathbf{n} d\xi = -\int_{\partial\Delta_e\setminus\mathscr{I}} \chi\mathbf{f}'_k\cdot\mathbf{n} d\xi \end{aligned} \tag{3.12}$$

Substituting Equation 3.12, the integration domain of the right side of Equation 3.11 consists of the surfaces of the tetrahedral elements.

3.2.2 Stability of System Dynamics

While solving the system of Equations 2.7, we employ the conjugate gradient method. Note that when a tetrahedron gets cut in two parts, the ratio of the volume of these two parts may be highly skewed. This leads to a high condition number for the system, which indicates a loss of stability. In order to improve the stability, we apply pre-conditioning with the volume ratios of cut tetrahedra and we constrain the system. The discretized version of the system Equation 2.7 is

$$\underbrace{\left(\mathbf{M}+\Delta t^2\mathbf{K}\right)}_{\mathbf{A}}\mathbf{v}^{k+1} = \underbrace{\mathbf{M}\mathbf{v}^k - \Delta t\left(\mathbf{K}\mathbf{x}^k + \mathbf{f}_0 + \mathbf{f}_p - \mathbf{f}_{ext}\right)}_{\mathbf{b}} \tag{3.13}$$

where $\mathbf{f}_0, \mathbf{f}_p$ and $\mathbf{f}_{ext}$ denote initial force, plasticity force and external force respectively. The parameters, $\mathbf{v}^k$ and $\mathbf{x}^k$ denote velocities and positions of the nodes at k^{th} timestep. To stabilize this system of equations, we apply the following two methods.

3.2.2.1 Pre-conditioning

A pre-conditioner matrix $\mathbf{T}_p$ is applied to our system dynamics Equation 3.13 as shown in Equation 3.14.

$$\mathbf{Ax} = \mathbf{b} \implies \mathbf{T}_p^T \mathbf{A} \mathbf{T}_p \mathbf{y} = \mathbf{T}_p^T \mathbf{b} \implies \mathbf{x} = \mathbf{T}_p \mathbf{y} \tag{3.14}$$

The diagonal pre-conditioner matrix is constructed as

$$\mathbf{T}_p = \text{diag}\left(\frac{1}{\sqrt{v_{1,j}}}, \frac{1}{\sqrt{v_{2,j}}}, \ldots \frac{1}{\sqrt{v_{n,j}}}\right) \tag{3.15}$$

where $i = 1, 2, \ldots n$ are node numbers and j denotes the cut number. $v_{i,j}$ denotes the volume ratio and is defined as

$$v_{i,j} = \frac{V_{i,j,enr}}{V_{i,supp}} = \frac{\sum_{e \in C_i} \sum_{j=1}^{N} w_{e,j}}{V_{i,supp}} \tag{3.16}$$

where C_i is the set of all incident tetrahedra on i^{th} node and $w_{e,j}$ is the weights of Gaussian quadrature corresponding to j^{th} cut. As a particular node i is shared by multiple tetrahedra, $V_{i,supp}$ denotes the volume of the support domain of the i^{th} node. Similarly, as a cut plane, j spans multiple tetrahedra, $V_{i,j,enr}$ denotes the volume of the enriched support domain volume associated with the i^{th} node and the jth cut-plane enrichment.

3.2.2.2 Constraining the System of Equation

Along with pre-conditioning, to improve the condition number of the system of equations, when the ratio $v_{i,j}$ is too small, we constrain the system of equations, Equation 3.13 like $(\mathbf{A} + \lambda \mathbf{I})\,\mathbf{x} = \mathbf{b}$ where $\lambda > 0$ and $\mathbf{I}$ is an identity matrix. The purpose of perturbing the diagonal elements of the matrix $\mathbf{A}$ is to prevent the issue of inverting a singular matrix.

3.3 Results

All the experiments presented here are carried out in a Windows 7 operating system with an Intel i7-4770K octa-core processor, 32GB DDR3 RAM and a single Nvidia Geforce GTX Titan GPU with 6 GB of graphics memory. We do not handle self-collision in our work.

3.3.1 Cutting and Visualizing the Object Mesh

During the cutting simulation, first, the cut surface plane is replicated on both sides of the volumetric mesh. The nodes of the elements that got cut are appropriately enriched for Is-XFEM simulation. Finally, when the object is released under gravity, the cut opening becomes visible. This can be seen in Figure 3.3 (left). In the figure, we show the boundary of the cut on the surface mesh with a blue solid line. The model is fixed in the middle and the two sides of the partitioned mesh dangle under gravity. In Figure 3.3(right) we show multiple cuts on a Lioness model. The cutting simulation with the Lioness mesh consisting of 1.3k tetrahedra runs at 122.5 frames/sec when no cut is present and the simulation rate drops to 109.7 frames/sec when a single cut is introduced. It again drops to 96.2 frames/sec when two cuts occur.

While simulating mesh cutting with Is-XFEM, we use a multi-resolution framework where physical computations are performed on a low-resolution mesh. A high-resolution mesh is embedded inside the low-resolution mesh for better visual appeal. For the sake of continuity, we move the details of the multigrid method in Section 7.3.2. Without the multigrid method, to the similar visual quality, the same Linoness mesh must consist of at least 40k tetrahedra. In that scenario, the Is-XFEM simulation remains no longer interactive and runs at 2-3 frames/sec.

3.3.2 XFEM vs Is-XFEM

We compare the results rendered using Is-XFEM and XFEM in Figure 3.4. We use the method presented in [Koschier et al., 2017] for the XFEM simulation. We perform the exact same cutting operation on a T-Rex model using both Is-XFEM and XFEM. While Is-XFEM remains

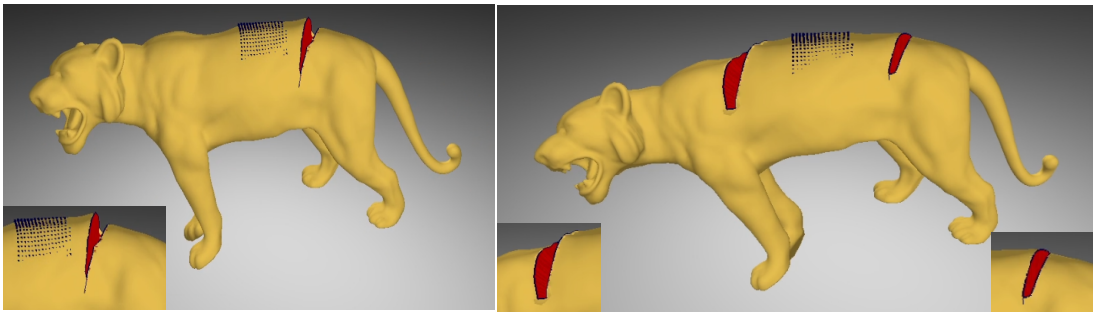


Figure 3.3: Single (left) cut and multiple (right) cuts on a Lioness model. The cut portions are zoomed up in the inset.

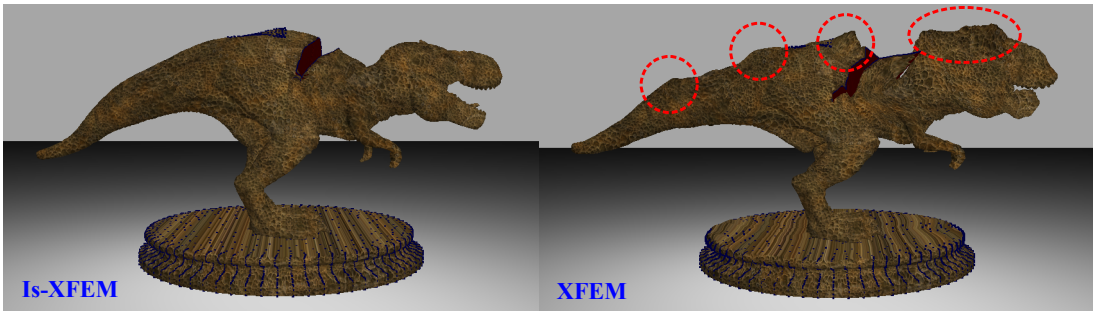


Figure 3.4: Cutting operation performed with Is-XFEM (left) and XFEM (right). Simulation rendered with Is-XFEM is more stable compared to XFEM. The kinks generated due to instability are marked with red circles and ellipses.

stable, XFEM becomes unstable. Is-XFEM improves the condition number of the system considerably [Wu and Li, 2015], rendering the system more stable.

3.4 Discussion

In this chapter, we present the first, novel application of Is-XFEM to interactive visual simulation. Our Is-XFEM approach for cutting deformable objects is physically accurate as per our objective mentioned in Section 1.2.1‘*Accuracy*’. While working on Is-XFEM, we come across a few major limitations of this method and decide against extending it to a full-fledged fracture simulation framework.

- One of the major limitations of both XFEM and Is-XFEM is that with the increase in the number of cuts in the object mesh the size of the system matrix for Is-XFEM increases.

This, in turn, makes the simulation slower.

- Another significant disadvantage of XFEM is that it is highly unstable. Even though Is-XFEM fares better in stability compared to XFEM, it still tends to diverge for a high number of random complex cuts.

- Finally, Is-XFEM can't run in a real-time interactive manner for very high-resolution meshes ($\sim$ 5k). For a mesh consisting of 5k tetrahedra and two cuts, the frame rate of Is-XFEM drops below 15 frames/sec, thus rendering the simulation non-interactive.

We address these limitations of Is-XFEM in the works presented in subsequent chapters of this thesis.

This page was intentionally left blank.

Chapter 4

Remeshing-Free Graph-based Finite Element Method for Fracture Simulation

4.1 Introduction[1]

The drawbacks of the XFEM method motivate us to develop a method whose run-time is independent of the number of cracks initiated inside the object mesh. In this Chapter, we propose a novel graph-based Finite Element Method for fracture simulation method. We perform our simulations on volumetric meshes with tetrahedral elements. This underlying mesh, on which all computation is performed, is called the *computational mesh*. This mesh induces a graph where its vertices become the nodes of the graph and the edges of the mesh become the edges of the graph. The computational mesh is never remeshed. This implies that, unlike XFEM, the

[1]Portions of this chapter are adapted from *Remeshing-Free Graph-Based Finite Element Method for Fracture Simulation* by Avirup Mandal, Parag Chaudhuri, and Subhasis Chaudhuri, published in Computer Graphics Forum, 2023.

Figure 4.1: Our novel fracture simulation algorithm produces the intricate fracture patterns that result from the tearing of a loaf of bread.

number of degrees of freedom of the system does not change with the introduction of cracks or cuts, i.e., the size of the initial system matrix does not change. This allows our method to scale to high-resolution models with very little extra computational overhead. The mesh that is rendered for visualization is the same as the computational mesh initially. This mesh, however, needs to be split and the fracture surfaces have to be reconstructed for rendering the fracture. This does not affect the computational mesh.

It has been shown in earlier work that the nodal forces of a discretized hyper-elastic FEM system can be completely represented in terms of a function of strain energy density along the edges of the induced graph [Reddy and Srinivasa, 2015]. We use this fact and follow prior work in structural mechanics [Khodabakhshi et al., 2016] to define a purely edge-based damage variable, which describes the extent of the damage for any edge in the graph. We build over prior work to reformulate the strain energy density of damaged elements in 3D volumetric object meshes and then simulate the independent movement of the fractured pieces. Further in our model, we can generate and control the diffusion of cracks into the object by imposing a non-local fracture criterion.

We demonstrate the efficacy and robustness of our method by simulating realistic examples of both brittle and ductile fractures. Our method's ability to handle fractures of a variety of materials ranging from stiff glass to squishy jello makes it useful for a wide range of applications.

Our contributions from this chapter are summarized as follows:

- We propose a novel remeshing-free, graph-based, computationally efficient and robust

FEM solution for fracture simulation of 3D models.

- We present a generalized version of a graph-based FEM algorithm that can simulate the fracture of materials with both linear and non-linear strain energy density.

- We theoretically prove that hyper-elastic strain energy density can be represented in closed form as a function of the length of the edges of any object mesh.

- We extensively validate the correctness of our model against real-world structural mechanics experiments.

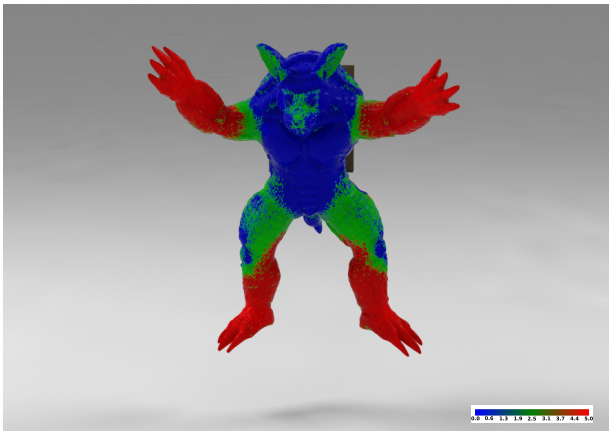
- We demonstrate via examples that our method serves as a unified FEM framework for simulating both brittle and ductile fracture.

The rest of the chapter is organized as follows. In Section 4.2 we first describe the theoretical formulation of graph-based FEM, and then detail the technical aspects of fracture via crack initiation and crack propagation. We discuss the implementation details about the visualization of fractured material, collision handling and the complete algorithm for our model in Section 4.4. Next, we present results for various kinds of materials undergoing fracture, generated using our method in Section 4.5. To validate our method and check its correctness, we compare our results with real-world benchmark fracture experiments and existing fracture simulation techniques in the literature. Finally, we conclude our chapter by discussing the limitations of our work and future directions.

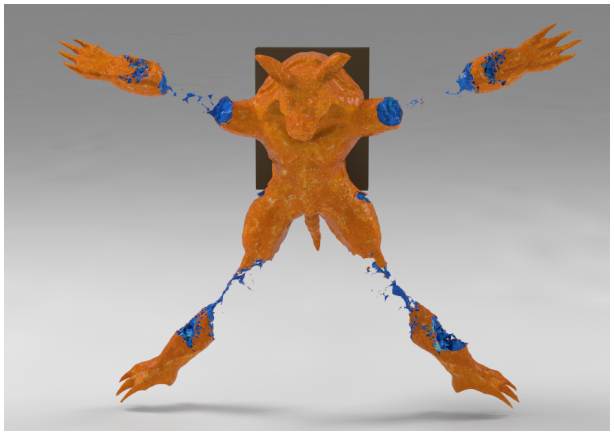
4.2 Governing Methods

We derive the formulation of graph-based FEM from the theory of classical FEM, in brief. Then we delve into the details of fracture generation using graph-based FEM. In our work, we always use tetrahedral finite elements with a linear basis for domain discretization.

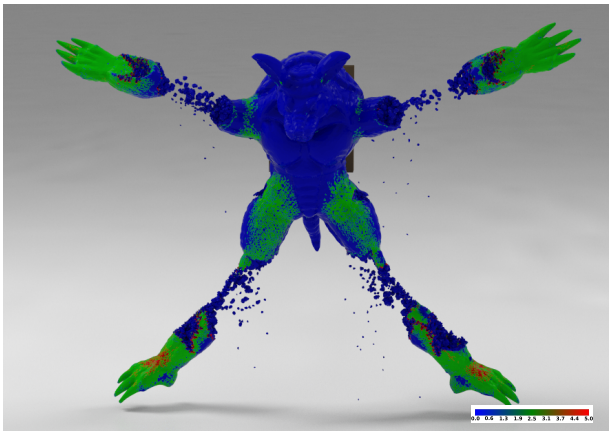
(a) The model just prior to fracture.



(b) Strain profile just prior to fracture.



(c) Model post fracture.



(d) Strain profile post fracture.

Figure 4.2: Here we show frames from a simulation where we rip off the limbs of a jello armadillo. In the strain profiles, the red-to-blue colour gradient denotes the highest to lowest strain values. The model fractures where the strain is highest prior to fracture. Post fracture, the strain drops.

4.2.1 Graph-based FEM

Let the hyper-elastic strain energy density of a tetrahedral element, Δ_e, be $\mathbf{\Psi}^e$ and $\mathbf{f}_{e_i}^{int}$ be the internal elastic force acting on the i^{th} node of the element. Now it can be proved [Reddy and Srinivasa, 2015] that nodal internal elastic forces of Δ_e can always be decomposed along the edge vectors connecting the nodes of Δ_e. The statement can be mathematically formulated as

$$\mathbf{f}_{e_i}^{int} = V_e \frac{\partial \mathbf{\Psi}^e}{\partial \mathbf{r}_i^e} = 2V_e \sum_{\substack{j=1 \\ j \neq i}}^{n_e} \frac{\partial \mathbf{\Psi}^e}{\partial \left(d_{ij}^e\right)^2} d_{ij}^e \hat{\mathbf{d}}_{ij}^e \tag{4.1}$$

where $\mathbf{r}_i^e$ is the world position vector for node i of element Δ_e. The parameter $d_{ij}^e = ||\mathbf{r}_i^e - \mathbf{r}_j^e||$ denotes the distance between the i^{th} and j^{th} nodes of Δ_e and $\hat{\mathbf{d}}_{ij}^e$ is unit vector along d_{ij}^e. Moreover, V_e and n_e represent the volume and the number of nodes of Δ_e respectively. In Equation 4.1 we can get to the rightmost term from the middle term by a few algebraic manipulations. We first take derivatives with respect to squared edge lengths and then apply a chain rule. Please check Appendix A for a more detailed derivation. It can be seen from Equation 4.1 that the magnitude of the nodal internal elastic forces depends only on the derivative of the strain energy density of the element, i.e., on $\left[\frac{\partial \mathbf{\Psi}^e}{\partial \left(d_{ij}^e\right)^2} d_{ij}^e\right]$. Additionally, the directions of the force are along its edges, i.e., along $\hat{\mathbf{d}}_{ij}^e$.

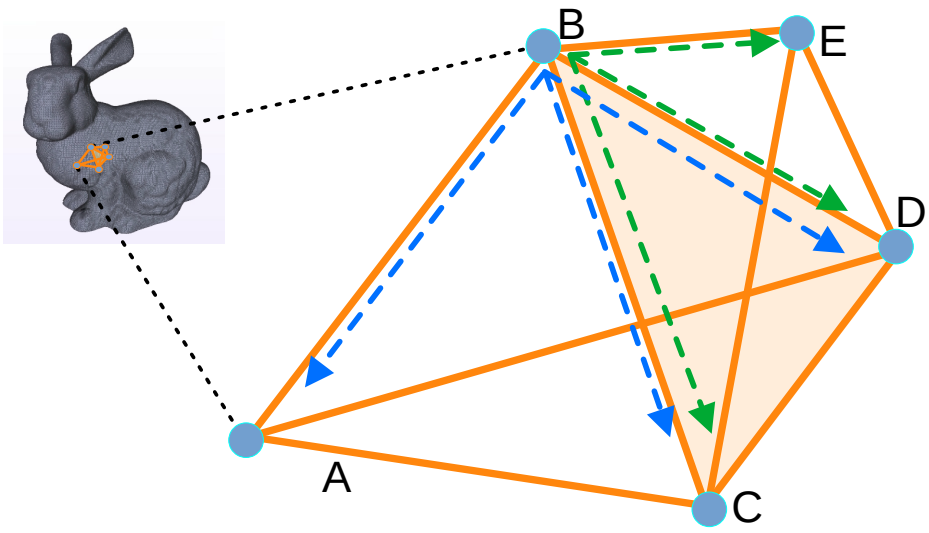


Figure 4.3: Nodal force distribution along the edges for graph-based FEM.

Figure 4.3 shows two tetrahedral elements, $\Delta_e ABDC$ and $\Delta_e BEDC$, which share a common face. These can be thought of as being a part of a larger object mesh. We assume that the overall mesh is deformed and each node in the mesh develops internal elastic force. Now according to Equation 4.1, internal elastic force on the node B for $\Delta_e ABDC$ can be written along the edges of $\Delta_e ABDC$ as $\mathbf{f}^B_{\Delta_e ABDC} = \mathbf{f}^{BC}_{\Delta_e ABDC} + \mathbf{f}^{BA}_{\Delta_e ABDC} + \mathbf{f}^{BD}_{\Delta_e ABDC}$ (shown as dotted blue lines). Similarly, internal elastic force on the node B for $\Delta_e BEDC$ can be written along the edges of $\Delta_e BEDC$ as $\mathbf{f}^B_{\Delta_e BEDC} = \mathbf{f}^{BC}_{\Delta_e BEDC} + \mathbf{f}^{BD}_{\Delta_e BEDC} + \mathbf{f}^{BE}_{\Delta_e BEDC}$ (shown as dotted green lines). The distribution of nodal internal elastic forces for the entire mesh can be computed similarly.

However, for Equation 4.1 to hold, we need to show that hyper-elastic strain energy density $\mathbf{\Psi}^e$ can be completely represented using just d^e_{ij}. We prove this in the next section.

4.2.2 Proof for Edge Length Dependence of Strain Energy Density

Along with the Right Cauchy-Green deformation tensor $\mathbf{C}$ and the three invariants I_C, II_C and III_C, let us introduce two more well-known anisotropic invariants [Weiss et al., 1996]

$$\begin{aligned} IV_C &= \mathbf{a}^T \mathbf{C}\mathbf{b} \\ V_C &= \mathbf{a}^T \mathbf{C}^T \mathbf{C}\mathbf{b} \end{aligned} \tag{4.2}$$

where $\mathbf{a}$, $\mathbf{b}$ are constant anisotropic fiber directions and they may or may not be equal to each other. Hyper-elastic energy density can generally be represented using a subset of these five invariants as $\mathbf{\Psi}^e = f(I_C, II_C, III_C, IV_C, V_C)$. Therefore, in order to show that the hyper-elastic strain energy density can be expressed as a function of the length of edges of a mesh, it is sufficient to show that every element of the set of invariants can be expressed in closed form using only the length of the edges of the mesh in a graph-based FEM setting.

Theorem 4.1. *Every element of the set of invariants* $\mathbf{I}_V = \{I_C, II_C, III_C, IV_C, V_C\}$ *can be expressed in closed form using only the length of the edges of a mesh used in FEM.*

In this prove we show that the Right Cauchy-Green deformation tensor, $\mathbf{C}$, can be expresses in terms the length of the edges in closed form. As all the invariants are derived from $\mathbf{C}$, the proof automatically follows. We present the complete proof of the theorem in Appendix E.1.

4.2.3 Fracture with Graph-based FEM

In this section, we present our model for edge-based fracture derived from Graph-based FEM. We take advantage of the fact that the stress of an element is always a function of hyper-elastic strain energy density. We can find normal stress in the direction of the edges of a tetrahedral element by using a suitable transformation. Next, depending on the state of the edge i.e., broken or unbroken, we manipulate the normal stress along the edges to degrade the strain energy density for a damaged element. Finally, using this updated strain energy density, we recalculate the elastic forces and tangent stiffness matrix for fracture simulation.

Using chain rule of differentiation and deformation gradient, we can rewrite the Equation 4.1 as

$$\mathbf{f}_{e_i}^{int} = 2V_e \sum_{\substack{j=1 \\ j\neq i}}^{n_e} \sum_{k=1}^{3} \sum_{l=1}^{3} \left[\frac{\partial \mathbf{\Psi}^e}{\partial \left[\mathbf{F}\right]_{kl}} \frac{\partial \left[\mathbf{F}\right]_{kl}}{\partial \left(d_{ij}^e\right)^2} \right] d_{ij}^e \hat{\mathbf{d}}_{ij}^e \tag{4.3}$$

where $\dfrac{\partial \mathbf{\Psi}^e}{\partial \left[\mathbf{F}\right]_{kl}}$ are components of first Piola-Kirchhoff stress tensor of Δ_e. Thus, the edge-based elastic forces can be completely represented as a function of stress in Δ_e. This stress in turn depends on the derivative of element strain energy density, $\mathbf{\Psi}^e$ w.r.t. the deformation gradient $\mathbf{F}$. Therefore, by manipulating the stress components properly we can change the internal elastic forces and in turn simulate the fracture of an element.

Let the rectangular Cartesian components of the Piola-Kirchhoff stress tensor, $\boldsymbol{\sigma}_c^e$, be denoted as

$$\boldsymbol{\sigma}_c^e = \begin{bmatrix} \sigma_{xx}^e & \sigma_{yy}^e & \sigma_{zz}^e & \sigma_{xy}^e & \sigma_{xz}^e & \sigma_{yz}^e \end{bmatrix} \tag{4.4}$$

The set of normal stress along the edges is represented as

$$\boldsymbol{\sigma}_d^e = [\sigma_{12}^e \;\; \sigma_{13}^e \;\; \sigma_{14}^e \;\; \sigma_{23}^e \;\; \sigma_{24}^e \;\; \sigma_{34}^e] \tag{4.5}$$

where σ_{ij}^e represents the normal stress along the edge formed by nodes i and j, in Equation 4.5.

Then the transformation of Cartesian stress to normal stress in the direction of an edge can be formulated as below [Reddy and Srinivasa, 2015][Khodabakhshi et al., 2016]

$$\begin{aligned}\sigma_{ij}^{e} = \sigma_{xx}^{e}\cos^2\phi_x + \sigma_{yy}^{e}\cos^2\phi_y + \sigma_{zz}^{e}\cos^2\phi_z \\ + \sigma_{xy}^{e}\cos\phi_x\cos\phi_y + \sigma_{xz}^{e}\cos\phi_x\cos\phi_z + \sigma_{yz}^{e}\cos\phi_y\cos\phi_z\end{aligned} \tag{4.6}$$

Assuming that $\hat{\mathbf{d}}_{ij}^{e}, \hat{\mathbf{x}}, \hat{\mathbf{y}}$ and $\hat{\mathbf{z}}$ denote unit vectors along the edge formed by nodes i and j of the tetrahedral element Δ_e, x-axis, y-axis and z-axis respectively, the following relations hold true

$$\cos\phi_x = \hat{\mathbf{d}}_{ij}^{e}\cdot\hat{\mathbf{x}},\ \ \cos\phi_y = \hat{\mathbf{d}}_{ij}^{e}\cdot\hat{\mathbf{y}},\ \ \cos\phi_z = \hat{\mathbf{d}}_{ij}^{e}\cdot\hat{\mathbf{z}} \tag{4.7}$$

Using Equation 4.6, we can write the final relation between two sets of stresses as

$$\begin{aligned}&\begin{bmatrix}\sigma_{12}^{e} & \sigma_{13}^{e} & \sigma_{14}^{e} & \sigma_{23}^{e} & \sigma_{24}^{e} & \sigma_{34}^{e}\end{bmatrix}^T \\ &= \mathbf{T}\begin{bmatrix}\sigma_{xx}^{e} & \sigma_{yy}^{e} & \sigma_{zz}^{e} & \sigma_{xy}^{e} & \sigma_{xz}^{e} & \sigma_{yz}^{e}\end{bmatrix}^T\end{aligned} \tag{4.8}$$

where, for convenience of notation, if we denote $\cos\phi$ as γ_ϕ, then

$$\mathbf{T} = \begin{bmatrix}
\gamma_{\phi_x^{12}}^2 & \gamma_{\phi_y^{12}}^2 & \gamma_{\phi_z^{12}}^2 & \gamma_{\phi_x^{12}}\gamma_{\phi_y^{12}} & \gamma_{\phi_x^{12}}\gamma_{\phi_z^{12}} & \gamma_{\phi_y^{12}}\gamma_{\phi_z^{12}} \\
\gamma_{\phi_x^{13}}^2 & \gamma_{\phi_y^{13}}^2 & \gamma_{\phi_z^{13}}^2 & \gamma_{\phi_x^{13}}\gamma_{\phi_y^{13}} & \gamma_{\phi_x^{13}}\gamma_{\phi_z^{13}} & \gamma_{\phi_y^{13}}\gamma_{\phi_z^{13}} \\
\gamma_{\phi_x^{14}}^2 & \gamma_{\phi_y^{14}}^2 & \gamma_{\phi_z^{14}}^2 & \gamma_{\phi_x^{14}}\gamma_{\phi_y^{14}} & \gamma_{\phi_x^{14}}\gamma_{\phi_z^{14}} & \gamma_{\phi_y^{14}}\gamma_{\phi_z^{14}} \\
\gamma_{\phi_x^{23}}^2 & \gamma_{\phi_y^{23}}^2 & \gamma_{\phi_z^{23}}^2 & \gamma_{\phi_x^{23}}\gamma_{\phi_y^{23}} & \gamma_{\phi_x^{23}}\gamma_{\phi_z^{23}} & \gamma_{\phi_y^{23}}\gamma_{\phi_z^{23}} \\
\gamma_{\phi_x^{24}}^2 & \gamma_{\phi_y^{24}}^2 & \gamma_{\phi_z^{24}}^2 & \gamma_{\phi_x^{24}}\gamma_{\phi_y^{24}} & \gamma_{\phi_x^{24}}\gamma_{\phi_z^{24}} & \gamma_{\phi_y^{24}}\gamma_{\phi_z^{24}} \\
\gamma_{\phi_x^{34}}^2 & \gamma_{\phi_y^{34}}^2 & \gamma_{\phi_z^{34}}^2 & \gamma_{\phi_x^{34}}\gamma_{\phi_y^{34}} & \gamma_{\phi_x^{34}}\gamma_{\phi_z^{34}} & \gamma_{\phi_y^{34}}\gamma_{\phi_z^{34}}
\end{bmatrix} \tag{4.9}$$

The matrix $\mathbf{T}$ is calculated at every timestep and it is invertible for non-degenerate tetrahedral elements, i.e., elements with non-zero volume.

The criteria for fracture in graph-based FEM is that if the weighted average of normal stress along the direction of any edge exceeds the critical stress threshold σ_{thres}^{e}, a crack is to be formed on that edge. First, we explain the method to calculate the weighted average of normal stress. Let $\mathbf{r}_{cen}$ be the position of the centroid of any tetrahedron of interest, Δ_{cen}, in the mesh at a particular time step t' during simulation. The weighted average of normal stress for Δ_{cen} can then be calculated as

$$\boldsymbol{\sigma}_{cen}^{e} = \sum_{||\mathbf{r}-\mathbf{r}_{cen}||\leq R_d} \omega(\mathbf{r}-\mathbf{r}_{cen})\boldsymbol{\sigma}_{c}^{e}(\mathbf{r}) \tag{4.10}$$

where ω is a weight kernel centered at $\mathbf{r}_{cen}$ and R_d is support of the kernel. In our current implementation, we use a simple linear average kernel for ω to allow easier calculation. Other kernels

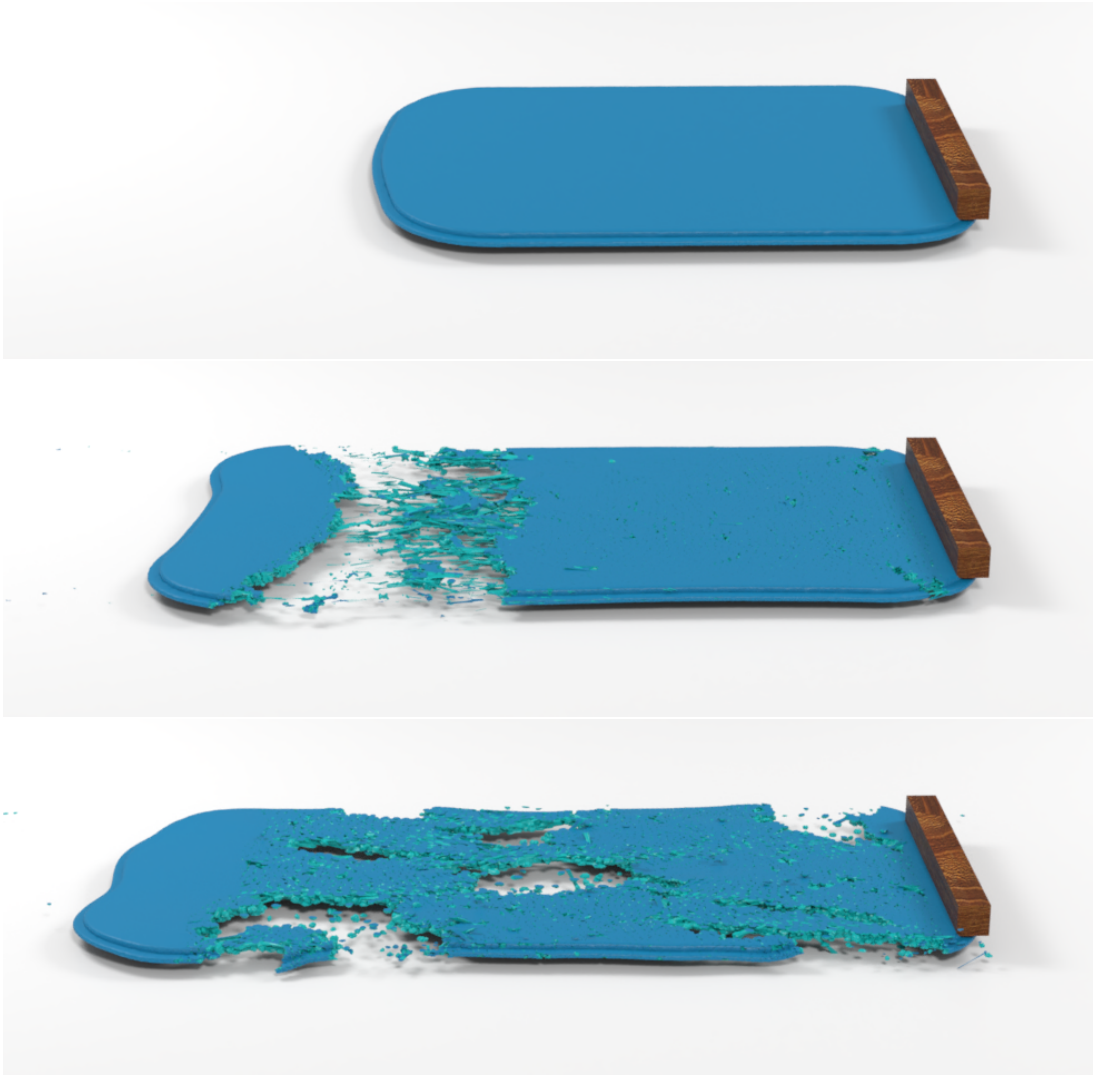


Figure 4.4: Effect of kernel size R_d on fracture. We show the tearing of a PVC doormat (top) with different sizes of kernel support ($R_d = 0$ in the middle, $R_d = 5$ at the bottom). As the size of kernel support increases, the cracks become more diffused over the object mesh.

e.g., a Gaussian kernel, can also be used. The exact diffusion fracture pattern (explained in the next paragraph) will be different for different kernels. Vector $\mathbf{r}$ and tensor $\boldsymbol{\sigma}_c^e(\mathbf{r})$ respectively refer to the position of centroid and Piola-Kirchhoff stress tensor (PK1) of any tetrahedron (see Section 2.1.1.1) residing inside the kernel support R_d, at the same time step t'.

Using Equations 4.8 and 4.10 the fracture criteria can, therefore, be defined as [Khodabakhshi et al., 2019]

$$\sigma_{ij}^{e^*} = \left[\mathbf{T}\,[\boldsymbol{\sigma}_{cen}^e]^T\right]_{ij} \geq \sigma_{thres}^e \tag{4.11}$$

where $\sigma_{ij}^{e^*}$ is a component of the weighted average of normal stress along the edge formed by nodes i and j. As the support of kernel, R_d, increases, the cracks become more diffused

i.e., they spread more inside the object [Khodabakhshi et al., 2019]. For $R_d = 0$, the fracture criterion depends only on the stress value of the tetrahedron of interest, Δ_{cen}, resembling a local fracture. The effect of kernel size R_d on fracture is shown in Figure 4.4. The top image shows an undamaged doormat model. In the middle and bottom images, the doormat has been torn apart by applying the same force at the free left end. As evident from the figure, in the middle image, where $R_d = 0$, the fracture is localized in a nearly horizontal tear at one place. On the contrary, in the image at the bottom, where $R_d = 5$, the cracks are propagated over the entire body of the doormat, in a diffused pattern.

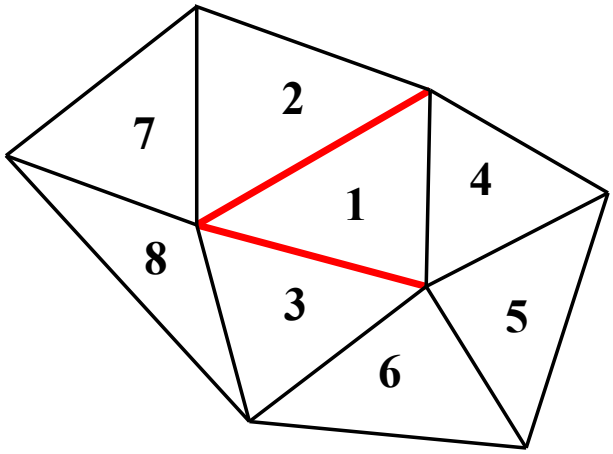


Figure 4.5: Fractured edges shared multiple elements.

Now for each edge of the mesh, we maintain a binary damage variable, $\zeta \in \{0,1\}$. A value of $\zeta = 0$ corresponds to an unbroken edge, while $\zeta = 1$ implies a crack is formed on the edge.

$$\zeta = \begin{cases} 0 & \text{if } \sigma_{ij}^{e^*} < \sigma_{thres} \\ 1 & \text{if } \sigma_{ij}^{e^*} \geq \sigma_{thres} \end{cases} \tag{4.12}$$

The number of broken edges in a tetrahedral element may vary from zero to six. Once an edge is broken, it will not get repaired in subsequent iterations and all the tetrahedra that share the broken edge will be considered damaged. As an example, a shared edge fracture is depicted in Figure 4.5, using a simplified, two-dimensional example. Two edges in the mesh, coloured red, are broken. These broken edges are shared by triangles 1 (two edges), 2 (one edge) and 3 (one edge). In the subsequent iterations, all three triangular elements are considered damaged with varying degrees of damage. Whenever an edge, e_{br}, gets broken, the corresponding nor-

mal stress of that edge becomes zero for all elements sharing that edge. Using Equations 4.8 and 4.12, we can write

$$\begin{aligned}&\begin{bmatrix}\sigma^e_{12_{frac}} & \sigma^e_{13_{frac}} & \sigma^e_{14_{frac}} & \sigma^e_{23_{frac}} & \sigma^e_{24_{frac}} & \sigma^e_{34_{frac}}\end{bmatrix}^T \\ &= \text{diag}\begin{bmatrix}\zeta_{12} & \zeta_{13} & \zeta_{14} & \zeta_{23} & \zeta_{24} & \zeta_{34}\end{bmatrix} \\ &\begin{bmatrix}\sigma^e_{12} & \sigma^e_{13} & \sigma^e_{14} & \sigma^e_{23} & \sigma^e_{24} & \sigma^e_{34}\end{bmatrix}^T \quad \forall e_{br} \in \Delta_e\end{aligned} \tag{4.13}$$

where ζ_{ij} and $\sigma^e_{ij_{frac}}$ are damage variables and stress after fracture along the edge formed by nodes i and j. In matrix form Equation 4.13 is written as

$$\left[\boldsymbol{\sigma}^e_{d_{frac}}\right]^T = \boldsymbol{\zeta}\,[\boldsymbol{\sigma}^e_d]^T = \boldsymbol{\zeta}\mathbf{T}\,[\boldsymbol{\sigma}^e_c]^T \tag{4.14}$$

where $\mathbf{T}$ is transformation matrix from Equation 4.9. Projecting back to Cartesian space, rectangular Cartesian components, $\boldsymbol{\sigma}^e_{c_{frac}}$ of the damaged Piola-Kirchhoff stress tensor can be written as

$$\left[\boldsymbol{\sigma}^e_{c_{frac}}\right]^T = \mathbf{T}^{-1}\left[\boldsymbol{\sigma}^e_{d_{frac}}\right]^T = \mathbf{T}^{-1}\boldsymbol{\zeta}\mathbf{T}\,[\boldsymbol{\sigma}^e_c]^T \tag{4.15}$$

Graph-based FEM may seem similar to mass-spring systems and peridynamics, but it has key differences as explained below.

4.2.3.1 Difference Between Graph-Based FEM and Mass-Spring Model

It can be tempting to assume that our model is similar to mass-spring or peridynamics-based fracture model [Levine et al., 2015], reformulated as FEM. However, there exist two crucial differences.

First, in peridynamics or mass-spring model the total energy density of an element with n_e number of nodes, can be formulated as

$$\boldsymbol{\Psi}^e = \sum_{i=1}^{n_e}\sum_{i>j}^{n_e}\boldsymbol{\Psi}^e_{ij} \tag{4.16}$$

That is, the strain energy density is just the sum of the strain energies along each edge without any cross-dependence. But in the case of the graph-based FEM method, strain energy density depends on the distance between all the edges of the element, which is not just a simple sum. This is also observed by Reddy and Srinivasa [2015].

Second, in the case of peridynamics, force at any point depends on points far away from it. Therefore it is a completely non-local dynamics, whereas, in graph-based FEM, force values of an element are derived locally from the same element stresses.

4.2.4 Fractured Strain Energy Density: Linear Elasticity

In graph-based FEM, instead of remeshing the initial mesh, we update the system equation of fractured mesh using a reformulation of elastic energy density to simulate fracture. In this section, we present the detailed derivation of hyper-elastic strain energy density for graph-based FEM in the case of linear elasticity problems. First, we derive it for undamaged conditions. The expression for the damaged condition is a simple extension of that.

4.2.4.1 Undamaged Condition

In linear elasticity the strain energy density for a tetrahedral element, Δ_e, can be written in terms of linear Cartesian strain $\boldsymbol{\varepsilon}_c^e$ and stress $\boldsymbol{\sigma}_c^e$ vectors as below.

$$\boldsymbol{\Psi}^e = \frac{1}{2}\boldsymbol{\sigma}_c^e \cdot \boldsymbol{\varepsilon}_c^e = \frac{1}{2}\boldsymbol{\varepsilon}_c^e \mathbf{E} \cdot \boldsymbol{\varepsilon}_c^e \tag{4.17}$$

where $\boldsymbol{\sigma}_c^e = \boldsymbol{\varepsilon}_c^e \mathbf{E}$ and linear Cartesian strain $\boldsymbol{\varepsilon}_c^e = \frac{1}{2}\left(\mathbf{F} + \mathbf{F}^T\right) - \mathbf{I}$ is expressed in vector format (in Voigt notation).

$$\boldsymbol{\varepsilon}_c^e = \begin{bmatrix} \varepsilon_{xx}^e & \varepsilon_{yy}^e & \varepsilon_{zz}^e & \varepsilon_{xy}^e & \varepsilon_{xz}^e & \varepsilon_{yz}^e \end{bmatrix} \tag{4.18}$$

The symmetric matrix $\mathbf{E}$ denotes elasticity tensor [Müller and Gross, 2004].

For linear elasticity, the strain components along the edges can be written as

$$\varepsilon_{ij}^e = \frac{l_{ij}}{L_{ij}} \tag{4.19}$$

where L_{ij} and l_{ij} denote the original length and increase in length of the edge formed by nodes i and j. In vector format original length and increase in length can be expressed as below.

$$\begin{aligned} \mathbf{L}_d^e &= \begin{bmatrix} L_{12}^e & L_{13}^e & L_{14}^e & L_{23}^e & L_{24}^e & L_{34}^e \end{bmatrix} \\ \mathbf{l}_d^e &= \begin{bmatrix} l_{12}^e & l_{13}^e & l_{14}^e & l_{23}^e & l_{24}^e & l_{34}^e \end{bmatrix} \end{aligned} \tag{4.20}$$

For a tetrahedral element edge-based normal strain from Equation 4.19 can be written in matrix form as

$$\boldsymbol{\varepsilon}_d^e = \begin{bmatrix} \varepsilon_{12}^e & \varepsilon_{13}^e & \varepsilon_{14}^e & \varepsilon_{23}^e & \varepsilon_{24}^e & \varepsilon_{34}^e \end{bmatrix} \tag{4.21}$$

For graph-based FEM we need to write down all the parameters e.g., strain energy density and internal force, using edge-based variables. Thus, transforming edge-based normal strain to

Cartesian strain, Equation 4.17 can be rewritten as

$$\boldsymbol{\Psi}^e = \frac{1}{2}\boldsymbol{\varepsilon}_c^e \mathbf{E} \cdot \boldsymbol{\varepsilon}_c^e = \frac{1}{2}\left[\mathbf{T}^{-1}\left[\boldsymbol{\varepsilon}_d^e\right]^T\right]^T \mathbf{E} \cdot \left[\mathbf{T}^{-1}\left[\boldsymbol{\varepsilon}_d^e\right]^T\right]^T \tag{4.22}$$

Now the nodal internal force vector $\mathbf{f}_e^{int}$ of element Δ_e of volume V_e can be written as

$$\begin{aligned}
\mathbf{f}_e^{int} &= V_e \frac{\partial \boldsymbol{\Psi}^e}{\partial \mathbf{u}_e} = V_e \frac{\partial \boldsymbol{\Psi}^e}{\partial \boldsymbol{\varepsilon}_c^e}\frac{\partial \boldsymbol{\varepsilon}_c^e}{\partial \mathbf{u}_e} = V_e \boldsymbol{\sigma}_c^e \frac{\partial \boldsymbol{\varepsilon}_c^e}{\partial \mathbf{u}_e} \\
&= V_e \boldsymbol{\sigma}_c^e \frac{\partial \boldsymbol{\varepsilon}_c^e}{\partial \mathbf{l}_d^e}\frac{\partial \mathbf{l}_d^e}{\partial \mathbf{u}_e} \text{ (Using Equation 4.20)} \\
&= V_e \boldsymbol{\sigma}_c^e \mathbf{A}_2^T \mathbf{A}_1^T = V_e \boldsymbol{\sigma}_c^e \mathbf{B} = V_e \boldsymbol{\varepsilon}_c^e \mathbf{E}\mathbf{B}
\end{aligned} \tag{4.23}$$

where $\mathbf{A}_1 = \left[\frac{\partial \mathbf{l}_d^e}{\partial \mathbf{u}_e}\right]^T$, $\mathbf{A}_2 = \left[\frac{\partial \boldsymbol{\varepsilon}_c^e}{\partial \mathbf{l}_d^e}\right]^T$ and $\mathbf{B}^T = \mathbf{A}_1\mathbf{A}_2$. The detailed expressions for $\mathbf{A}_1$ and $\mathbf{A}_2$ are given in Appendix B.

Similarly, we can formulate the tangent stiffness matrix in the graph-based FEM paradigm as below.

$$\begin{aligned}
\mathbf{k}_e &= \frac{\partial \mathbf{f}_e^{int}}{\partial \mathbf{u}_e} = V_e \left[\frac{\partial \boldsymbol{\varepsilon}_c^e}{\partial \mathbf{u}_e}\right]^T \left[\frac{\partial^2 \boldsymbol{\Psi}^e}{\partial \boldsymbol{\varepsilon}_c^e \partial \boldsymbol{\varepsilon}_c^e}\right] \left[\frac{\partial \boldsymbol{\varepsilon}_c^e}{\partial \mathbf{u}_e}\right] \\
&= V_e \left[\frac{\partial \boldsymbol{\varepsilon}_c^e}{\partial \mathbf{l}_d^e}\frac{\partial \mathbf{l}_d^e}{\partial \mathbf{u}_e}\right]^T \mathbf{E} \left[\frac{\partial \boldsymbol{\varepsilon}_c^e}{\partial \mathbf{l}_d^e}\frac{\partial \mathbf{l}_d^e}{\partial \mathbf{u}_e}\right] \\
&= V_e \left[\frac{\partial \mathbf{l}_d^e}{\partial \mathbf{u}_e}\right]^T \left[\frac{\partial \boldsymbol{\varepsilon}_c^e}{\partial \mathbf{l}_d^e}\right]^T \mathbf{E} \left[\frac{\partial \boldsymbol{\varepsilon}_c^e}{\partial \mathbf{l}_d^e}\right] \left[\frac{\partial \mathbf{l}_d^e}{\partial \mathbf{u}_e}\right] \\
&= V_e \mathbf{A}_1 \mathbf{A}_2 \mathbf{E} \mathbf{A}_2^T \mathbf{A}_1^T = V_e \mathbf{B}^T \mathbf{E}\mathbf{B}
\end{aligned} \tag{4.24}$$

Note that both internal elastic force $\mathbf{f}_e^{int}$ and tangent stiffness matrix $\mathbf{k}_e$ match the formulation of conventional FEM.

4.2.4.2 Damaged Condition

When an element gets fractured the strain energy density from Equation 4.17 and Equation 4.22 can be written as

$$\begin{aligned}
\boldsymbol{\Psi}^e &= \frac{1}{2}\boldsymbol{\sigma}_{c_{frac}}^e \cdot \boldsymbol{\varepsilon}_{c_{frac}}^e = \frac{1}{2}\boldsymbol{\varepsilon}_{c_{frac}}^e \mathbf{E} \cdot \boldsymbol{\varepsilon}_{c_{frac}}^e \\
\text{Using } \left[\boldsymbol{\sigma}_{c_{frac}}^e\right]^T &= \mathbf{T}^{-1}\boldsymbol{\zeta}\mathbf{T}\left[\boldsymbol{\sigma}_c^e\right]^T \\
&= \frac{1}{2}\left[\mathbf{T}^{-1}\boldsymbol{\zeta}\mathbf{T}\left[\boldsymbol{\varepsilon}_c^e\right]^T\right]^T \mathbf{E} \cdot \left[\mathbf{T}^{-1}\boldsymbol{\zeta}\mathbf{T}\left[\boldsymbol{\varepsilon}_c^e\right]^T\right]^T
\end{aligned} \tag{4.25}$$

Now following the same line of arguments as presented in Equation 4.23 and Equation 4.24, the internal force and tangent stiffness matrix for fractured elements can be repre-

sented as

$$\mathbf{f}^{int}_{e_{frac}} = V_e \boldsymbol{\varepsilon}^e_c \mathbf{T}^T \boldsymbol{\zeta} \mathbf{T}^{-T} \mathbf{E} \mathbf{T}^{-1} \boldsymbol{\zeta} \mathbf{T} \mathbf{B} \tag{4.26}$$

$$\mathbf{k}_{e_{frac}} = V_e \mathbf{B}^T \mathbf{T}^T \boldsymbol{\zeta} \mathbf{T}^{-T} \mathbf{E} \mathbf{T}^{-1} \boldsymbol{\zeta} \mathbf{T} \mathbf{B} \tag{4.27}$$

4.2.5 Fractured Strain Energy Density: Non-linear Elasticity

Non-linear elasticity demands a different reformulation of the elastic energy density. In this section, we present two different approaches for this reformulation that work in the case of non-linear elasticity. There are also some approaches we tried that did not produce the correct result. These are instructive for understanding and are presented in Appendix C for completeness.

4.2.5.1 First Approach — Linearization

Our first approach is based on the linearization of non-linear strain energy density. We reparameterize the material parameters μ and λ to reproduce the stress tensor, $\boldsymbol{\sigma}^e_c$, of linear elasticity according to Hooke's law.

$$\boldsymbol{\sigma}^e_c = 2\mu_{\text{Lamé}} \boldsymbol{\varepsilon}^e_c + \lambda_{\text{Lamé}} \text{trace}\,(\boldsymbol{\varepsilon}^e_c)\,\mathbf{I} \tag{4.28}$$

where coefficients $\mu_{\text{Lamé}}$ and $\lambda_{\text{Lamé}}$ are Lamé parameters in linear elasticity. The parameter $\boldsymbol{\varepsilon}^e_c = 1/2(\mathbf{F} + \mathbf{F}^T) - \mathbf{I}$ is linear strain.

Given Young's modulus ($\mathbf{Y}$) and Poisson's ratio (ν), Lamé parameters can be defined as

$$\begin{aligned} \mu_{\text{Lamé}} &= \frac{\mathbf{Y}}{2(1+\nu)} \\ \lambda_{\text{Lamé}} &= \frac{\mathbf{Y}\nu}{(1+\nu)(1-2\nu)} \end{aligned} \tag{4.29}$$

For Neo-Hookean energy density [Smith et al., 2018] in Equation 4.30, if we set $\mu = 4/3\mu_{\text{Lamé}}$ and $\lambda = \lambda_{\text{Lamé}} + 5/6\mu_{\text{Lamé}}$, it becomes consistent [Smith et al., 2018] with Equation 4.28. With these linearized stress and strain values, we can rewrite the linearized Neo-Hookean energy density expression and follow the same line of arguments as presented earlier for fracture simulation.

$$\boldsymbol{\Psi}_{neo} = \frac{\mu}{2}(I_C - 3) + \frac{\lambda}{2}(J - \alpha)^2 - \frac{\mu}{2}\log(I_C + 1) \tag{4.30}$$

where $\alpha = 1 + \frac{\mu}{\lambda} - \frac{\mu}{4\lambda}$.

Similarly for Saint–Venant Kirchhoff energy density given in Equation 4.31, linearization [Kikuuwe et al., 2009] can be done by putting $\mu = \mu_{\text{Lamé}}$ and $\lambda = \lambda_{\text{Lamé}}$. We have simulated fracture using this method on Saint–Venant Kirchhoff energy density in Figure 4.6.

$$\Psi_{stvk} = \frac{\lambda}{8}(I_C - 3)^2 + \frac{\mu}{4}(II_C - 2I_C + 3) \tag{4.31}$$

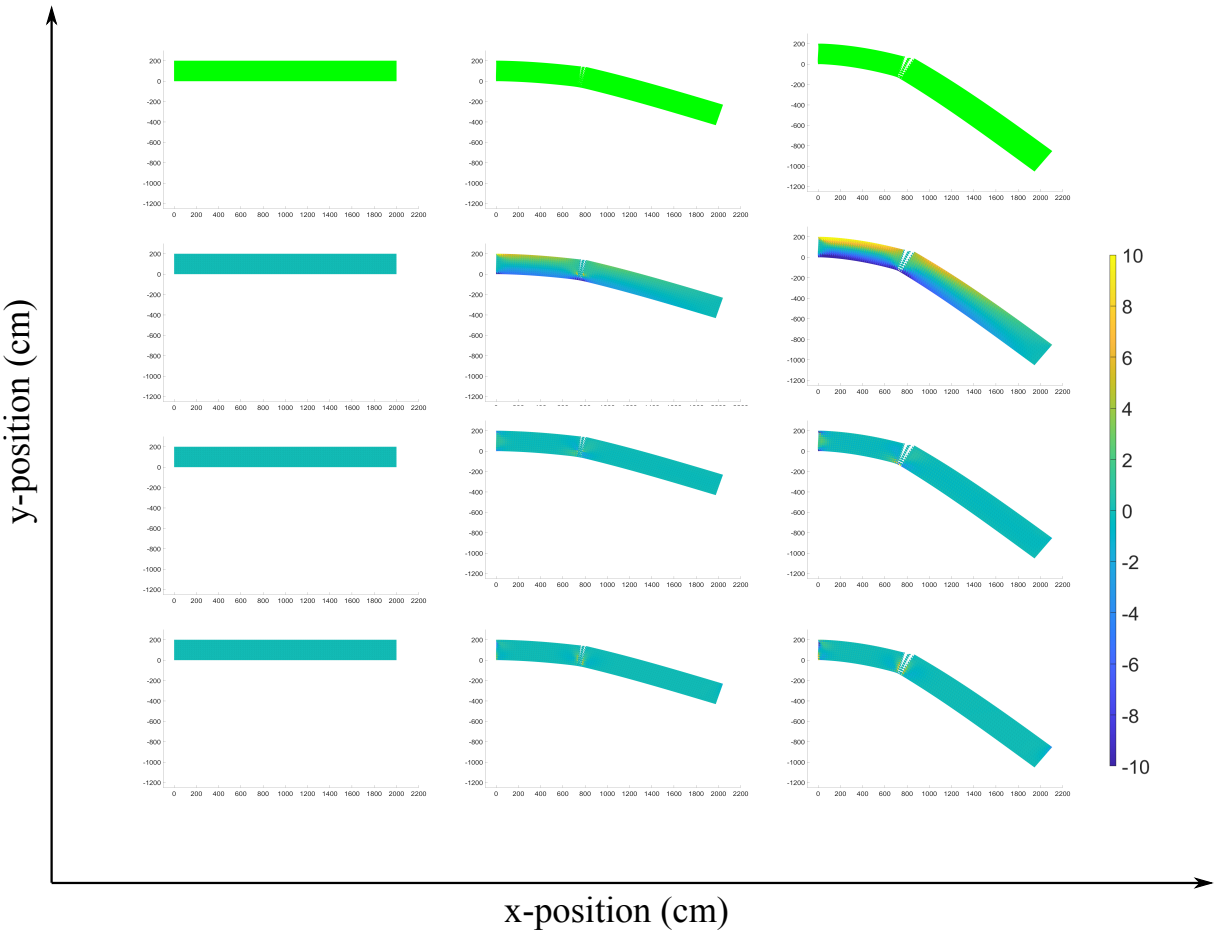


Figure 4.6: Linearization of strain energy density for fracture simulation on a 2D bar (1^{st} row). The corresponding strain profiles are shown in $2^{nd}(\sigma_x)$, $3^{rd}(\sigma_{xy})$ and $4^{th}(\sigma_y)$ row respectively. Positive energy represents the stretching of an object, while negative energy corresponds to its compression.

The derivations for the linearization of these strain energies are given in Appendix D.

The main advantage of using this formulation is that even if an element gets fractured into two or more disjoint fragments, the residual force continues working on the remaining intact edges. However, for linearization operation, we assume infinitesimal linearized strain

tensor $\varepsilon = 1/2(\mathbf{F} + \mathbf{F}^T) - \mathbf{I}$ instead of Lagrangian finite strain tensor. As higher-order terms of the strain tensor are ignored for linearization, it can not capture large deformations without introducing significant artefacts. Thus, compared to non-linear models which use Lagrangian finite strain tensor, it produces more deformation artefacts.

4.2.5.2 Second Approach — Monotonic Degradation

Our second approach is based on the monotonic degradation of non-linear strain energy density. It is common in fracture simulation to degrade the energy density by a monotonic degradation function to allow material separation [Amor et al., 2009] [Miehe et al., 2015] [Wolper et al., 2019]. Thus, using Equation 4.5 and Equation 4.13 the hyper-elastic strain energy density for fractured elements can be defined as

$$\mathbf{\Psi}^e_{frac} = \chi^e \left(\boldsymbol{\sigma}^e_{d_{frac}}, \boldsymbol{\sigma}^e_d \right) \mathbf{\Psi}^e \tag{4.32}$$

The monotonic function $\chi^e \left(\boldsymbol{\sigma}^e_{d_{frac}}, \boldsymbol{\sigma}^e_d \right)$ signifies the fractured hyper-elastic strain energy density as a fraction of original one depending on the number of damaged edges.

$$\chi^e \left(\boldsymbol{\sigma}^e_{d_{frac}}, \boldsymbol{\sigma}^e_d \right) = \frac{\sum\limits_{\substack{i,j=1 \\ i<j}}^{4} |\sigma^e_{ij_{frac}}|}{\sum\limits_{\substack{i,j=1 \\ i<j}}^{4} |\sigma^e_{ij}|} \tag{4.33}$$

where i and j denote nodes of the tetrahedral mesh element, Δ_e. Further, this updated hyper-elastic strain energy density is used to calculate internal force and tangent stiffness matrix for fractured elements. We simulate fracture using this method on Saint–Venant Kirchhoff energy density in Figure 4.7.

The main advantage of this method is that we can use Lagrangian finite strain tensor for non-linear elasticity. Thus using this approach, large deformations can be better simulated without any artefacts contrary to the earlier linearization method. However, in this approach, we assign zero value to the fractured hyper-elastic strain energy density i.e. $\mathbf{\Psi}^e_{frac} = 0$, in Equation 4.32 when an element fractures into two or more disjoint fragments. Thus, in this case, when an element produces two or more disjoint segments, no residual force remains in the intact edges. This is a drawback compared to the previous linearization model where residual force continues working on the remaining intact edges even for disjoint fragments. However, any edge in an

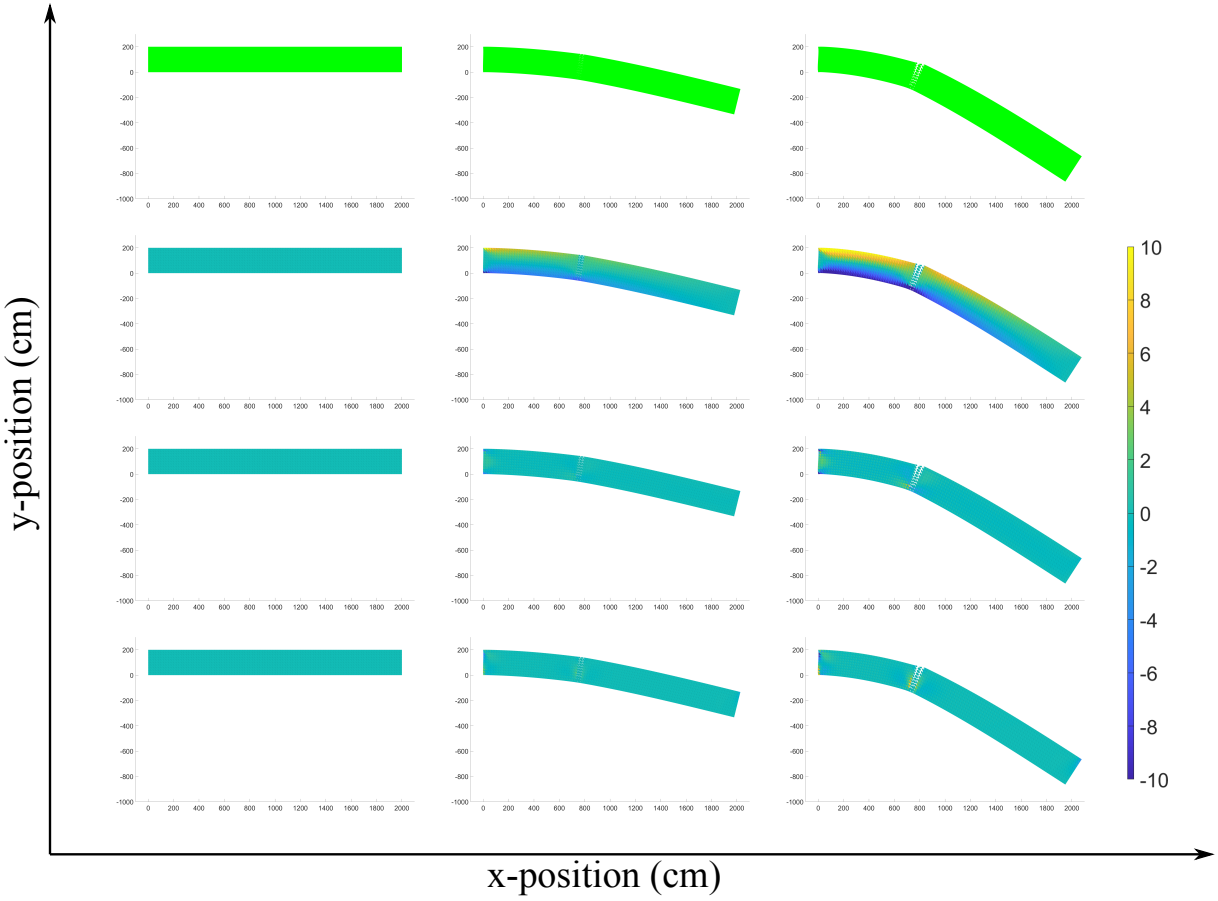


Figure 4.7: Degradation of hyper-elastic strain energy density for fracture simulation on a 2D bar (1^{st} row). The corresponding strain profiles are shown in $2^{nd}(\sigma_x)$, $3^{rd}(\sigma_{xy})$ and $4^{th}(\sigma_y)$ row respectively. Positive energy represents the stretching of an object, while negative energy corresponds to its compression.

object mesh is shared by multiple elements all of which may not be fully damaged. The internal forces from these undamaged elements continue to work on those intact edges and thus, keep the nodes of the undamaged edges together for a fully split tetrahedron.

From Figure 4.6 and Figure 4.7, we can see that there is no significant change in the visual output of the fracture simulation regardless of the reformulation method used. Moreover, irrespective of methods used for updating strain energy density after fracture, both of these approaches use the rich discrete differential geometric approach [Srinivasa, 2021] of graph-based FEM to simulate fracture. From the discussion of the two approaches, we can conclude that if our main objective is to simulate large deformation-based fractures, we should use the second approach. On the other hand, if our goal is to accurately capture the residual stress after fracture for some engineering applications, then the first approach should be preferred.

4.3 Anisotropy in Graph-based FEM

In this section, we delve into the details of simulating anisotropic fracture with graph-based FEM.

4.3.1 Anisotropic Elasticity

Isotropic materials compress or stretch equally in all directions irrespective of the direction of applied force. But many real-world materials have bundles of fibres in some preferred directions, making them stiffer and softer in others. When stretched/compressed, these materials will resist more in the stiffer directions compared to softer directions, before damage sets in. Accordingly, this makes the fracture of anisotropic materials significantly different from the isotropic ones.

In order to model anisotropic materials, we add an anisotropic hyper-elastic strain energy density along with the regular isotropic strain energy density.

$$\mathbf{\Psi}^e_{tot} = \mathbf{\Psi}^e_{aniso} + \mathbf{\Psi}^e_{iso} \tag{4.34}$$

where $\mathbf{\Psi}^e_{tot}$, $\mathbf{\Psi}^e_{iso}$ and $\mathbf{\Psi}^e_{aniso}$ denote total, isotropic and anisotropic strain energy density respectively.

To model anisotropic strain energy density, Kim et al. [2019] defined a new sign-preserving anisotropic invariant

$$I_4 = \mathbf{a}^T \mathbf{S}\mathbf{a} \tag{4.35}$$

where the stretch matrix $\mathbf{S}$ is obtained from the polar decomposition of deformation gradient $\mathbf{F} = \mathbf{RS}$. We use the inversion-safe, anisotropic strain energy density proposed by Kim et al. [2019] as follows.

$$\mathbf{\Psi}^e_{AA} = \frac{\mu}{2}\left(\sqrt{I_5} - \Pi(I_4)\right)^2 \tag{4.36}$$

where $I_5 = \mathbf{a}^T \mathbf{C}\mathbf{a}$ and Π is the signum function. The properties of these invariants are explained previously in Section 2.1.1.3. For a detailed discussion on how this energy density satisfies the condition of inversion safety and stability, please refer to [Kim et al., 2019]. As an example of

fracture simulation after incorporating anisotropy in graph-based FEM, see Figure 4.15. Here we illustrate the fracture of a piece of steak which tears along the anisotropic fibre direction. Also, notice the thread-like stretchy fibres that get revealed when the meat is torn apart.

4.3.2 Theoretical Considerations

Graph-based FEM crucially depends on the fact that all involved strain energies can be represented as a function of the edge lengths of the simulation mesh. This has to be true for anisotropic strain energy density $\mathbf{\Psi}^e_{AA}$, for the above-described formulation to work. We prove that this is indeed so, in the Appendix E.2.

The theorems, proved in the appendix, together give a geometric interpretation of any hyper-elastic strain energy density in terms of the edge length for an undamaged mesh.

We also present a proof sketch in Appendix E.3 to show that the deformation of the fractured/damaged mesh can then be interpreted as smooth map from one Riemannian manifold to another. We believe this interpretation provides a convincing argument in support of the representative capacity of graph-based FEM to model fracture/damage.

4.4 Implementation

Next, We present details of how we create visualizations of the fracture simulation. Following that, we discuss collision detection for fractured pieces. Then we present a complete algorithm for our model of fracture simulation.

4.4.1 Surface Remeshing for Visualization

Our FEM computations are performed on the graph induced by the computational mesh, $\mathscr{M}_c$, that is never remeshed. However, to visualize the fracture we need to split the mesh used for visualization. For that purpose, a separate visualization surface mesh, $\mathscr{M}_v$, is maintained in

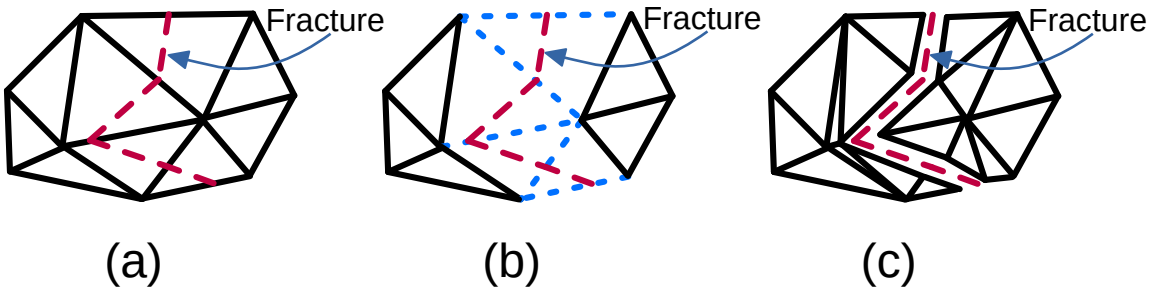


Figure 4.8: (a) Original mesh with a fracture, (b) Computational mesh ($\mathscr{M}_c$), with damaged edges marked in blue, on which system dynamics get evaluated, (c) Mesh for visualization ($\mathscr{M}_v$) is split and the fracture surface is reconstructed.

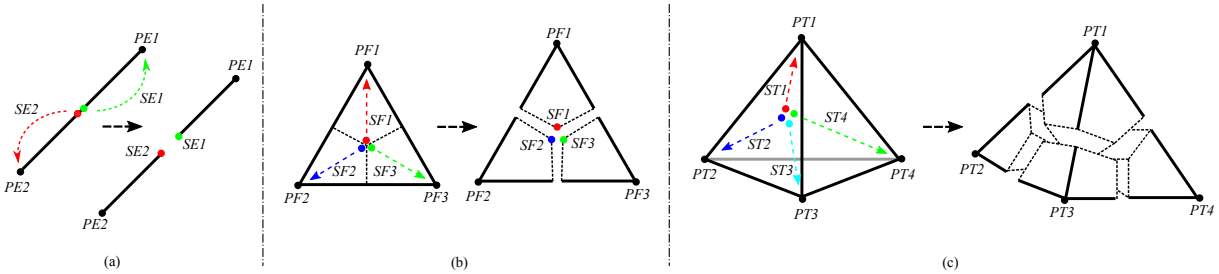


Figure 4.9: Splitting of edge (left), face (middle) and tetrahedron (right) for visualization of fracture. Here, tetrahedron is the computational mesh ($\mathscr{M}_c$) and the outer surfaces, generated due to splitting, are visualization mesh ($\mathscr{M}_v$).

addition to $\mathscr{M}_c$ and is the same as the outer surface of volumetric mesh initially. We clarify this distinction between the $\mathscr{M}_c$ and $\mathscr{M}_v$ meshes using a simple 2D example. Figure 4.8b depicts that when a fracture occurs, the system dynamics is evaluated on $\mathscr{M}_c$ with weakened elements. When we need to render the fractured elements, we split the $\mathscr{M}_v$ mesh that is used for visualization, as shown in Figure 4.8c and reconstruct the fracture surface. Remeshing of $\mathscr{M}_v$ does not affect $\mathscr{M}_c$. Moreover, the reconstruction of the fracture surface for visualization adds very little computational overhead to the overall simulation cost. In the previous Section 4.2, we explain how the system dynamics is computed on $\mathscr{M}_c$ in the presence of a fracture.

We now explain the details of how we split $\mathscr{M}_v$ for rendering. We use a node-associated four-split configuration of a tetrahedron, as shown in Figure 4.9c, for fracture visualization. When a tetrahedral element splits, the corresponding edges and faces of the element split too. We explore two different split strategies, as explained at the end of this section.

When normal stress along an edge, *PE1PE2* (Figure 4.9a), crosses the critical threshold,

it splits into two segments *PE1SE1* and *PE2SE2*. *PE1* and *PE2* are original nodes in the mesh. Let us call them parent nodes. The other two newly generated nodes, *SE1* and *SE2*, are child nodes to *PE1* and *PE2*, respectively. The child nodes do not contribute to the system dynamics but rather follow the movements of their parent nodes. Similarly, when a face gets fractured into three segments (Figure 4.9b), three child nodes, *SF1*, *SF2* and *SF3* are assigned to their corresponding parent nodes, *PF1*, *PF2* and *PF3*. Similarly four child nodes, *ST1*, *ST2*, *ST3* and *ST4* (Figure 4.9c) are assigned to four parent nodes *PT1*, *PT2*, *PT3* and *PT4* for a damaged tetrahedron. It is important to note that a triangular face or tetrahedron may not split into all three or four segments during the simulation. For example, with reference to Figure 4.9b, if we assume that only both the segments containing node *PF1* get damaged i.e., edge *PF2PF3* remain undamaged, then in such a case, child node *SF1* follows the movement of parent node *PF1* but child node *SF2* and child node *SF3* follow the average movement of parent nodes *PF2* and *PF3*. When similar scenarios, although more complex, arise for a tetrahedron, we apply the same principle. All possible cases of different kinds of fractures are taken into account in our simulation.

Figure 4.10: The left column displays fracture where an edge is split in middle. The right column displays fracture where an edge is split in the ratio of the forces on its nodes. The fractured areas in each image are zoomed up in the bottom column for better visualization.

In this work, we split edges, faces and tetrahedra with respect to their centroids. Furthermore, we also examine how the visualization output gets affected if an edge is split at a different position instead of in the middle. For this purpose, we split the edge such that the lengths of the sections are in inverse proportion to the ratio of its nodal forces. In Figure 4.10, in the left column, we render fracture on the bread model where all edges are split in the middle. In the right column, the same simulation is repeated with exact parameters and the edges are split in inverse

proportion to the ratio of their corresponding nodal forces. As evident from the figure, there is not a lot of difference in the simulation outputs. We observe that both simulations require equivalent computation time.

4.4.2 Collision Detection

After the fracture, disjoint vertices have no internal elastic force acting on them. But external body forces e.g., gravity or impulse force, continue to be applied on all the vertices whether disjoint or not. So we need no special treatment for collision detection. Continuous Collision Detection (CCD) [Tang et al., 2010a] [Tang et al., 2012] [Wang et al., 2021] is performed between $\mathscr{M}_v$ and the collider mesh (see like the hammer in Figure 4.13, 4.14). For fractured pieces colliding with the floor, we use Discrete Collision Detection (DCD) [Erleben et al., 2005b] routine (see Figure 4.14). When the visualization mesh, $\mathscr{M}_v$, collides with another object mesh, we first calculate the impulse forces for each of them using the principles of CCD and DCD. Now, as shown in Figure 4.9c, each fractured segment of a tetrahedron has exactly one parent node. For each fractured segment, we accumulate all the impulse collision forces corresponding to that particular segment and apply the accumulated force to the parent node associated with the segment. However, as discussed in the previous section, for a partially fractured tetrahedron, there will be disjoint segments which contain more than one parent node. In that case, the accumulated force is equally distributed among all the parent nodes present in the fractured segments. Thus even though the visualization mesh, $\mathscr{M}_v$, accurately detects the collisions, the effects of the collisions are handled by the computational mesh, $\mathscr{M}_c$. Currently, we do not handle self-collision between the fractured pieces.

4.4.3 Algorithm

The full algorithm of our method is presented in Algorithm 1. We implement our model using the open-source Vega FEM library [Sin et al., 2013]. In the implementation of this algorithm, for solving the system dynamics we use an implicit backward Euler integrator with a conjugate gradient solver [Sin et al., 2013]. All the simulations are performed on an Intel Core i7-9750H

Algorithm 1: Remeshing-Free Graph-based Fracture

Initialize FEM simulation;

Let n_v be total no of vertices of $\mathscr{M}_c$;

Let $[x]_{n_v \times 1}$ be initial position vector;

while *True* **do**

  for *Each element in* $\mathscr{M}_c$ **do**

    Calculate stress along the edges σ_{ij}^e *[as per Eq. 4.6]*;

    if $\sigma_{ij}^e > \sigma_{thres}$ **then**

      Fracture the edge *[as per Eq. 4.11]*;

      Update strain energy density $\mathbf{\Psi}^e$ *[as explained in Sec. 4.2.4 & Sec. 4.2.5]*;

      Update internal force $\mathbf{f}_e^{int}$ *[as explained in Eq. 2.12]*;

      Update stiffness matrix $\mathbf{k}_e$ *[as explained in Eq. 2.15]*;

      Remesh the $\mathscr{M}_v$ for visualization *[as explained in Sec. 4.4.1]*;

    end

    Resolve all collisions with $\mathscr{M}_v$ *[as explained in Sec. 4.4.2]* ;

    Calculate impulse force due to collision;

    Add all external forces to the vertices of $\mathscr{M}_c$;

  end

  Build full system $[M]_{n_v \times n_v} [v]_{n_v \times 1} = [f]_{n_v \times 1}$;

  Solve for velocity vector $[v]_{n_v \times 1}$;

  for *Each vertex in* $\mathscr{M}_c$ *and* $\mathscr{M}_v$ **do**

    Update position vector by $[x]_{n_v \times 1} += \Delta t \cdot [v]_{n_v \times 1}$;

  end

end

CPU with 12 threads at 2.60 GHz. For visualization, the simulation results are raytraced in Houdini. Apart from the models obtained from the Stanford 3D scanning repository, the other 3D models used in this project are obtained from *free3d.com*, *sketchfab.com*, *turbosquid.com*, *cgtrader.com* and *lifesciencedb.jp*. Open-source software TetWild [Hu et al., 2018b] and TetGen [Si, 2015] are used to generate the volumetric simulation meshes.

4.5 Results

We present multiple visualizations of fracture simulation for a variety of materials to demonstrate the robustness of our method and the diversity it can handle. We present dynamic fracture simulation results which mimic real-world events like tearing a loaf of bread, collision of hard objects with solid jade models and objects made of porcelain shells. We also show the effect of applying tensile or impact forces to rubber-like and jello-like materials. Following that we discuss the effects of mesh resolution on both visualization and computational mesh during a fracture simulation.

Next, we experimentally validate our model by recreating benchmark fracture experiments in simulation that are commonly performed in a structural mechanics laboratory, with our framework and compare the results with expected results from similar experiments performed in the real world. These experimental simulations are complete dynamics simulations that demonstrate the accuracy of our method. Finally, we compare our method with existing related works.

Mesh resolution and timing results for the simulations are reported in Table 4.1. The timestep for all our simulations is $5.0e-03$.

4.5.1 Dynamic Simulations

In a complete dynamic simulation, we take into account the effects of external forces and solve full system dynamics as presented in Equation 2.7 with mass & damping matrix regularizers.

We use Neo-Hookean energy with monotonic degradation for ductile fracture simulation

Simulation	*# Tetrahedra*	*sec/frame*	ρ *(kg/m^3)*	**Y** *(Pa)*	ν	R_d	σ_{th} *(Pa)*	σ_p *(Pa)*
Loaf	620.6k	4.38	1.0e+03	1.0e+07	0.35	1.0	4.0e+06	1.0e+06
PVC doormat	234.4k	1.62	1.0e+03	1.0e+08	0.45	0.0 & 5.0	2.5e+06	1.5e+05
Jello armadillo	159.1k	1.09	1.0e+03	1.0e+06	0.3	2.0	1.5e+05	5e+03
Meat Filled Loaf (Meat)	620.5k	4.57	1.0e+03	1.0e+09	0.4	0.0	8.0e+06	5.0e+06
Meat Filled Loaf (Bread)	620.5k	4.57	1.0e+03	1.0e+07	0.35	1.0	4.0e+06	1.0e+06
Steak	186.1k	1.33	1.0e+03	1.0e+09	0.4	0.0	8.0e+06	5.0e+06
Solid jade armadillo	4.7k	0.04	1.0e+03	1.0e+10	0.4	0.0	5.0e+07	-
Porcelain shell armadillo	41.1k	0.33	1.0e+03	1.0e+10	0.4	0.0	5.0e+07	-
Solid glass bunny	23.7k	0.18	1.0e+03	1.0e+10	0.45	0.0	5.0e+07	-
Porcelain shell bunny	38.8k	0.29	1.0e+03	1.0e+10	0.45	0.0	5.0e+07	-
Granite cylinder	45.8k	0.39	2.75e+03	4.5e+10	0.33	0.0	9.56e+06	-
A516 Gr.70 steel bar								
Brittle at $-40°C$	27.8k	0.24	7.7e+03	2.2e+11	0.28	0.0	6.55e+08	-
NVE 36 steel bar								
Ductile at $23°C$	27.8k	0.24	7.7e+03	1.9e+11	0.28	0.0	5.4e+08	8.3e+07

Table 4.1: Simulation parameter table. Parameters ρ, **Y**, ν and R_d (Equation 4.10) denote density, Young modulus, Poisson's ratio and support of the kernel for crack diffusion. Parameters σ_{th} and σ_p denote yield and plasticity threshold.

as it is better at preserving large deformation (see Section 4.2.5.2 for explanation). StVK energy with linearization is used for brittle fracture simulation as brittle materials show very little deformation.

4.5.1.1 Ductile Fracture

We illustrate the highly detailed and intricate fracture effects produced by tearing a loaf of bread in Figure 4.1 and Figure 4.11. It is evident from these figures that our simulation model can model complex fracture patterns and can easily scale to high-resolution meshes. In Figure 4.2 we simulate the limbs of a jello armadillo being ripped off. Here we also visualize the corresponding strain profiles. In the frame shown on the leftmost side, the armadillo appears just prior to the fracture. Thus, it has the highest strain on elements that are going to be damaged (coloured red). Once damage sets in post-fracture, as shown on the right, the strain decreases

Figure 4.11: Our method produces the intricate fracture patterns that result from the tearing of a loaf of bread. We show rendered frames from the simulation (to be seen left to right, top to bottom). The loaf model has around 620k tetrahedra.

again. The disconnected nodes in the damaged mesh have very low strain values and thus appear in blue-green. In Figure 4.4, we show the tearing of a doormat with different sizes of kernel support, R_d, thus producing localized (middle) and diffused (bottom) fracture.

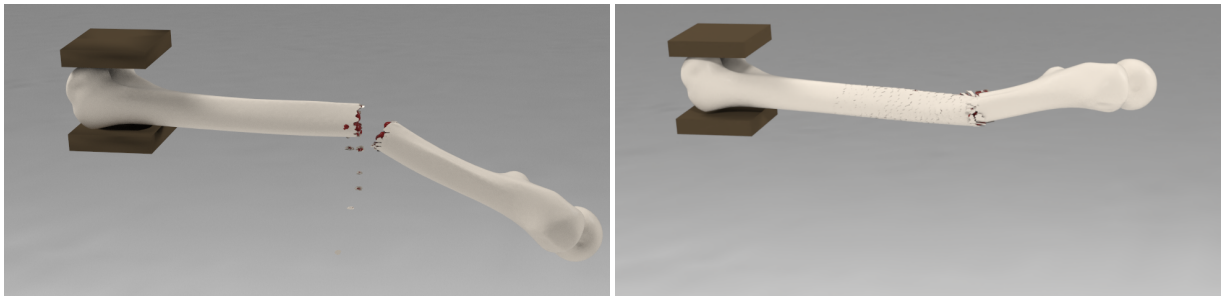


Figure 4.12: Fracture of bone: splitting (left) and buckling (right).

In Figure 4.12, we show frames from two simulations of fracture in a femur bone. Bone displays both brittle and quasi-ductile behaviour depending on the direction of loading being applied [Peterlik et al., 2006]. The left image shows a simulation where we hold the bone at one end and apply a side impact force, thus causing a brittle, transverse split fracture [o AAOS, 2020], caused by shear stresses that develop in the bone. On the right, we compress the bone, causing it to bend and behave in a quasi-ductile manner, producing a butterfly, comminuted or open fracture [o AAOS, 2020]. This behaviour of the bone emerges automatically from our simulation model.

4.5.1.2 Brittle Fracture

For brittle fracture simulation, over our regular graph-based FEM, we use a stress-based fracture surface initiation and propagation algorithm similar to [Koschier et al., 2015] [Glondu et al.,

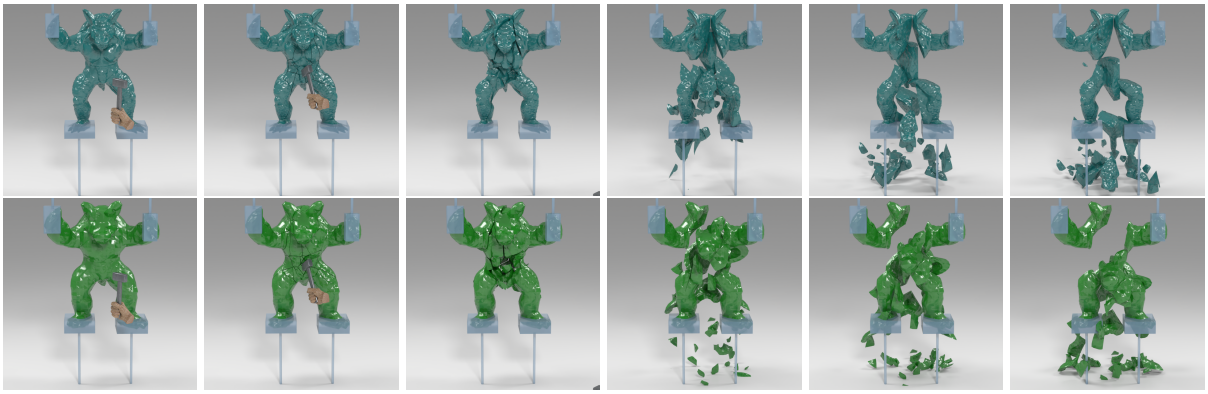


Figure 4.13: Fracture of an armadillo model made of solid blue jade (top row) and porcelain shell (bottom row).

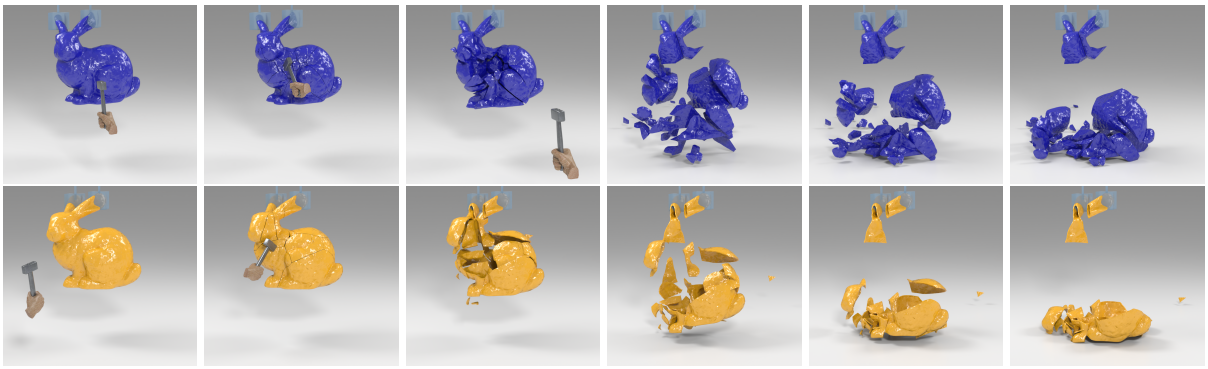


Figure 4.14: Fracture of a bunny model made of solid glass (top row) and a porcelain shell (bottom row).

2013]. Similar to those works, first we identify a few regions containing the highest amount of stress and then initiate a single fracture surface from each of these regions. Each of these fracture surfaces propagates to the boundary of the object. We first calculate the intersections of a fracture surface with the edges of our FEM mesh and then mark these edges as damaged with the embedded intersection points. Finally, the graph-based FEM is used to simulate the fractured object.

In the top row of Figure 4.13, a hammer hits an armadillo made of solid blue jade to generate large chunks of debris from the fracture. In the bottom row of the same figure, a hammer smashes a porcelain shell model of an armadillo.

Using our fracture simulation method, in the top row of Figure 4.14 we render the fracture of a solid glass model of a bunny when it is hit by a hammer. The fracture of a shell model of a bunny made of porcelain is simulated in the bottom row of the same figure.

4.5.2 Anisotropic Fracture

For anisotropic material, we use anisotropic energy density from Equation 4.36 along with invertible Neo-Hookean energy density [Xu et al., 2015].

Figure 4.15: A piece of steak breaks on pulling, according to anisotropic muscle fibers in the meat, to reveal fine-scale geometry around the fracture.

In Figure 4.15 we tear a piece of steak to reveal the intricate fracture pattern originating along the anisotropic fibre direction. The anisotropic stretchy muscle fibres inside the meat can be seen in the figure as fine thread-like structures.

Figure 4.16: A meat-filled loaf is torn apart. The outer bread is isotropic while the meat inside contains anisotropic fibers.

In Figure 4.16, a piece of a meat-filled loaf of bread is torn apart. Here we use a single mesh model with different material properties at different parts of the mesh. The coating is made of normal bread modelled as an isotropic material, while the meat inside contains anisotropic fibres in x-direction. It can be seen that the two materials fracture differently. The meat resists more before fracture due to its fibre strength.

4.5.3 Effect of Mesh Resolution

We show the effect of mesh resolution on the fracture simulation in Figure 4.17. In the figure, the computational mesh resolution is increased from top to bottom with topmost, middle and bottom-most models consisting of 1.5k, 38k and 620k tetrahedra respectively. As evident from the figure, with the increase of mesh resolution, finer and more intricate details of the fracture can be captured. The simulation times required are 0.013 sec/frame, 0.31 sec/frame and 4.38 sec/frame respectively from lowest to highest resolution mesh.

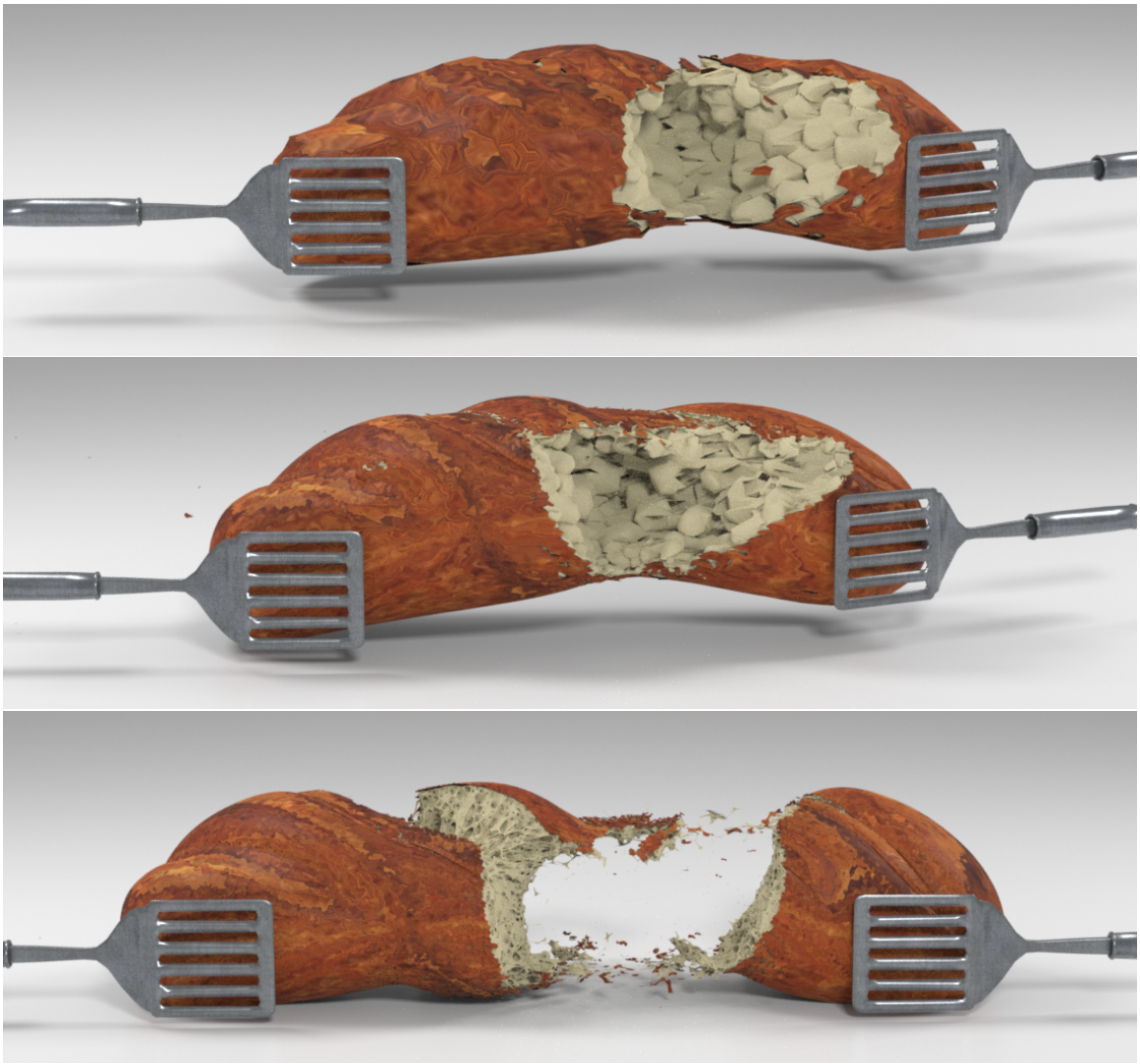


Figure 4.17: Computational mesh resolution is increased from top to bottom.

4.5.4 Debris control

We can control the amount of debris produced due to fracture by tuning different parameters. In Figure 4.18, a bread fracture are shown with varying amount of debris. The parameters that are required to be tuned in this example to produce less debris are plasticity and fracture threshold. The image on the left has plasticity and fracture threshold $2.5e+06$ Pa and $3.0e+06$ Pa respectively. On the contrary, the plasticity and fracture threshold for the image on right is $1.0e+06$ Pa and $4.0e+06$ Pa respectively.

Figure 4.18: Fracture simulation with varying amount of debris. The left column shows more debris while the right column shows less amount of debris.

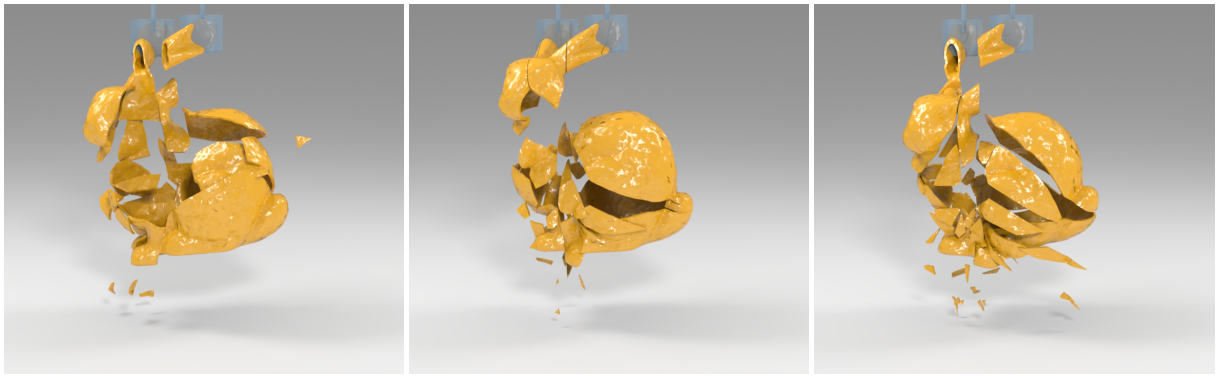


Figure 4.19: Fracture simulation with increasing (left to right) amount of debris.

For brittle fracture, as discussed earlier, first we pick out a few regions containing the highest amount of stress and then initiate a single fracture surface from each of these regions. Now, if more number of high-stress regions are picked out, the amount of debris also increases. In Figure 4.19, the amount of debris for the brittle fracture of a porcelain bunny shell is depicted. In the figure, we choose 50 high-stress regions for the leftmost image, 150 high-stress regions for the middle image and 300 high-stress regions for the rightmost image. It can be seen in the figure that the amount of debris increases from left to right with the increasing number of high-stress regions. The user can easily change this parameter to control debris.

4.5.5 Experimental Validation

To evaluate and validate how faithfully real-world fracture can be simulated using our method, we present a qualitative and quantitative comparison with three benchmark laboratory fracture experiments. These are the Charpy impact test, the Izod impact test and the Brazilian test.

4.5.5.1 Charpy Impact Test

Figure 4.20: **Charpy Impact Test**: The left image shows the configuration of the Charpy test. In the middle image, a specimen with a higher plasticity threshold splits into two, while on the right a specimen with a low plasticity threshold only bends and gets partially damaged.

In this test, we use a steel specimen of dimension $10mm \times 10mm \times 55mm$ with a 45° 'V' shaped groove in the middle of it (see the left image of Figure 4.20). A pendulum of known mass and length then impacts the back of the groove with both ends of the specimen held against fixed beams. Now depending on the tensile strength and plasticity threshold of the material, the specimen breaks fully into two pieces or is partially damaged and bends in a three-point configuration. We simulate this experiment using our method, on specimens made of steel with different tensile strengths (the name of the steel is mentioned in Table 4.1). Instead of the entire swinging pendulum, as present in the real Charpy test setup, we simulate an impact with a fast-moving block (as can be seen in the image) that hits the specimen at the same location where the specimen is supposed to be hit in the real test. As shown in Figure 4.20 the steel with a higher plasticity threshold (middle image) breaks into two pieces while the other one (right image) splits partially and bends. Simulation material parameters used in this experiment are obtained from [Tronskar et al., 2002].

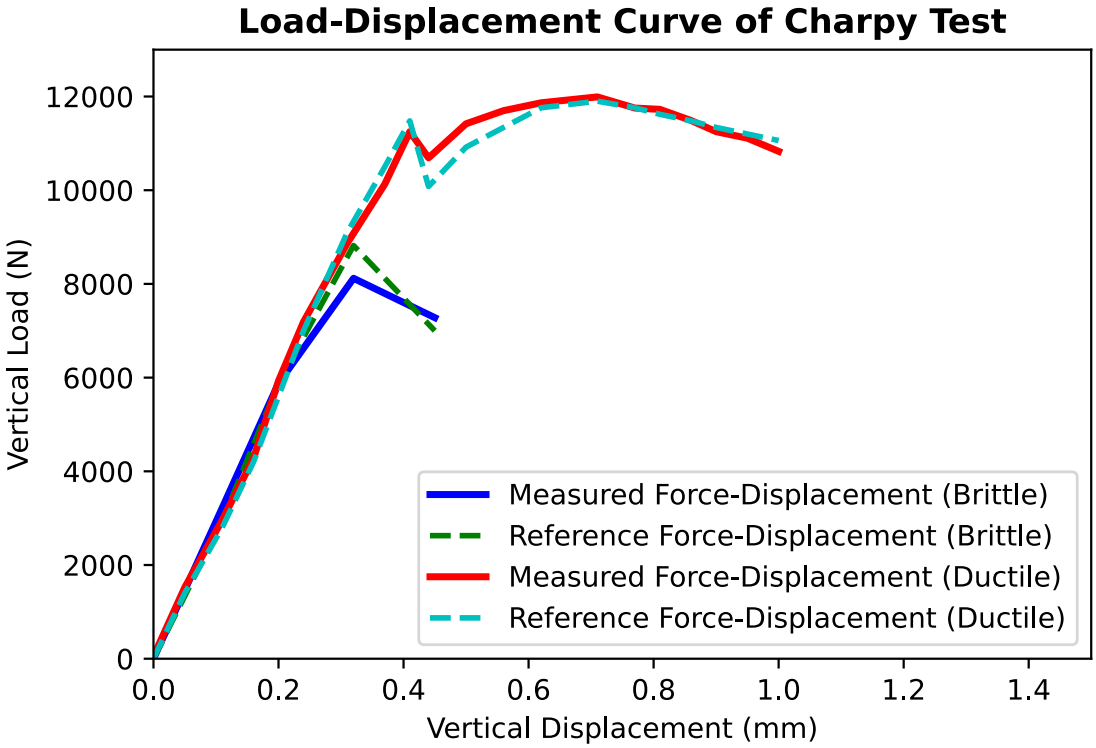


Figure 4.21: The plot shows the load-displacement curves of our simulated Charpy impact test. The simulated curves are compared with ground truth curves from actual laboratory experiments.

4.5.5.2 Izod Impact Test

The Izod impact test specimen is shown in the left image of Figure 4.22. Here the specimen is held in a cantilever beam configuration with one end fixed and the pendulum hits it on the other end. The results of the Izod test performed on the same materials as before are shown in the middle and right images of the figure.

In Figure 4.21 and Figure 4.23 we plot the load-displacement curves [Tronskar et al., 2002] for the Charpy and Izod tests respectively as obtained from our simulated experiments. The solid blue curve denotes simulated brittle fracture, which splits sharply beyond a load threshold. The solid red curve which denotes simulated ductile fracture indicates a plastic flow in the material. The brittle and ductile fracture curves for both tests closely resemble the ground truth reference force-displacement curves shown in the same figure with dotted lines. The ground truth reference curves are reproduced from the same experiments reported in [Tronskar et al., 2002]. The reference data for ductile fracture is obtained from Figure 10

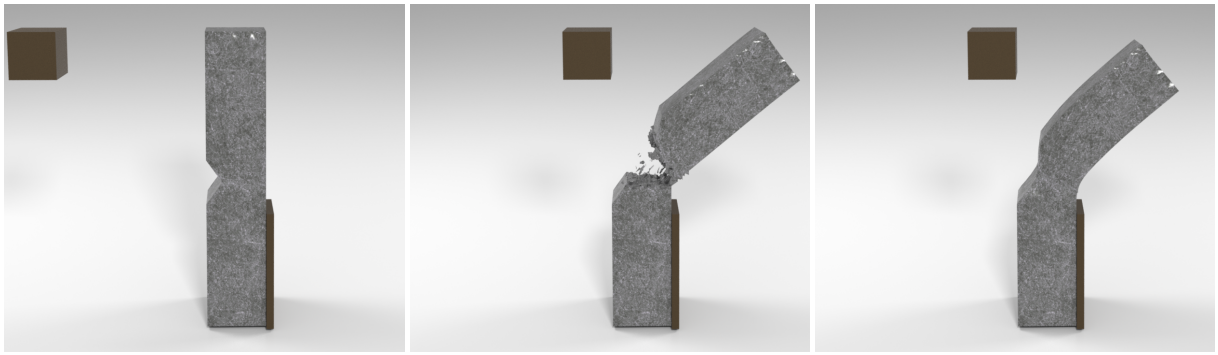


Figure 4.22: **Izod Impact Test**: The left image shows the configuration of the Izod test. In the middle image, a specimen with a higher plasticity threshold splits into two, while on the right a specimen with a low plasticity threshold only bends and gets partially damaged.

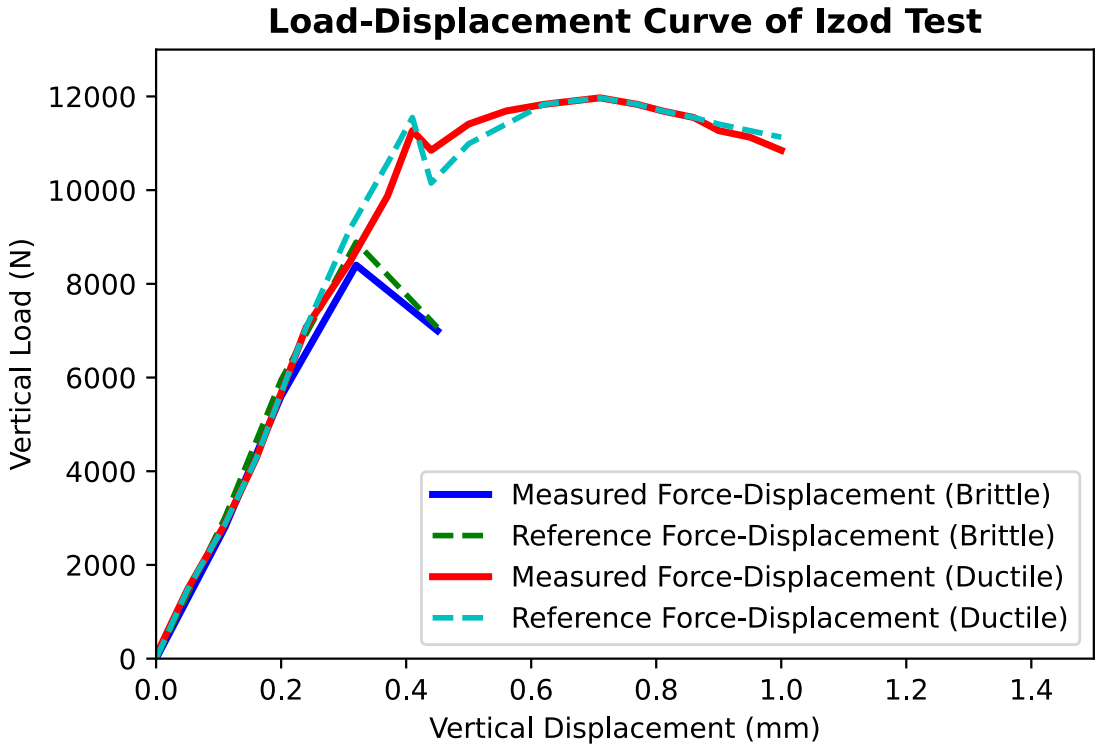


Figure 4.23: The plot shows the load-displacement curves of our simulated Izod impact test. The simulated curves are compared with ground truth curves from actual laboratory experiments.

in [Tronskar et al., 2002]. In this figure, both the force/load and load-line displacement are plotted against the time axis. However, in our plots, we depicted the force/load curve against the load-line displacement. Because in standard practice, the ductility and brittleness of material are denoted using the load-displacement curve as illustrated in Figure 4 in [Tronskar et al., 2002].

Similarly, the reference data for brittle fracture is obtained from Figure 12 in [Tronskar et al., 2002]. Brittle material fractures completely after the elastic region whereas ductile material partially fractures and then bends due to plasticity. This proves that brittle and ductile material simulation performed by our method closely matches the behaviour of real-world materials, both qualitatively and quantitatively.

4.5.5.3 Brazilian Test

The Brazilian test is a laboratory test for the indirect measurement of tensile strength. In this test, a vertical load is applied at the highest point of a cylinder, the axis of which is placed horizontally and is supported on a horizontal plane. Here we use a cylinder made of granite. In Figure 4.24 the initial (left) and fractured (middle & right) configurations of the cylinder are shown. We compare the results of the experiment performed in simulation using our method with a similar real experiment from literature [Ghouli et al., 2021] [ExpeditionWorkshed, 2013] by plotting their corresponding load-displacement curves in Figure 4.25. The close match between the two curves validates the accuracy of our simulation model. Simulation material parameters used in this experiment are obtained from [Ghouli et al., 2021].

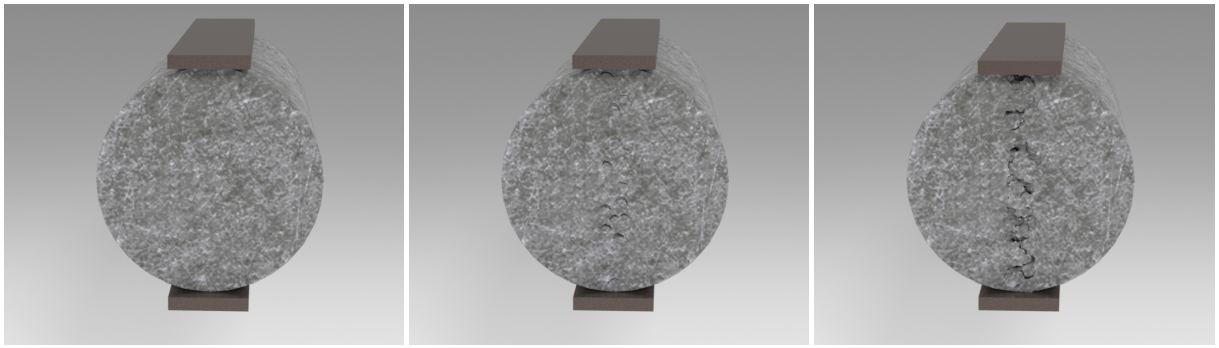


Figure 4.24: **Brazilian test**: Initial (left) and fractured (middle & right) configuration of cement cylinder.

4.5.6 Comparison with Existing Techniques

To the best of our knowledge, this is the first work in computer graphics that presents a FEM-based fracture method that is graph-based, remeshing-free, highly stable and can incorporate

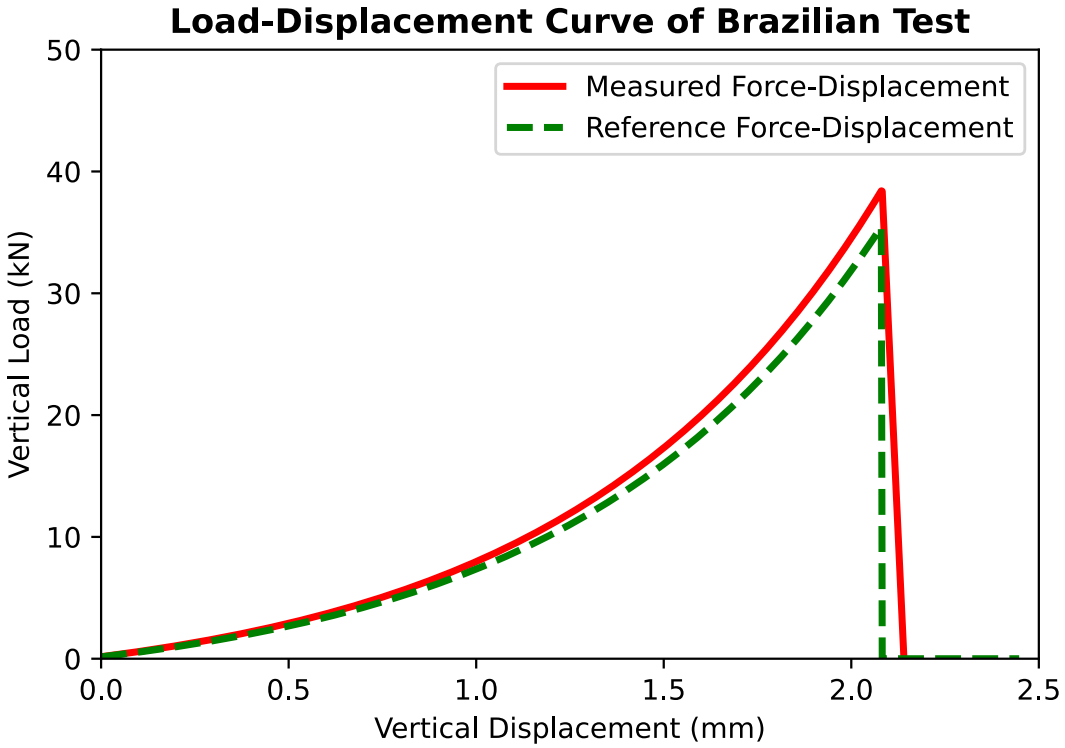


Figure 4.25: **Brazilian test**: Load displacement curve for original and simulated experiments.

a large number of cracks with little extra computational overhead on FEM. We compare our method with other existing fracture simulation algorithms in the literature.

4.5.6.1 XFEM vs Our method

Unlike XFEM [Koschier et al., 2017] [Chitalu et al., 2020] or VNA [Sifakis et al., 2007] [Wang et al., 2015], the size of the system matrix (see Equation 2.7) remains constant throughout the simulation, thus reducing the computational cost over standard FEM substantially. We implement the method presented by Koschier et al. [2017] and then compare it with our method. The visual comparison between XFEM and graph-based FEM is presented in Figure 4.26. As shown in the figure, we use a cylindrical object consisting of 600 DOFs (nodes) and 1566 tetrahedra with a pre-defined vertical split in the middle. To simulate the split, we need to enrich appropriate nodes for XFEM and relabel the appropriate edges as damaged for our graph-based FEM simulation. After enrichment, the number of DOFs (original nodes and enriched nodes together) for XFEM simulation increases to 852 while the number of DOFs remains the same in graph-based FEM. Next, we let the disjoint piece fall under gravity and calculate the

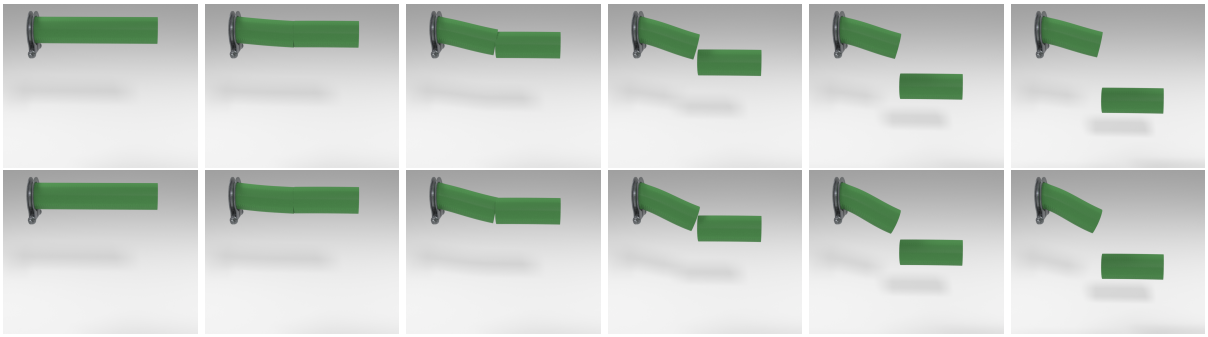


Figure 4.26: **XFEM vs Our Method**: A cylinder is split with a single cut. This is simulated using XFEM (top row) and Graph-based FEM (bottom row). Graph-based FEM produces comparable results with significantly lower run-time.

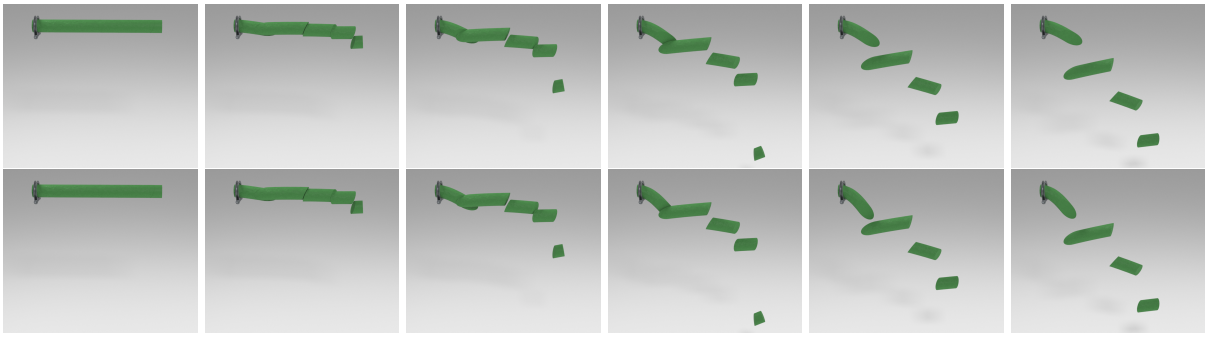


Figure 4.27: **XFEM vs Our Method**: A cylinder is split into many parts with multiple random cuts. This is simulated using XFEM (top row) and Graph-based FEM (bottom row).

average frame rate for both simulations. The simulation of XFEM takes around 0.271 sec/frame while our graph-based FEM requires 0.009 sec/frame. Similarly, in Figure 4.27, we use another cylindrical object consisting of 7.7k DOFs and 33.5k tetrahedra with multiple random splits. After enrichment, the number of DOFs for XFEM simulation increases to 9.2k while the number of DOFs remains the same in graph-based FEM. The simulation of XFEM takes around 5.87 sec/frame while our graph-based FEM requires 0.253 sec/frame.

We also present a numerical comparison in terms of simulation run-time between our method and XFEM-based fracture simulation by Chitalu et al. [2020]. Using their open source code-base [Chitalu, 2020], the XFEM-based method requires 2743 sec to render 29 number of cuts for a solid bunny model consisting of 23.7k tetrahedra. On the other hand, our graph-based FEM model requires only 0.18 sec to render the same number of cuts on the same bunny model. Thus, we have a clear advantage in terms of the speed of simulation. Moreover, simulating a large amount of debris with existing FEM or XFEM-based solutions in literature requires

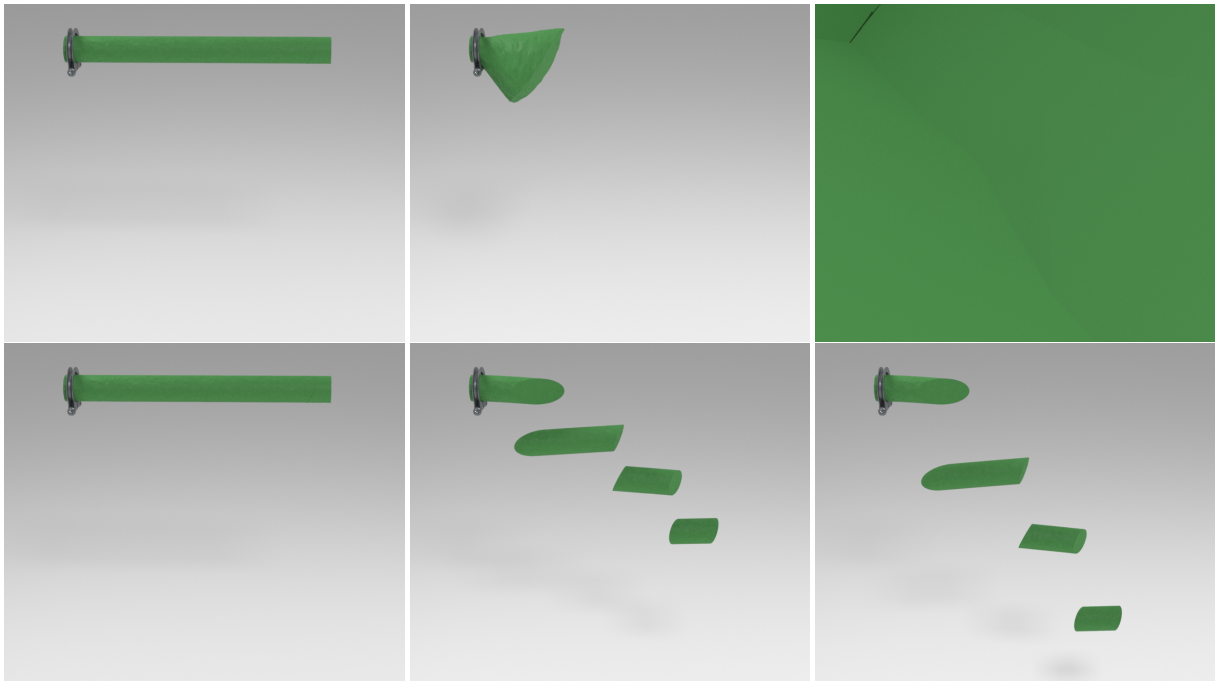


Figure 4.28: **Stability of XFEM vs Our Method**: Multiple random splits of a cylindrical object are simulated using XFEM (top row) and Graph-based FEM (bottom row). For a larger time step $\left(\Delta t = 5 \times 10^{-2}\text{sec}\right)$ XFEM simulation becomes unstable while our method remains stable.

prohibitively expensive numerical computation due to the remeshing and linear scaling of the system matrix. Our method can simulate a very high amount of fracture debris easily. Thus, our method enjoys the good characteristics of both FEM and XFEM methods, without the overheads.

Moreover, while rendering multiple random cuts using XFEM, we notice that quite often it generates skewed volume ratios of the tetrahedral elements. This in turn makes the system matrix degenerate with high-condition number. Thus for a larger time step $\left(\Delta t = 5 \times 10^{-2}\text{sec}\right)$ XFEM quickly becomes unstable (see Figure 4.28). On the contrary, as the system matrix remains unchanged throughout the simulation for graph-based FEM, our method can remain stable even for a considerably larger time step.

4.5.6.2 Remeshing-based FEM vs Our method

As no open source implementation of the work [O'Brien and Hodgins, 1999] that simulates fracture using FEM followed by appropriate remeshing, is available, we compare our method to it by reproducing the fracture based on the numerical data as reported in their published

work. Brittle fracture of an adobe wall, which contains 338 nodes and 1109 elements initially, is rendered in [O'Brien and Hodgins, 1999]. After the fracture, the mesh size grows to 6892 nodes and 8275 elements in the final configuration with multiple disjoint segments. Plugging in the same set of parameters as reported in [O'Brien and Hodgins, 1999], Vega FEM takes $0.11-0.14$ sec/frame to simulate a similar number of disjoint objects, which have an equal number of nodes and elements together. We manually put a hard constraint on the number of fragments to get an equal number of disjoint fragments while breaking the object mesh. Compared to this, our method can simulate the fracture of an object to produce a similar number of disjoint pieces in just 0.007 sec. Our object mesh consists of approximately 400 nodes and 1200 elements with the same set of material parameters as before. In our graph-based FEM, the number of nodes and tetrahedra remain unchanged even after fracture, resulting in a speed-up in computation time compared to remeshing-based methods. This shows a reduction by nearly 15 to 20 times in required simulation time using our method.

4.5.6.3 Comparison With Real Materials for Anisotropy

We qualitatively compare our fracture simulation framework with real-world materials. In Figure 4.29 (top) we show a piece of raw meat being torn apart by a lion. We render a similar simulation of tearing off a piece of steak using our framework beside it. As visible in the figure, our framework can reproduce real-world fracture phenomena, e.g., intricate fracture of the stretchy fibres in the anisotropy direction (green ellipse), a partially separated and dangling piece of flesh (blue ellipse). Similarly, in the bottom row of Figure 4.29, we present a real-world and simulated fractured wooden branch. As depicted in the figure, our framework can reproduce various intricate fracture patterns, e.g., the extruded portion of the broken branch generated due to extra anisotropic stiffness (green ellipse) and thread-like stiff wooden fibres (red ellipse).

4.6 Discussions

We present a novel graph-based remeshing-free FEM approach for ductile and brittle fracture. Our method is capable of simulating fracture for isotropic as well as anisotropic materials. We derive theoretical proofs to show that our method works for a large class of non-linear hyper-

Figure 4.29: A piece of raw meat torn apart by a lion is compared with the tearing of meat simulated using our method (top). A broken piece of a real wooden branch is compared with the breaking of a wooden branch simulated using our method (bottom.) ©Top left image: *Dreamstime* royalty-free images.

elastic strain energy densities including anisotropy by illustrating multiple dynamic experiments with different materials. We establish the high stability, speed and robustness of our method. We evaluate the appeal and realism of our fracture simulations framework through results on objects made of a wide variety of materials. The correctness and accuracy of our method are validated by comparisons with benchmark fracture experiments. We also compare with existing XFEM simulation methods, providing quantitative and visual evidence supporting the better performance of our approach.

Unlike Is-XFEM, graph-based FEM does not keep on increasing its system matrix size with the introduction of new cracks. Thus, the computational cost of graph-based FEM is independent of the number of discontinuities introduced due to fracture and nearly identical to a regular FEM. Even though graph-based FEM is capable of producing particularly realistic fractures of a variety of materials at a high speed, it does have some limitations as listed below.

- Graph-based does not take into account the effect of materials impurities in fracture simulation. Impurity is almost inherent to any material due to reasons ranging from environ-

mental effects to human interventions.

- Though graph-based FEM is faster than the state-of-the-art XFEM method, it can not run interactively for high-resolution meshes.

In our next work, we extend graph-based FEM to include the effect of impurity in fracture simulation.

Chapter 5

Simulating Fracture in Impure Materials

5.1 Introduction[1]

If we take a closer look at the wide array of material fractures occurring around us in the real world, we notice that not all materials fracture the same way. The state or condition of the material affects the fracture as well. While some materials remain fresh, others get corrupted over time due to environmental and human interventions, which in turn impose uncertainties on the fracture criteria. Purely deterministic approaches with pre-defined fracture thresholds cannot model such phenomena. We enhance the graph-based FEM method presented in Chapter 4, using novel probabilistic damage mechanics to model changes in material properties due to the presence of impurities.

[1]Portions of this chapter are adapted from *Simulating Fracture in Anisotropic Materials Containing Impurities* by Avirup Mandal, Parag Chaudhuri, and Subhasis Chaudhuri, published in ACM SIGGRAPH Conference on Motion, Interaction and Games (MIG), 2022.

We employ random graph algorithms to implement novel probabilistic damage mechanics that utilize the graph structure of the simulation object mesh in graph-based FEM to model fracture in the presence of material impurities. Even when simulation strategies are available for fracture, getting a specific fracture pattern, as desired by an artist, to appear during simulation can require hours of manual parameter tuning. We present a method for control of fracture that allows an artist to direct the fracture by designing impurity maps on the objects. These maps govern the distribution of impurities in the material of the object. In conjunction with our method for simulating fracture in anisotropic materials with impurities, impurity maps can guide the cracks formed during simulation in a manner such that the fracture pattern closely resembles the impurity map.

The specific contributions of this work presented in this chapter are:

- A novel, random graph-based, probabilistic damage mechanics to simulate fracture in the presence of material impurities.
- We prove using a Markov random process abstraction that even after the introduction of impurities, no sharp discontinuities or spurious divergences arise in the value of various node parameters of the FEM simulation mesh.
- A method for the artist-controlled design of fracture.

The rest of the chapter is organized as follows. Section 5.2 and 5.3 present our model for probabilistic damage mechanics, detailing its formulation as a random graph. Section 5.4 describes our framework for artist-controlled fracture design. Next, we present the results of fracture simulations created using our method for various kinds of materials in Section 5.5.

5.2 Probabilistic Damage Mechanics

So far, we are simulating materials that are (theoretically) 100% pure. But in the real world, it is impossible to get a material without some impurity. Moreover, impurities are distributed over the material randomly. Incorporating impurities in a mathematically consistent way poses a huge challenge. Producing the desired effect of these impurities on the fracture simulation is

even more challenging as the random distribution of impurities imposes probabilistic threshold criteria on fracture.

Impurities in materials are not considered in most fracture simulation literature. Impurities introduce randomness in the fracture threshold. Earlier, Hahn and Wojtan [2015] varied the toughness throughout the material (in a single direction) with a cosine interpolation function that allows the generation of granular fracture patterns. However, their method is deterministic and does not capture the randomness of fracture. We use a random graph-based formulation to represent probabilistic damage mechanics in a material with impurities. During the last two decades, random graph models have been used extensively for the studies of evolving communication networks. Starting from classic Erdős–Rényi random graph [Gilbert, 1959] which randomly joins edges between two nodes, various random graph algorithms [Barabási and Albert, 1999] [Davis et al., 2021] are proposed as models in a variety of problems like the dynamic topology of the world wide web to the transmission flow of the COVID-19 pandemic. Works by Pittel [2010] and Janson and Warnke [2021] gave a rigorous mathematical analysis of the properties of the evolution of dynamic multi-graph networks and preferential attachment with them.

In graph-based FEM, cracks due to fracture are generated depending on the stress value along the direction of an edge contained in an underlying graph structure. We find this graph structure is well suited to use a random graph [Pittel, 2010] [Janson and Warnke, 2021] strategy to incorporate probabilistic threshold criteria.

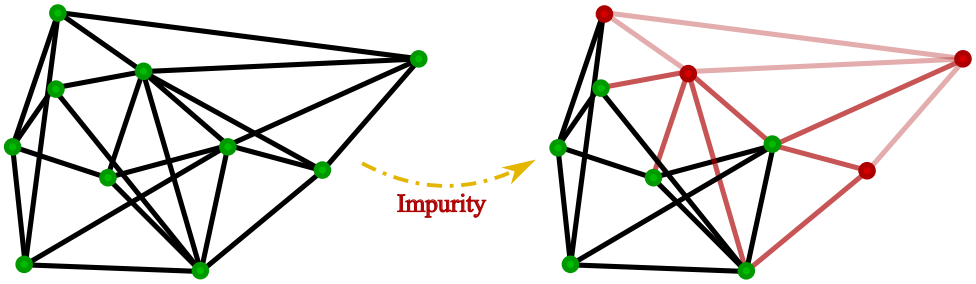


Figure 5.1: Impurities are added to a few nodes (shown in red) of the graph. Edges in light red, between red nodes with impurity, have the lowest threshold for fracture. Edges in darker red have only one node with impurity and thus have a higher threshold for fracture. Black edges between nodes with no impurity have the highest threshold for fracture.

Let us define a random graph growth process $\{G(n_v, \mu)\}$ on a vertex set $[n_v]$ with μ suc-

cessive steps. Let $G(n_v, 0)$ be a graph with no edges. Recursively, given a graph $G(n_v, \mu)$ of vertex degree $\mathbf{d} = \{d_1, d_2, \ldots d_n\}$, a new graph $G(n_v, \mu + 1)$ is obtained by inserting a set of new edges between any two still disjoint random nodes i and j with $i \neq j$. The probability that an edge e will join two currently disjoint vertices, $i \neq j$, is proportional to $(d_i + \alpha)(d_j + \alpha)$. Here d_i, d_j are the degrees of i, j in $G(n_v, \mu)$ and $\alpha > 0$ is a user-defined constant. The degree of any node refers to the number of other nodes it is connected to. The sequence $\{G(n_v, \mu)\}$ is a Markov process.

A multigraph process is defined as a sequence $\{MG(n_v, \mu)\}$ of multigraphs where self-loops and multiple edges on a single vertex are allowed. Given a current multigraph, $MG(n_v, \mu)$, with degree $\mathbf{d}$ on the vertex set n_v, a new edge joins two vertices, i and j, with probability proportional to $2(d_i + \alpha)(d_j + \alpha)$ and forms a loop $i \longrightarrow i$ with probability proportional to $(d_i + \alpha)(d_i + \alpha + 1)$. Note that in multigraph while $i \neq j$, they may not necessarily be disjoint. This random process forms a Markov chain whose state space is the set of all multigraphs on the vertex set n_v.

In our context, we start from an intermediate multigraph, $MG(n_v, \mu)$, with degree $\mathbf{d}$ on a vertex set of n_v nodes. Let us assume that the structure of the intermediate multigraph which contains n_v nodes and n_e edges, coincides with the initial undamaged graph generated by our FEM simulation mesh model. Then graph-based FEM for fracture simulation can be conceived as removing edges from a graph, contrary to the addition of edges as explained above. More specifically, the removed edges in $MG(n_v, \mu)$ denote those fractured edges of the FEM mesh on which the the internal force and normal stress are zero. However, it still remains a Markov chain whose state space is the set of multigraphs. Given any multigraph, $MG(n_v, \mu)$ with degree $\mathbf{d}$, we define the probability to remove an edge between vertices i and j as

$$p_{ij} = \begin{cases} 1 - h\left[2\left(d_i + \frac{1}{|\alpha_i|}\right)\left(d_j + \frac{1}{|\alpha_j|}\right)\right] & \text{Material degradation} \\ 1 - h\left[2\left(d_i + |\alpha_i|\right)\left(d_j + |\alpha_j|\right)\right] & \text{Material upgradation} \end{cases} \tag{5.1}$$

where α_i and α_j are random but constant initial values associated with node i and j. Function h maps the random numbers from sample space to probability space. Note that we do not include the probability of self-loop in Equation 5.1 as it does not occur in a consistent FEM mesh. To incorporate the impurities of any material, we assign Additive White Gaussian Noise (AWGN) to the nodes of the object mesh at the start of the simulation.

$$\alpha_k \in \mathcal{N}(0, \sigma^2), \;\; \alpha_k \neq 0 \;\; \forall k \in n_v \tag{5.2}$$

Higher values of α_k (for any node k) denote a higher amount of impurities and vice versa. We consider two different kinds of effects on material after adding impurities,

- **Material degradation**: This is the case when material strength decreases after adding impurities e.g., adding mud to asphalt/tar.
- **Material upgradation**: This is the case when material strength increases after adding impurities e.g., adding copper to pure gold.

Function h can be any strictly increasing function with range $[0,1]$. One such function is

$$h(x) = \frac{e^x - 1}{e^x + 1} \quad \forall\, x \in (0, \infty) \tag{5.3}$$

The choice of function depends on the user and is not unique. In case of material degradation, as evident from Equation 5.1, if the amount of impurities increases, a higher value of α_k makes the function value goes down. This, in turn, makes the fracture probability of the corresponding edge go up. A similar line of argument can be applied in the case of material upgradation but the other way around. To summarize, in our method edges of the volumetric mesh have explicit yield strengths that vary based on the probabilistic approach. As shown in Figure 5.1, when impurities are added to the nodes (shown in red), the strength of the edge between them goes down/up. The change in the strength of the edge depends on the impurity content of the corresponding nodes. Moreover, if the degree/connectivity of a node decreases, i.e, the neighbourhood of the node gets damaged, then the probability of that node getting fractured increases. This closely resembles the real-world phenomena where it is much easier to break a partially damaged material compared to an intact one.

5.3 Markov Random Field Based Analysis of Impurity Addition

Even in the presence of impurities, a material can be assumed to be locally homogeneous. In other words, we can say that the distribution of any material property of an object is a spatial Markov Random Field (MRF), i.e., material property for any point in the object, depends on

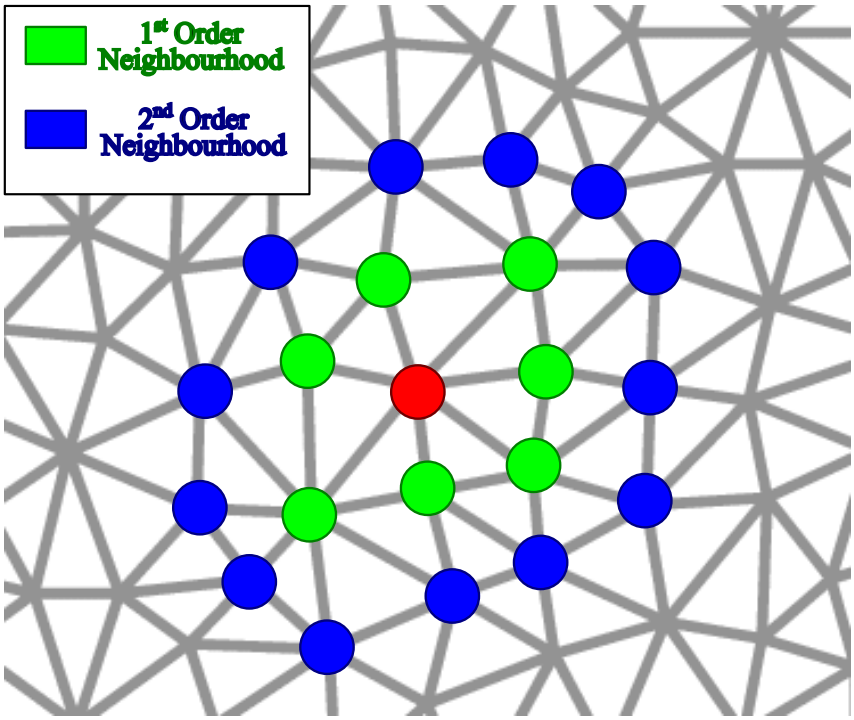


Figure 5.2: Neighbourhood structure of a node. Green and blue points refer to the 1st-order and 2nd-order neighbourhood to the original red coloured node respectively.

the properties of other points only in its local neighbourhood. It is important to examine, after adding AWGN for impurity, whether the posterior distribution preserves this locality property. Without this locality property, the object may show undesired kinks or spurious divergence due to uneven acceleration and velocity caused by the addition of impurities.

Let us first consider the neighbourhood structure of a node in an object. As shown in Figure 5.2, for any given node i, its 1st-order neighbourhood consists of the nodes which are directly connected to it. The 2nd-order neighbourhood denotes the nodes that are connected to the 1st-order nodes. Similarly, we can define higher-order neighbourhoods. We assume that any material property associated with a node will only depend on its 1st-order neighbourhood.

Now, let us assume that $X = \{x_s\}_{s \in n_v}$ denotes a *Markov process* [Gallager, 2013] where x_s represents any node variable e.g., position, velocity, acceleration etc., normalized to $[-1, +1]$. A neighbourhood relation can be defined as $\eta = \{\eta_s\}_{s \in n_v}$, where $\eta_s \subseteq n_v$ is the set of neighbours of s. Now for a MRF, the following property has to hold.

$$P(X_s = x_s | X_r = x_r, r \in n_v, r \neq s) = P(X_s = x_s | X_r = x_r, r \in \eta_s) \tag{5.4}$$

where η_s denotes n^{th}-order neighbourhood of node s. Equation 5.4 indicates that the value of

any node variable depends only on the nodes which reside inside its local neighbourhood, η_s.

Using Hammersley-Clifford theorem (see [Kindermann and Snell, 1980] [Besag, 1974] for a detailed proof), we can say that a distribution, P defines a MRF on n_v with neighbourhood relation η iff it is a Gibbs distribution with respect to the same graph, (n_v, η). Mathematically it can be represented as

$$\begin{aligned} P(X_s = x_s | X_r = x_r, r \in n_v, r \neq s) &= \frac{\exp\{-\sum_{c \in A} V_c(x)\}}{\sum_{s \in n_v} \exp\{-\sum_{c \in A} V_c(x)\}} \\ &= P(X_s = x_s | X_r = x_r, r \in \eta_s) \end{aligned} \tag{5.5}$$

and $U(x)$ is the energy function

$$U(x) = \sum_{c \in A} V_c(x) \tag{5.6}$$

where the set A consists of cliques (collection of nodes such that every two nodes are neighbours) in (n_v, η) that contains x_s. $V_c(x)$ is called the clique potential that only depends on x_s.

In the Ising model [Ising, 1925], which is widely used in this context, the energy function $U(x)$ is given by

$$U(x) = -\beta \sum_{<s,t>} x_s x_t, \quad \beta > 0 \tag{5.7}$$

where $s, t \in n_v$ denotes all the neighbouring pairs of nodes. Using the Ising model, we can define a prior distribution of X, i.e., any node variable without impurity as

$$f(x) = \left(\frac{1}{\sum_{s \in n_v} \exp\{U(x)\}} \right) \exp\left\{ -\beta \sum_{<s,t>} x_s x_t \right\} \tag{5.8}$$

The *degradation process* $Y = \{y_s\}_{s \in n_v}$ is then defined as

$$y_s = x_s + \alpha_s \tag{5.9}$$

where α_s is the normalized randomness in the node variable introduced due to impurity. So the degradation model is a conditional probability distribution given as

$$f_{Y|X}(y|x) = \left(\frac{1}{2\pi\sigma^2} \right)^{\frac{|\eta_s|}{2}} \exp\left\{ -\frac{1}{2\sigma^2} \sum_{s \in S} (y_s - x_s)^2 \right\} \tag{5.10}$$

Now we can define the distribution of node variables after the impurity is added as

$$\begin{aligned} f(y) &= \int_x f(y|x) f(x) dx \\ &= \int_{x \in \eta_s} \left(\frac{1}{\sum_{s \in n_v} \exp\{U(x)\}} \right) \\ &\exp\left\{ -\beta \sum_{<s,t>} x_s x_t - \frac{1}{2\sigma^2} \sum_{s \in S} (y_s - x_s)^2 \right\} dx \end{aligned} \tag{5.11}$$

Now from Equation 5.11 we can see that node variable distribution depends only on the nodes t that reside inside the n^{th}-order neighbourhood, η_s of the node s. Thus we can say that Y is also a Markov random process. This proves that the addition of impurities does not affect the locality property of any node variable. We use our random graph model for the simulation of materials with various kinds of impurities and the results are presented in Section 5.5.

5.4 Artist Controlled Fracture Design

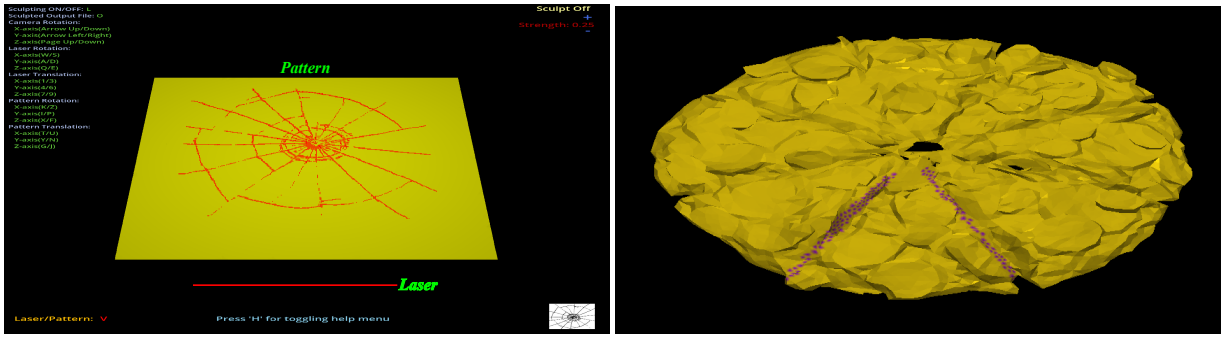


Figure 5.3: Our interactive user interface allows the artist to design impurity maps on objects (left). An artist interacts with and marks shallow cuts on a pizza mesh model using an impurity map (right).

We present a method to allow artists to design any specific fracture pattern that they want by leveraging our random graph-based formulation for impurity addition. An interactive interface, shown on the left in Figure 5.3, allows an artist to control a virtual sculpting tool to interactively create an *impurity map* on the object. The impurity map is called so because it is used to control the distribution of impurities in the region that are marked by the artist. A surface map can also be propagated inwards into the object, akin to a volumetric texture.

While using the interface, the artist can draw a free-hand impurity map using a laser sculpting tool, by placing it on the object mesh in any position. When it intersects an edge of the mesh, we weaken the fracture threshold of that edge by adding impurity to the two nodes which contain the edge. The artist can also control the potency α_k of the impurity being added.

Furthermore, our framework can embed any 2D pattern from an image to an object mesh. Given the image of a pattern, our framework generates a design with small line segments that

closely replicate the pattern. The user can also move this design in all six degrees of freedom and project this pattern on an object mesh. After projection, the edges, where the segments make up the pattern, intersect the object mesh. The pattern thus gets embedded into the mesh as an impurity map and weakens the intersected edges as discussed in the previous paragraph. The user can also control the weakening parameter value, i.e., lower the number, weaker the edge strength and vice versa. We define this as the potency of impurity. If the weakening parameter value is set to zero, it corresponds to a sharp cut. During simulation when the fracture propagates, the cracks are guided by the impurity map and thus follow the pattern created by the artist.

In the right image of Figure 5.3, we show the impurity map being added to a pizza model. Here, like a real-life pizza maker who makes shallow cuts on the pizza to make slices but does not completely separate them, we weaken the pizza along shallow cuts that follow the impurity map. The final simulation is shown in Figure 5.8. When the pizza slice is pulled apart, it closely follows these shallow cut lines but at the same time generates stretchy bits of cheese.

5.5 Results

We showcase a wide variety of fracture simulation demos to demonstrate the robustness of our method and the diversity it can handle. We use an implicit backward Euler integrator with a conjugate gradient solver [Sin et al., 2013] to solve the system dynamics. All the simulations are performed on a machine with Intel Core i7-9750H CPU at 2.60 GHz. Our multi-threaded implementation runs purely on the CPU using 12 threads and is not accelerated by a GPU. The simulation results are raytraced in Houdini for visualization. The 3D models used to generate the results are obtained from `turbosquid.com`, `cgtrader.com`, `free3d.com` and `sketchfab.com`. Volumetric simulation meshes are generated using open-source software TetWild [Hu et al., 2018b]. Simulation parameters for all our experiments are given in Table 5.1. All the materials in our simulation have simulation timestep $5.0e-03$ sec.

Simulation	*# Tetrahedra*	*sec/frame*	ρ *(kg/m^3)*	Y *(pa)*	ν	R_d	σ_{th} *(Pa)*	σ_p *(Pa)*
Dry Wooden Log	142.5k	1.09	1.0e+03	1.0e+10	0.45	0.0	5.0e+09	-
Wet Wooden log	142.5k	1.18	1.0e+03	1.0e+10	0.45	0.0	5.0e+09	-
Pizza	106.6k	0.81	1.0e+03	1.0e+07	0.4	0.0	3.0e+07	1.0+06
Pure Gold Bar	83.8k	0.67	1.0e+03	1.0e+08	0.2	0.0	7.0e+08	2.0+05
Gold-Copper Alloy Bar	83.8k	0.73	1.0e+03	1.0e+08	0.2	0.0	7.0e+08	2.0+05
Porcelain Column	40.3k	0.31	1.0e+03	1.0e+10	0.47	0.0	5.0e+09	-
Slab (isotropic)	84.3k	0.61	1.0e+03	1.0e+08	0.45	0.0	5.0e+06	3.0+05
Slab (anisotropic)	84.3k	0.65	1.0e+03	1.0e+08	0.45	0.0	5.0e+06	3.0+05
Slab (with impurities)	84.3k	0.70	1.0e+03	1.0e+08	0.45	0.0	5.0e+06	3.0+05
Tube	103.1k	0.78	1.0e+03	1.0e+08	0.4	0.0	5.0e+06	3.0+05

Table 5.1: Simulation parameter table. Parameters ρ, $\mathbf{Y}$, ν and R_d (Equation 4.10) denote density, Young modulus, Poisson's ratio and support of the kernel for crack diffusion. Parameters σ_{th} and σ_p denote yield and plasticity threshold.

5.5.1 Impurity Induced Fracture

In Figure 5.4, we compare the fracture of a dry vs a rotten and damp wooden log. The addition of the impurity weakens the material here, just like a rotten wooden log has lower material strength after getting damp by absorbing water and thus produces more disjoint segments. Additionally, all the parameters in the images of the two rows are the same except for the addition of impurity. However, as the wood gets weakened due to the water impurity (bottom row), for the same applied force it fractures early and stretches more compared to the dry wood fracture (top row). Note that there is no explicit wetting simulation here and water is (figuratively) being considered to be like a material-weakening impurity, added to produce the effect of dampness in the material.

Using our impurity-inducing model, we can also strengthen the material. We demonstrate by simulating the addition of copper to pure gold. In Figure 5.5, we stretch a thin bar of pure gold (in the top row) and gold-copper alloy (in the bottom row). The gold-cooper alloy demonstrates much more ductility and deforms more before undergoing fracture compared to the thin bar of pure gold.

Figure 5.4: A piece of dry wooden branch is broken by pulling it from one side (Top row). A damp branch that has water (figuratively) as an impurity is broken in the same way as before (Bottom row). The water weakens the integrity of the material.

Figure 5.5: A bar made of pure gold is stretched (Top row). A bar made of gold-copper alloy is stretched (Bottom row). The impure gold, i.e., gold-copper alloy is much more ductile and deforms more before fracturing.

5.5.2 Effect of Impurity Potency

In Figure 5.6, we show the effect of impurity potency on fracture. We take a hollow cylindrical tube and add an impurity map in the middle of it. The impurity map is displayed on the left-most image of Figure 5.6. The impurity map shows the location of the impurities (in magenta) added to the object. The yellow regions have no impurities. The tube is then hinged at one end and pulled from the other. In the next three images of Figure 5.6, the α_k value is decreased from left to right (leftmost - 0.8, middle - 0.5, rightmost - 0.2). Higher values of α_k mean more potent impurities, i.e., they cause more weakening of the material in this case. As evident from Figure 5.6, decreasing the value of α_k lessens the effect of impurities on the overall strength of the object and causes fractures in multiple locations in the tube as opposed to only in the region where impurities have been added.

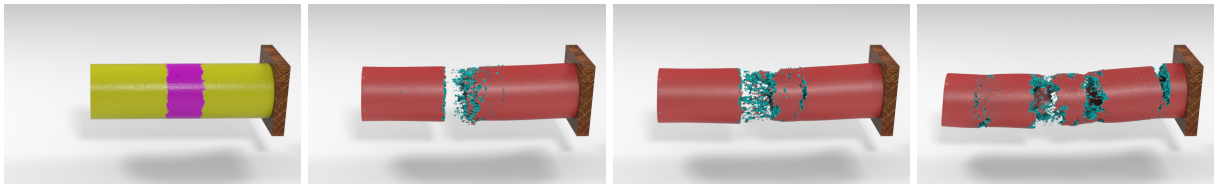


Figure 5.6: Illustration of impurity map (left). The effect of the impurity is decreased from left to right (next three).

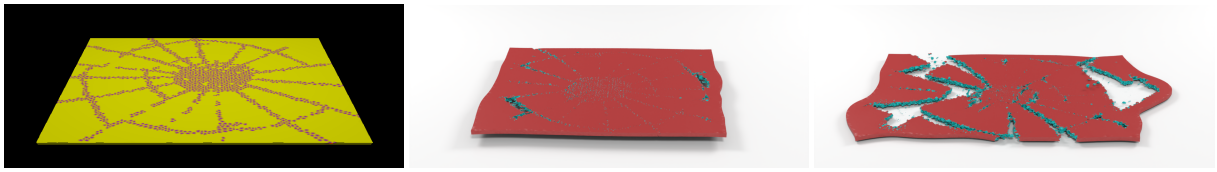


Figure 5.7: The slab model, embedded with the impurity map created from an image pattern (left), is torn apart by pulling it from both ends (middle, right).

5.5.3 Artist Controlled Fracture

Figure 5.7 shows an impurity map created from an image of a pattern (pattern is depicted in Figure 5.3 (left)) applied to a slab mesh and demonstrates a simulation of controlled ductile fracture. When the slab is pulled from both ends, it comes apart along the embedded pattern lines. Additionally, the fracture initially sets in on the outer part of the slabs where the forces are applied. Once the slab becomes disjoint due to the fracture on the outer parts, the applied force does not find a way to propagate down to the centre of the object mesh. So, the centre of the slab remains less damaged despite having more impurity density. In Figure 5.8, we render a pizza slice being pulled out from a pizza, which has been pre-cut by an artist as described earlier. This is another demonstration of artist-controlled ductile fracture.

Figure 5.8: A pizza slice is separated from the whole, along with the cuts made by an artist, and leaves stretchy bits of cheese hanging. This is an example of an artist-controlled fracture design using our method.

In Figure 5.9 a porcelain column is embedded with impurity maps corresponding to a single sharp fracture line and a more complex fracture pattern. When the column is hit by a metal ball, the simulation produces sharp shards or fragments that cleanly break along the fracture edges defined by the pattern. This illustrates the ability of our method to produce artist-controlled brittle fractures as well.

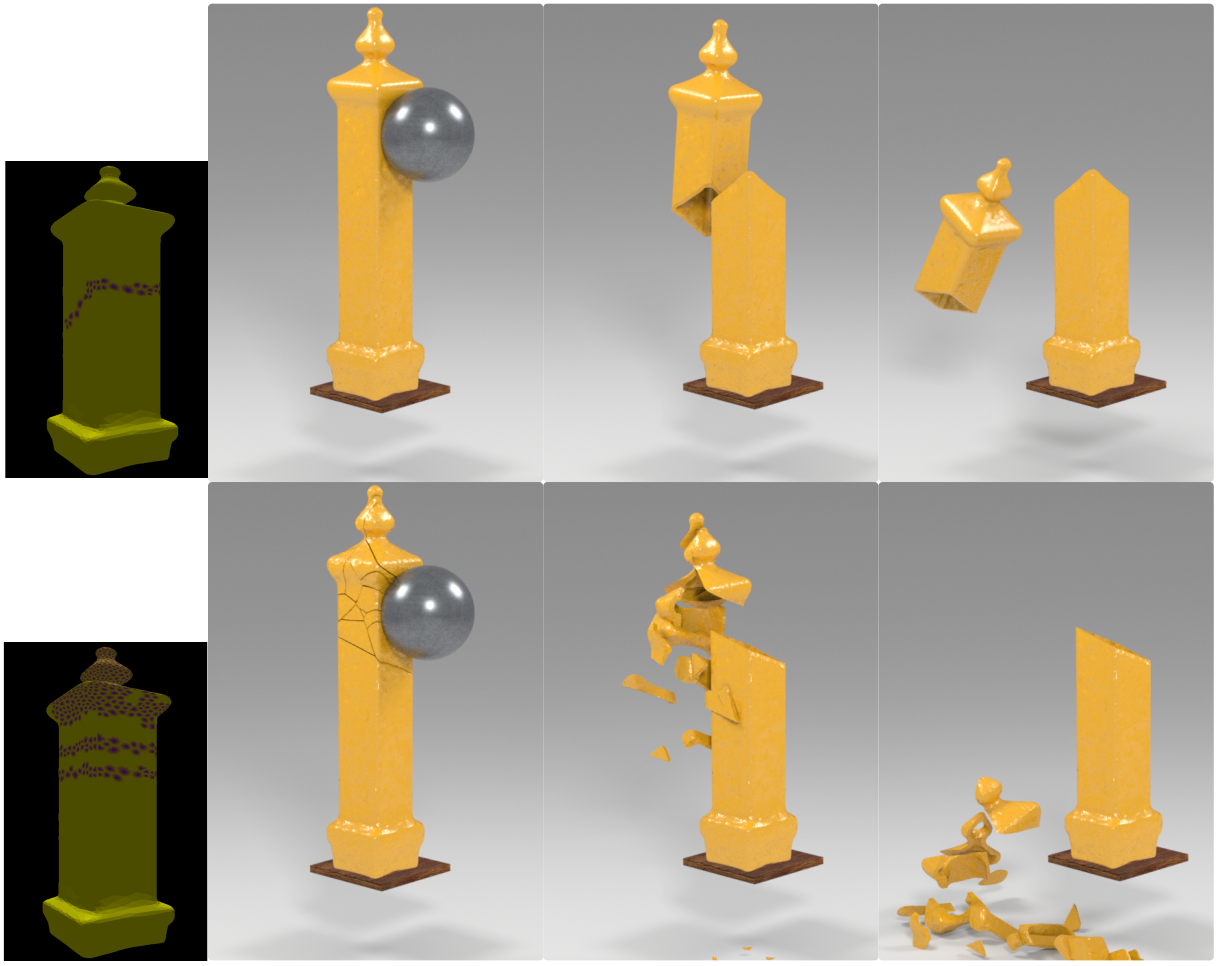


Figure 5.9: A porcelain column breaks on impact according to the embedded impurity map (leftmost inset images) with a single sharp fracture line in the top row images and a more complex fracture pattern in the bottom row images.

5.5.4 Effect of the Random Graph

The random graph-based implementation is crucial to simulating materials with impurities and for artist control of fracture. In the absence of the random graph, disintegration due to fracture

is more chaotic and does not follow the impurity distribution closely.

In Figure 5.10, we see an artist-designed impurity map in the first image for a brittle object. When simulated with the random graph formulation, the object breaks only at the intended location, i.e., the arm. However, when the same impurity map is used without the random graph formulation, the simulation causes more disintegration than intended. Without the random graph formulation, the crack lines do not have any preferred direction of propagation. So, these crack lines disperse more and generate more disjoint segments.

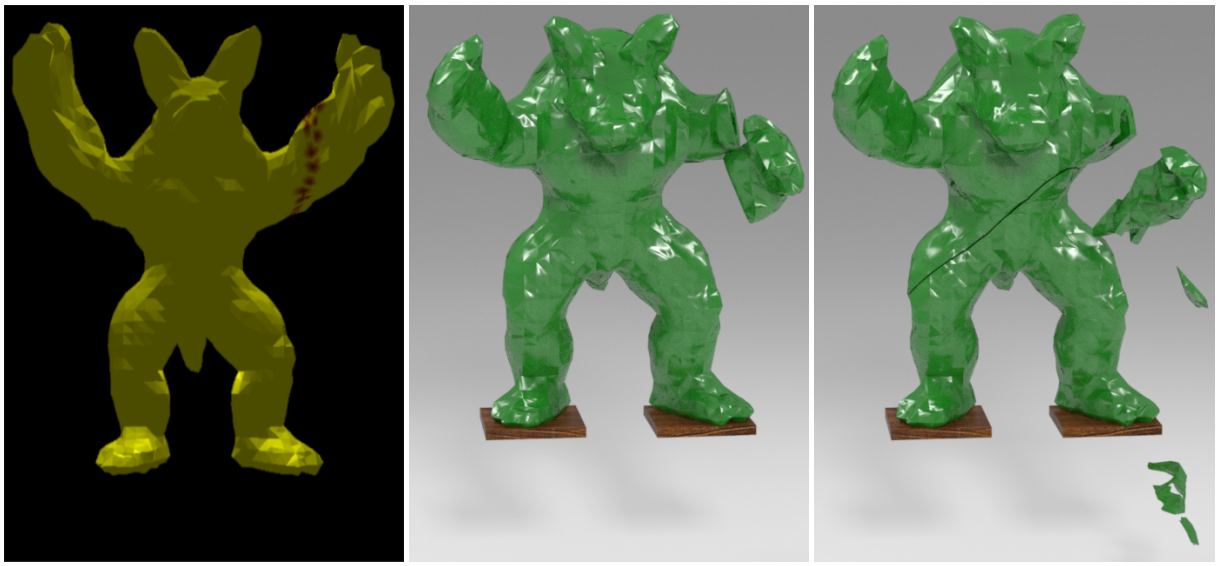


Figure 5.10: The artist-created impurity map is shown in the left image. Simulated fracture pattern with (middle) and without (right) random graph is shown next.

Similarly before, we simulate a slab that is hinged at one end and which has a band of Gaussian noise impurity in the middle. We tear the slab by pulling it from its free end. We first simulate fracture with our random graph formulation (Figure 5.11 middle) and then run the same simulation without it (Figure 5.11 right). It can be seen from the figure when the random graph formulation is not used, the model disintegration is more chaotic, does not follow the impurity distribution closely and intricate debris pattern disappears.

5.5.5 Effect of Different Noise Distributions

In Figure 5.12 we show that altering just the noise distribution model for impurity addition from Gaussian, to uniform within our random graph formulation does not produce any large variation

Figure 5.11: Effect of using the random graph can be seen here. The impurity map is shown first. A simulated fracture pattern with (middle) and without (right) random graph formulation is shown next.

in the generated fracture pattern.

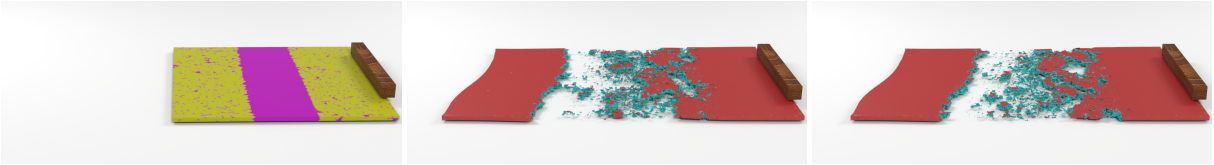


Figure 5.12: Changing the noise model used to add the impurity within the random graph does not produce much difference in the fracture pattern. Simulated fracture pattern produced when the impurity is generated using Gaussian random variable (middle) and uniform random variable (right).

5.5.6 Comparison With Existing Methods

We perform a comparison study between fracture simulation for isotropic materials performed using the algorithm presented in Section 4.2 of Chapter 4 versus fracture simulation for materials with impurities, simulated with our method. For this experiment, we use a rectangular slab that is hinged at one end and pulled from another end. The middle image of Figure 5.13 shows the anisotropic fibre directions, drawn in black stripes(see Section 4.3 of Chapter 4). The impurity map is depicted on the rightmost image of the figure. The impurity map shows the location of the impurities (in magenta) added to the object. The yellow regions have no impurities. In Figure 5.14 we show the fracture results for isotropic (1^{st} & 2^{nd} column) and anisotropic (3^{rd} & 4^{th} column) material. Moreover, on the right side of each image, we show its corresponding strain profile. The strain values are depicted via a colour map with red (largest), green (medium) and blue (smallest) colours. A fracture of the same slab with impurity-laden material is seen in the last two columns of the same image.

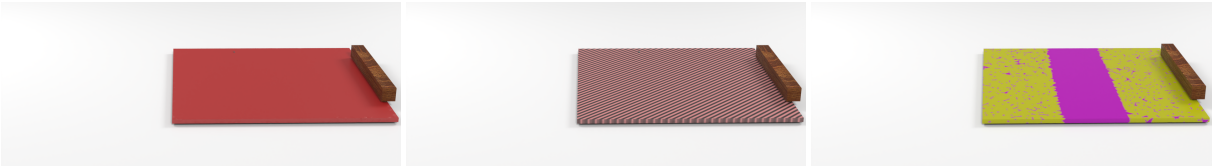


Figure 5.13: Illustration of the isotropic slab (left) in rest position, the direction of anisotropy (middle) and impurity map (right).

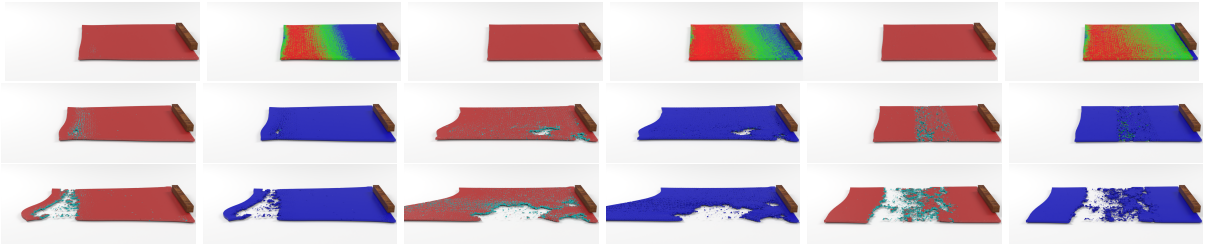


Figure 5.14: Comparison of fracture and corresponding strain profile for slab made of isotropic material (1st & 2nd column)[see Section 4.5.1 of Chapter 4], anisotropic material (3rd & 4th column)[see Section 4.5.2 of Chapter 4] and material with impurities (5th & 6th column).

The slab made of isotropic material breaks with a nearly clean horizontal fracture near the free edge being pulled. On the other hand, the slab made of anisotropic material breaks along fracture lines that are guided by the anisotropic fibre direction. Here we can clearly see patches that break along the slanted fracture lines. Due to the presence of anisotropic fibres, the material is stronger along that direction. In the last two columns of Figure 5.14, we show the fracture of the same slab, made of an impure material. The impurities are heavily concentrated towards the middle of the slab in the shape of a band. The slab contains a few more scattered pockets of impurity over the whole body. It can be seen that the slab primarily gets damaged along the impurity band and produces more disjoint fragments compared to the other two cases that have no impurities. The computational cost of the simulation with anisotropic materials or with materials having impurities is almost the same as simulating only isotropic materials.

5.6 Discussion

We introduce a novel probabilistic damage mechanics formulation to incorporate the uncertainties during fracture, caused by the presence of material impurities. We leverage this formulation

to create a framework that allows the artist-controlled design of fracture. We demonstrate the appeal and accuracy of all of the above methods for ductile and brittle fracture of various objects for materials with or without impurities.

Our random graph-based FEM formulation can simulate fracture for isotropic, anisotropic and impure materials. It can handle linear as well as non-linear strain energies. Moreover, fracture of both ductile and brittle material is possible using our framework. However, random graph-based FEM still carries the limitation of graph-based FEM i.e., being non-interactive and not real-time for high-resolution meshes.

In our work presented in the next chapter, we address this limitation.

This page was intentionally left blank.

Chapter 6

Galerkin Enhanced Graph-based FEM for Interactive Fracture Simulation

6.1 Introduction

Fracture simulation is a crucial component in various graphics applications for creating immersive virtual content. Applications such as video games, mixed reality experiences, and surgical simulators require not only visually appealing fracture simulations but also instantaneous interaction feedback. To achieve high frame rates with physical plausibility, stable large-time-step simulations for fracture simulations, the Finite Element Method (FEM)O'Brien and Hodgins [1999]Pfaff et al. [2014]Chitalu et al. [2020], Boundary Element Method (BEM)Hahn and Wojtan [2015]Hahn and Wojtan [2016], and Material Point Method (MPM)Wolper et al. [2019] Wolper et al. [2020] are widely used. These methods utilize an implicit time integration scheme; however, they are computationally expensive, and the computational cost increases with the number of cracks in the object mesh. As a result, these methods are often restricted to

low-resolution objects for real-time simulations.

To address these issues, researchers have been exploring alternative approaches to fracture simulation that offer real-time interaction feedback while maintaining visual fidelity. As discussed in Chapter 4 and Chapter 5, graph-based FEM [Mandal et al., 2023] [Mandal et al., 2022b] algorithm predicts the evolution of cracks in a remeshing-free manner and runs independently of the number of cracks. Instead of remeshing, graph-based FEM reformulates the hyper-elastic strain energy of the fractured object mesh to simulate the independent movement of the disjoint pieces. Thus, graph-based FEM is faster and more stable than the existing fracture simulation algorithms.

This advantage alone, however, is not enough for interactive fracture simulations at high frame rates on high-resolution meshes. In order to cross this hurdle, we present an interactive fracture and cutting simulation framework based on graph-based FEM, accelerated using a Galerkin multigrid method [Xian et al., 2019]. The Galerkin multigrid method allows for efficient general deformable object simulation and can maintain interactive rates even when working with very high-resolution meshes when parallelized on GPU.

The rest of the chapter is organized as follows. Section 6.2 briefly explain Galerkin multigrid method. Next, we present the results of our interactive fracture simulations framework for various kinds of materials in Section 6.3.

6.2 Galerkin Multigrid Method

A Galerkin multigrid method [Xian et al., 2019] can be conceived as an iterative method of solving a set of linear systems of an object mesh by passing it down to increasingly coarser resolution meshes at multiple levels. As shown in Figure 6.1, the set of linear systems is defined from finer to coarser resolution meshes.

Initially, the linear system, $\mathbf{A}_1\mathbf{x}_1 = \mathbf{b}_1$, described in Equation 2.7 and Equation 3.13, is solved using a stationary method such as Gauss-Seidel, Jacobi or Conjugate Gradient. However, they are stopped much earlier before convergence is reached. An early stop of these iterative methods produces a smooth error. The smooth/residual error is defined as $\mathbf{r}_1 = \mathbf{b}_1 - \mathbf{A}_1\mathbf{x}_1$.

Even though the solver is stopped early, the high-frequency error at each level gets removed quickly first, leaving only low-frequency error in the system. The low-frequency error is then passed to the next level containing a coarser resolution mesh. The error then becomes a high-frequency error at that lower level.

At the coarser level, $l+1$, the residual from level l, is restricted using the restriction matrix

$$\mathbf{b}_{l+1} = \mathbf{R}_l \mathbf{r}_l \tag{6.1}$$

The error update to the vertices of the level $l+1$ is computed, again using few iterations of a stationary method, as $\mathbf{A}_{l+1}\mathbf{e}_{l+1} = \mathbf{b}_{l+1}$

Subsequently, the residual for level $l+1$ is computed as

$$\mathbf{r}_{l+1} = \mathbf{b}_{l+1} - \mathbf{A}_{l+1}\mathbf{e}_{l+1} \tag{6.2}$$

After the solve at the coarsest level, the error values at each level are interpolated back to the finer level using prolongation or interpolation matrix $\mathbf{P}_l$.

$$\mathbf{e}_l = \mathbf{P}_l \mathbf{e}_{l+1} \tag{6.3}$$

At the finest level, the next position is updated using $\mathbf{x}_1 = \mathbf{x}_1 + \mathbf{e}_1$.

Since the solve at each level can be stopped early, and only errors of lower frequency get propagated to coarser levels, it is not necessary to re-discretize the system matrix using the Finite Element Method on the coarser mesh in the Galerkin multigrid method, unlike pure geometry-based multigrid methods. Moreover, as the whole system is solved in multiple levels each of which runs just a few iteration loops of some conventional solver, the whole system runs at an interactive frame rate.

The parameters from the finer to the coarser level or vice-versa are passed using symmetric restriction and interpolation matrices at each level l, $\mathbf{R}_l$ and $\mathbf{P}_l$.

$$\mathbf{A}_{l+1} = \mathbf{R}_l \mathbf{A}_l \mathbf{P}_l \tag{6.4}$$

Also, $\mathbf{R}_l = \mathbf{P}_l^T$. The matrices $\mathbf{A}_l$ denote linear systems matrices at l^{th} level. The number of levels depends on the initial system size and the specific demand of the application.

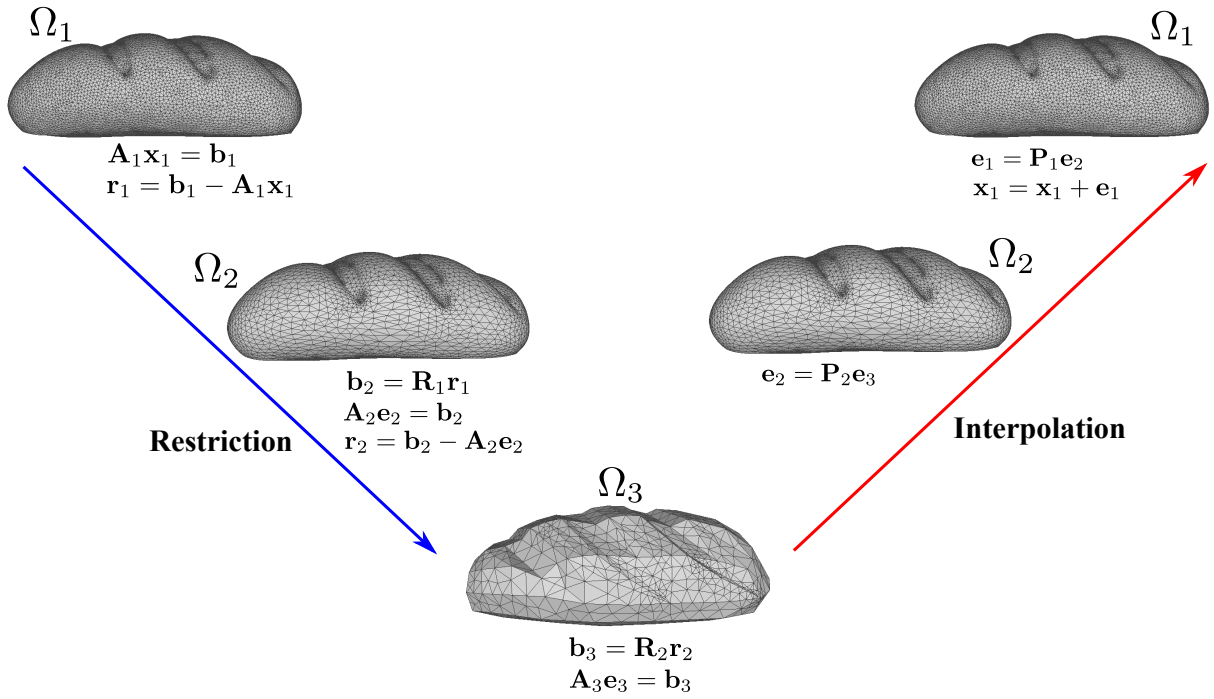


Figure 6.1: **Galerkin Multigrid Method**: The system equation is solved in multiple levels using finer to coarser meshes. At each level, the solver is terminated much earlier before convergence and the residual error is passed to the next level.

Xian et al. [2019] build a multi-level hierarchy by uniformly sampling vertices at various resolutions. Then the $\mathbf{R}_l$ and $\mathbf{P}_l$ Galerkin projection matrices are setup using skinning-space coordinates with piece-wise constant interpolation weights.

6.2.1 Building Grid Hierarchy

Given an object mesh consisting of n vertices, the coarser level mesh grids are constructed using the furthest point sampling method [Brandt et al., 2018] which is a special case of the k-means++ algorithm [Arthur and Vassilvitskii, 2007]. As depicted in Figure 6.2, let us denote the set of all the vertices, k_{g_1}, of the full resolution mesh by $\mathbf{\Omega}_1$. The vertex set, $\mathbf{\Omega}_2$, of the immediate next level coarser mesh contains k_{g_2} number of vertices and is a subset of $\mathbf{\Omega}_1$.

$$\mathbf{\Omega}_2 \subseteq \mathbf{\Omega}_1, \quad k_{g_2} \leq k_{g_1} \tag{6.5}$$

The set $\mathbf{\Omega}_2$ is first initialized with a random vertex in $\mathbf{\Omega}_1$. Using Dijkstra's algorithm, we then compute the geodesic distances to $\mathbf{\Omega}_2$ of all other vertices. Finally, the most distant vertex in

$\mathbf{\Omega}_1 \setminus \mathbf{\Omega}_2$ is picked and added to the set $\mathbf{\Omega}_2$. The geodesic distances to the new $\mathbf{\Omega}_2$ are updated in the next iteration. This process is repeated until the size of $\mathbf{\Omega}_2$ becomes k_{g_2}. In the same way, even lower-resolution grids, $\mathbf{\Omega}_3$, $\mathbf{\Omega}_4$, $\mathbf{\Omega}_5$, ... $\mathbf{\Omega}_l$, are constructed, if required.

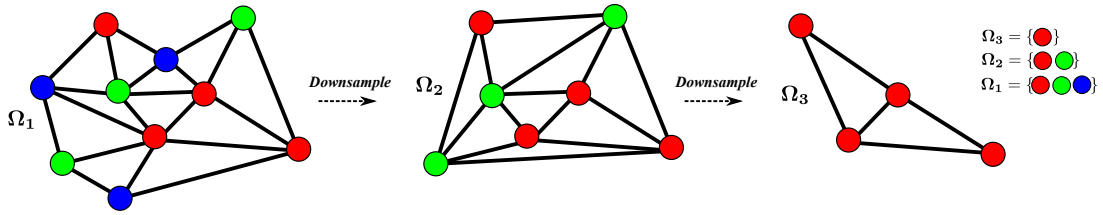


Figure 6.2: Illustration of Galerkin grid hierarchy construction. Ω_l contains the vertices on the l^{th} level grid and $\Omega_{l+1} \subseteq \Omega_l$.

6.2.2 Building Galerkin Projection Matrices

Each node in a mesh at any level is assumed to possess twelve degrees of freedom. The interpolation between different levels is constructed using Linear Blending Skinning (LBS).

$$\mathbf{x}_i = \sum_j \omega_{ij} \mathbf{T}_j \mathbf{X}_i \tag{6.6}$$

where $\mathbf{x}_i \in \mathbb{R}^{3\times 1}$ is position of i^{th} vertex, $\mathbf{T}_j \in \mathbb{R}^{3\times 4}$ is the affine transformation matrix of the j^{th} control handle, $\mathbf{X}_i \in \mathbb{R}^{4\times 1}$ is the homogeneous coordinate of the rest-pose position of vertex i, and ω_{ij} is the weight of handle j to vertex i. A visual depiction of LBS is presented in Figure 6.3. In the figure, a finer-level node (shown in blue) is controlled by four coarser-level nodes (shown in yellow). These coarser-level nodes are called control handles.

Rewriting and rearranging Equation 6.6 over all vertices gives the following relation.

$$\mathbf{X} = \mathbf{U}\mathbf{q} \tag{6.7}$$

where $\mathbf{X} = \left[\mathbf{x}_0^T, \mathbf{x}_1^T, \ldots \mathbf{x}_{n-1}^T\right]$ is the position of all vertices at the finest level;
$\mathbf{q} = [vec(\mathbf{T}_0), vec(\mathbf{T}_1), \ldots vec(\mathbf{T}_{n-1})] \in \mathbb{R}^{12k\times 1}$ denotes all the skinning space degrees of freedom; and $\mathbf{U} \in \mathbb{R}^{3n\times 12k}$ is the linear transformation matrix between $\mathbf{q}$ and $\mathbf{X}$. Each block of $\mathbf{U}$ can be written down explicitly as $\mathbf{U}_{ij} = \omega_{ij}\mathbf{X}_i^T \otimes \mathbf{I}_3 \in \mathbb{R}^{3\times 12}$ where $\otimes \mathbf{I}_3$ is a Kronecker product with a 3×3 identity matrix. Following the same argument as before, the interpolation between

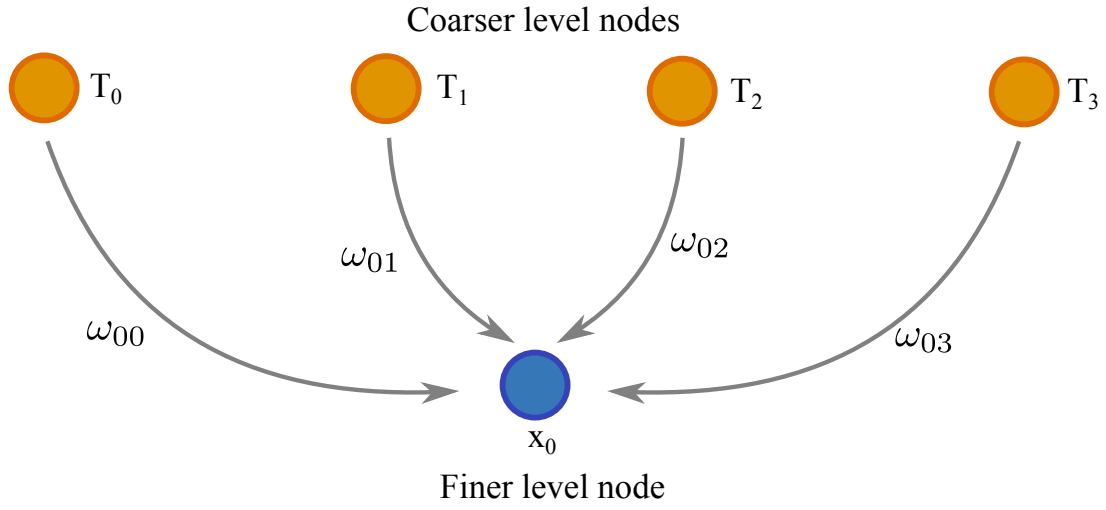


Figure 6.3: Illustration of linear blending skinning.

multiple coarser levels can be calculated as below.

$$\mathbf{q}_l = \mathbf{U}_l \mathbf{q}_{l+1} \tag{6.8}$$

Finally, the interpolation matrix $\mathbf{P}$ for different levels are defined as $\mathbf{P}_l = \mathbf{U}_l$. To maintain the symmetry of the system matrix while using the Galerkin multigrid projection method, the restriction matrix is set to $\mathbf{R}_l = \mathbf{U}_l^T$.

The weight parameters ω_{ij} can be derived using biharmonic skinning weights [Jacobson et al., 2011]. However, using biharmonic weights in the multigrid method quickly leads to huge memory consumption choking the simulation. To overcome this problem, [Xian et al., 2019] used piecewise constant weights i.e, $\omega \in \{0,1\}$. Using piecewise constant weights instead of biharmonic weights introduces more high-frequency errors in the system. However, as discussed earlier, the high-frequency errors produced due to these non-smooth weights get suppressed in the coarser levels. For further details about the Galerkin multigrid method, please refer to [Xian et al., 2019].

6.2.3 Galerkin Enhanced Graph-based FEM

The Galerkin multigrid method is utilized to simulate a framework for deformable object simulation. To initialize Galerkin multigrid, first, we construct projection matrices, $\mathbf{U}_l$ (interpolation matrix, $\mathbf{P}_l = \mathbf{U}_l$ and the restriction matrix, $\mathbf{R}_l = \mathbf{U}_l^T$) and system matrices, $\mathbf{A}_l$ for each level, l.

At each timestep, we run an iteration of graph-based FEM to compute the fracture at the finest level of the object mesh. In order to do this we calculate normal stress along the edges of the finest-level mesh using Equation 4.6. The edge is considered to be damaged if the stress value along an edge exceeds a critical threshold and subsequently uses the reformulated internal hyper-elastic strain energy density as given in Section 4.2.4 and 4.2.5. The graph-based FEM requires no volume remeshing after fracture i.e., the initial tetrahedral mesh remains the same throughout the simulation.

Therefore, the projection matrices, $\mathbf{U}_l$ of the Galerkin multigrid method at each level remain unchanged. The updated parameters, such as internal force, which are derived from the reformulated strain energy after fracture can thus be directly transferred to the coarser meshes. These updated parameters and other collision constraints are incorporated into the simulation by updating the system matrix $\mathbf{A}_1$. The propagation to coarser levels then happens naturally by $\mathbf{A}_{l+1} = \mathbf{R}_l\mathbf{A}_l\mathbf{P}_l$ (see Equation 6.4). The residuals (Equation 6.2) are similarly passed down using Equation 6.1. Finally, after solving the system equations at each level, we interpolate back the results to the finest level mesh using Equation 6.3. The whole process is summarized in Algorithm 2. If we consider cutting instead of fracture, everything remains the same except that the crack lines are initiated by the user instead of the stress threshold.

Algorithm 2: Galerkin enhanced Graph-based FEM

Set up d-level the grid hierarchy, $\mathbf{\Omega}_1, \ldots \mathbf{\Omega}_d$;

Construct the projection matrices $\mathbf{U}_1, \ldots \mathbf{U}_d$;

Compute and store the multi-level system matrices $\mathbf{A}_1, \ldots \mathbf{A}_d$;

Compute and store the multi-level external force vectors $\mathbf{A}_1, \ldots \mathbf{A}_d$;

for *Each element in object mesh* **do**

    Calculate mesh fracture using graph-based FEM;

end

Update collision and fracture constraints (e.g. internal force) to $\mathbf{A}_1, \ldots \mathbf{A}_d$ with Equation 6.4;

Compute residual at each level with Equation 6.2 ;

Pass it down to the next level with Equation 6.1;

Run specific solver (e.g. GS, CG, PCG) at each level ;

Interpolate the solution back to the finest level with Equation 6.3;

6.3 Results

In this section, we present multiple simulation examples of interactive cutting and fracture using our method. We also compare our method to the existing state-of-the-art FEM methods developed to simulate fracture.

All the experiments presented here are carried out with an Intel i7-9750H octa-core processor, 16GB DDR4 RAM and a single Nvidia RTX 2060 GPU with 6 GB of graphics memory. All the simulation examples are parallelized using CUDA.

Parameters for each simulation result are presented in Table 6.2.

6.3.1 Gelerkin Grid Setup

Before presenting the results, we quickly explain the grid structure of the Galerkin method. All the results are accompanied by a grid setup depicted as $level_n/level_{n-1}/\ldots/level_2/level_1/all$. It denotes the number of vertices at each level of the Galerkin grid mesh and '*all*' denotes the vertices of the finest level input mesh.

We perform a hyperparameter study of the grid structure. We notice that increasing the number of levels does not contribute significantly to the speed-up or accuracy of the simulation. We use volume gain to characterize accuracy (explained in Section 6.3.4) in these results. In multigrid Galerkin, the solver at each level is terminated after a few fixed used-defined number of steps, even before the convergence is attained. However, if the number of these steps is increased, the error gets reduced at the expense of simulation time. For example, we simulated a sphere hanging under gravity (see Figure 6.7) with a 50/all grid setup. At the finest level, we use Preconditioned Conjugate Gradient (PCG) solver with 3 cycles and at the coarser level, Conjugate Gradient (CG) solver with 20 cycles. The simulation runs at 118.7 frames/sec with a volume error of 19%. However, if the cycles of PCG and CG are increased to 30 and 200 respectively, the simulation runs at 23.8 frames/sec with a volume error of 11%.

6.3.2 Fracture with Galerkin enhanced graph-based FEM

In Figure 6.4, we render the fracture of an armadillo mesh using Galerkin multigrid method augmented with graph-based FEM. In the figure, the left arm of the armadillo is detached from its body when pulled by the user. The armadillo model consists of 160k tetrahedra and runs at 34.9 frames/sec with a grid setup of 50/1000/all. This same cutting simulation of the armadillo mesh requires 1.4 sec/frame, if Is-XFEM is used. Therefore Galerkin multigrid with graph-based FEM is around $\times 25$ faster than Is-XFEM.

Figure 6.4: Illustration of an original (left) and fractured (right) armadillo mesh.

6.3.3 Comparison Study

6.3.3.1 Normal Graph-based FEM vs Galerkin enhanced Graph-based FEM

We compare the performance of our method Galerkin multigrid method with normal graph-based FEM. In Figure 6.5, we simulate the fracture of a steak. The steak mesh consists of 186.1k tetrahedra and runs at 0.75 frames/sec when regular graph-based FEM is used. However, Galerkin multigrid method combined with graph-based FEM accelerates the simulation speed by nearly $\times 33$ at 24.7 frames/sec with a grid set up of 50/1000/all. However, as is evident

from the figure, normal graph-based FEM is better at preserving the intricate fracture patterns compared to our method. So the gain in accuracy comes at the cost of some speed, however, the simulation still stays physically plausible.

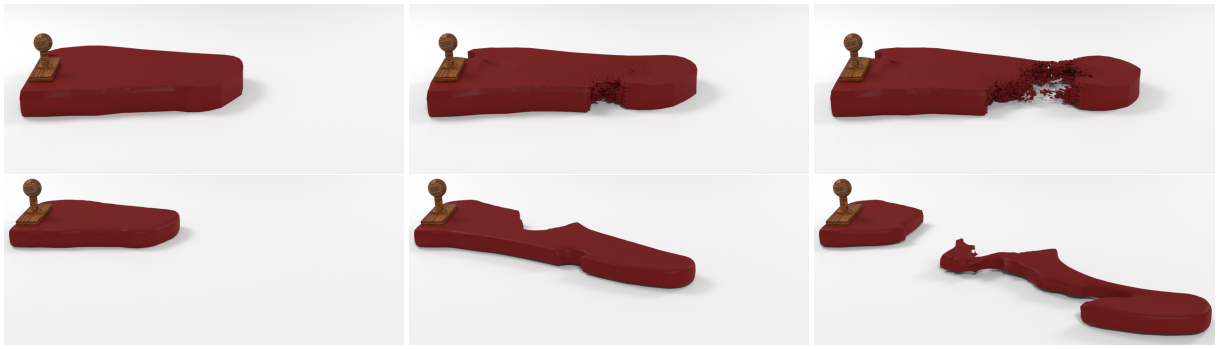


Figure 6.5: A piece of steak consisting of 186.1k tetrahedra is torn apart from pulling at one end. This is simulated using normal graph-based FEM (top row) and Galerkin-enhanced graph-based FEM (bottom row). Normal graph-based FEM runs at 0.75 frames/sec whereas Galerkin-enhanced graph-based FEM runs at 24.7 frames/sec.

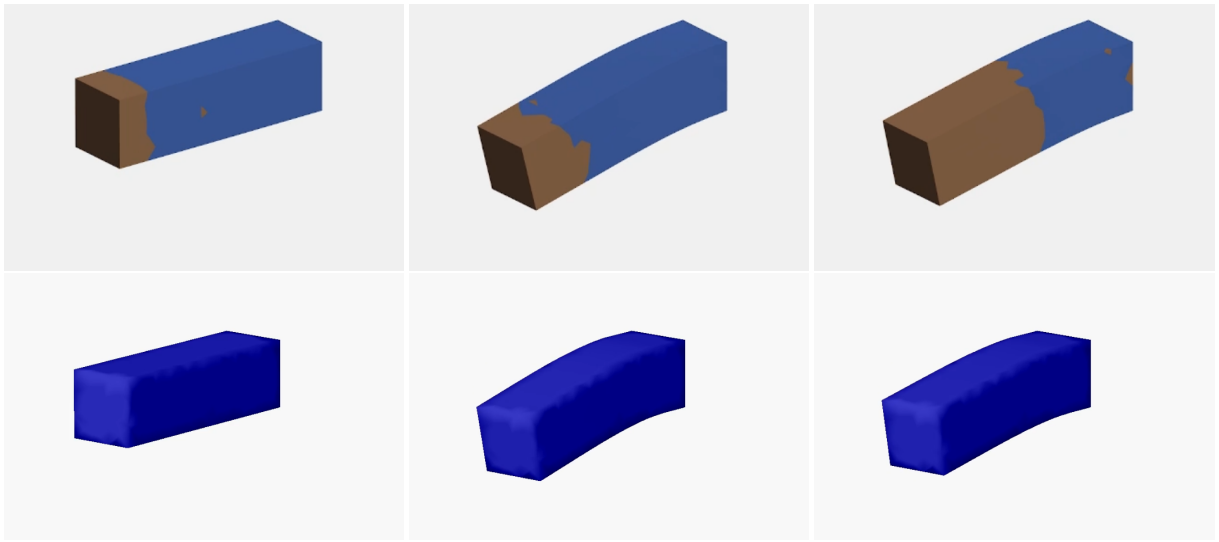


Figure 6.6: A beam is hinged at one end and released under gravity. Adaptive rigidification (Top Row) runs at 50.7 frames/sec while Galerkin multigrid combined with Graph-based FEM runs at 320.9 frames/sec.

6.3.3.2 Adaptive Rigidification vs Galerkin enhanced Graph-based FEM

Here we compare our method with the adaptive rigidification technique [Mercier-Aubin et al., 2022b]. As adaptive rigidification does not permit fracture of solid we limit our comparison

up to the deformation of the solid and simulation run-time. In Figure 6.6, the deformation of a rectangular block, consisting of 4k tetrahedra, is simulated using both adaptive rigidification (top row) and our method (bottom row). The block is hinged at one end and released under gravity. The brown portion of the block during the adaptive rigidification method depicts the rigidified solid portion of the block which does not contribute to the FEM simulation. Adaptive rigidification runs at 50.7 frames/sec while our method runs at 320.9 frames/sec with a grid set up of 5/all. Thus, while the visual simulation remains similar, a speed-up of around $\times 6$ can be gained using our method.

6.3.4 Volume Gain

Even though our method runs much faster than the existing state-of-the-art deformation simulation techniques, the gain in speed comes at the cost of some accuracy. We compare the accuracy of the Galerkin-enhanced graph-based FEM with other existing algorithms in terms of volume gain as shown in Table 6.1. In Figure 6.7, we simulate a sphere that hangs under gravity and is hinged at the top. This hanging sphere is simulated with different algorithms using various hyper-elastic strain energies as explained in Table 6.1. As evident from the table, compared to our method, only Co-rotational FEM [Müller and Gross, 2004] which uses linear strain energy performs poorly in terms of volume preservation. However, more recent FEM deformation simulation techniques like Invertible Principal-Stretch Material design [Xu et al., 2015] [Sin et al., 2013], stable Neo-Hookean [Smith et al., 2018] introduce less error than our method.

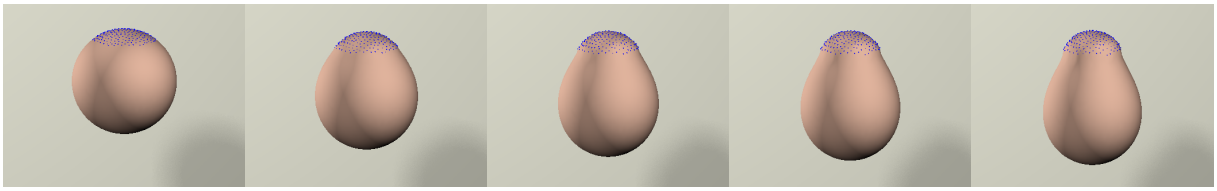


Figure 6.7: A sphere mesh hangs under gravity which is fixed at the top.

Method	*Volume Gain*
Co-rotational FEM [Müller and Gross, 2004]	85%
Invertible Neo-Hookean [Sin et al., 2013] [Xu et al., 2015]	3.2%
Invertible StVK [Sin et al., 2013] [Xu et al., 2015]	10.7%
Stable Neo-Hookean [Smith et al., 2018]	4.3%
Our Method [Xian et al., 2019]	**19**%

Table 6.1: Volume gain in different existing algorithms using various hyper-elastic strain energies.

6.3.5 Experimental Validation

We simulate a Charpy Impact test using our method and with regular graph-based FEM. In the experiment, a notched steel specimen is held on the two ends and a swinging pendulum hits it in the middle as depicted in Figure 6.8. After the collision, the specimen gets split into two disjoint pieces.

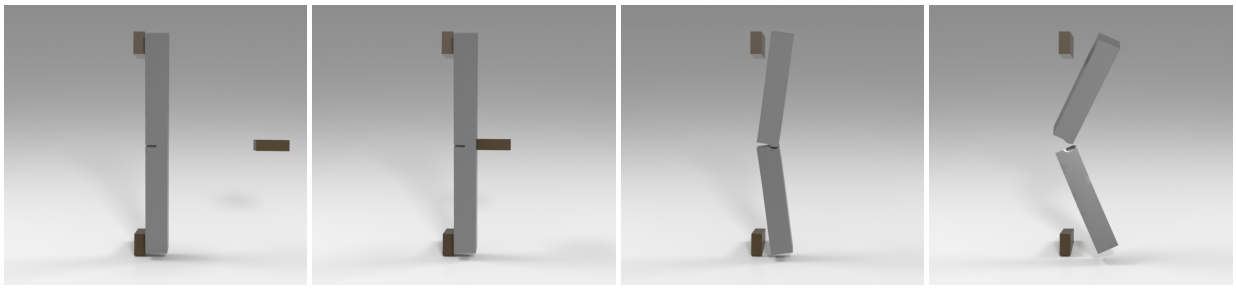


Figure 6.8: **Charpy Impact Test with Galerkin enhanced graph-based FEM**: The left image shows the configuration of the Charpy test. When hit by a swinging pendulum (shown with the moving square block), the notched specimen gets split into two pieces.

The same test is performed using normal graph-based FEM as depicted in Figure 6.9. As evident from the images graph-based FEM has better at preserving finer details and rendering correct deformation. The notched bar consists of 28k tetrahedra. While Galerkin multigrid method runs at 152.5 frames/sec with a grid set up of 50/all, regular graph-based FEM runs at around 4.7 frames/sec.

In Figure 6.10, we plot the load-crack mouth opening displacement (CMOD) curve obtained from our simulated Charpy impact test experiment. The solid blue line with a shaded

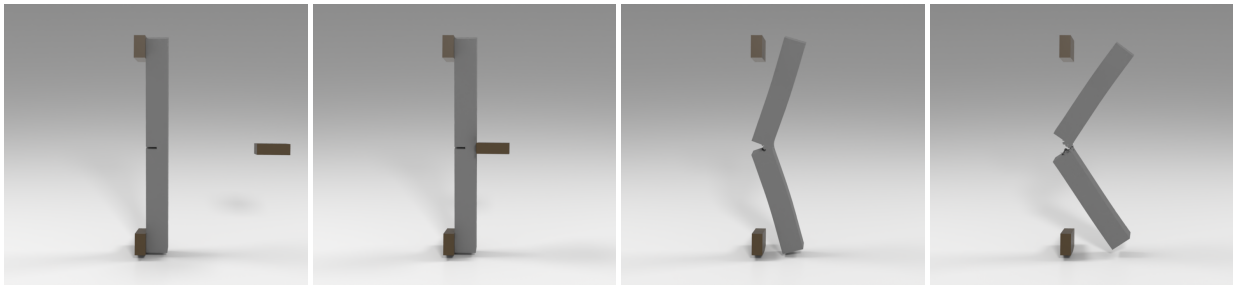


Figure 6.9: **Charpy Impact Test with normal graph-based FEM**: The left image shows the configuration of the Charpy test. When hit by a swinging pendulum (shown with the moving square block), the notched specimen gets split into two pieces.

envelope is the experimental load-CMOD curve from [Areias and Belytschko, 2005] and the dotted magenta line represents the same curve derived from the XFEM simulation reported by [Areias and Belytschko, 2005]. The dashed red line and green dashed-dotted line denote the plot for our simulation set-up using Galerkin-enhanced graph-based FEM and normal graph-based FEM respectively. The figure shows that our experimental results closely follow the real-world laboratory experiment results.

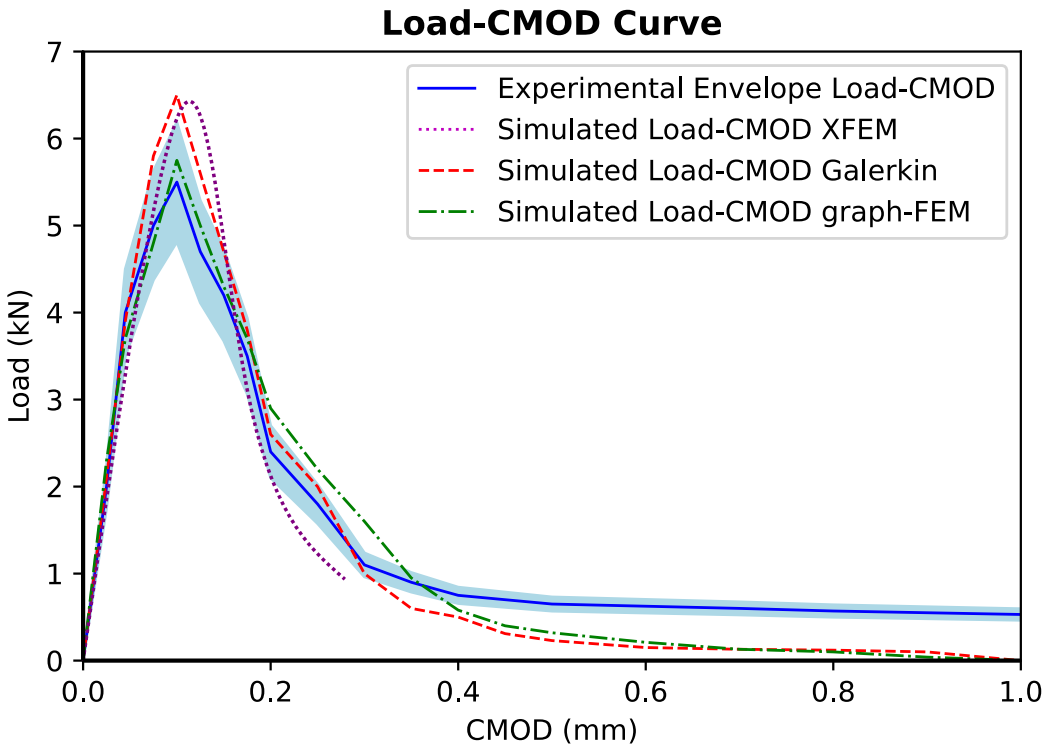


Figure 6.10: The plot compares the load-crack mouth opening displacement (CMOD) curve of the simulated Charpy impact test with ground truth curves from actual laboratory experiments.

Simulation	*#Tet*	*Grid Setup*	*fps*
Armadillo (Fig. 6.4)	160k	50/1000/all	34.9
Sphere (Fig. 6.7)	34.9k	50/all	118.7
Steak (Fig. 6.5)	186.1k	50/1000/all	24.7
Rectangle block (Fig. 6.6)	4k	5/all	320.9
Charpy test (Fig. 6.9)	28k	50/all	152.5
Human body (Fig. 7.11)	40k	50/all	127.2
Human body (Fig. 7.16)	40k	50/all	132.3

Table 6.2: Parameters for simulation experiments.

6.4 Discussion

In this chapter, we build an interactive, real-time fracture simulation framework using Galerkin multigrid and graph-based FEM. The framework can render fracture for very high-resolution mesh at an interactive rate. Moreover, it does not introduce any computational overhead irrespective of the number of cracks introduced inside the object mesh. In the next chapter, we demonstrate a complete virtual sculpting application for object manipulation. The sculpting framework is developed using algorithms and techniques that are proposed so far.

Chapter 7

Interactive Sculpting Application

7.1 Introduction[1]

Combining all the algorithms and techniques that are discussed so far, we build an application for stable simulation of interactive, physics-based virtual sculpting. Sculpting is an art that relies on the perception of material by touch, in addition to its visual appearance. Traditionally simulated virtual sculpting tools offer a visual rendering of the object being sculpted but entirely miss the tactile aspect of the art form. Our framework is coupled with appropriate haptic feedback that allows for tactile interaction with the material being sculpted.

[1]Portions of this chapter are adapted from *Interactive Physics-Based Virtual Sculpting with Haptic Feedback* by Avirup Mandal, Parag Chaudhuri, and Subhasis Chaudhuri, published in ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D), 2022 and from *Real-time Physics-based Mesh Deformation with Haptic Feedback and Material Anisotropy* by Avirup Mandal, Parag Chaudhuri, and Subhasis Chaudhuri, published in International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - GRAPP, 2023

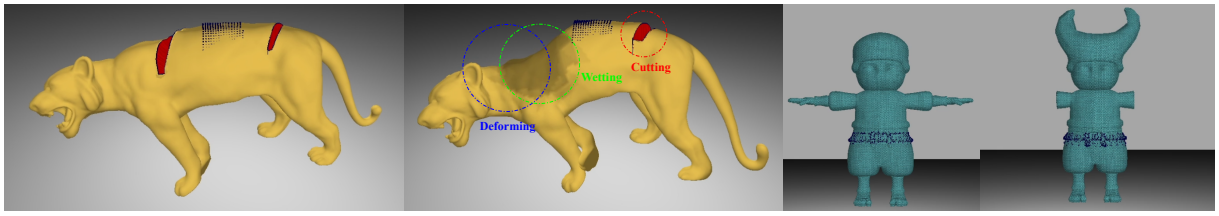


Figure 7.1: From left to right we show multiple cuts and an example with multiple sculpting operations on the Lioness model. Finally, the last two images on the right depict original and sculpted versions of a Toy Ninja model.

Moreover, many of the existing sculpting tools like Z-brush [Pixologic, 2023], Blender [Blender, 2023] etc. perform shape editing in a purely geometric approach which does not capture the accurate physical behaviour of the material. Physically accurate mesh manipulation provides digital artists with a more natural and effective sculpting experience. In our framework, the forces needed for the haptic feedback and the accurate behaviour of the material being sculpted under these forces are generated using a robust physics-based simulation. Our sculpting framework can be used for reshaping 3D models starting from any arbitrary initial mesh.

Sculpting can be performed on stone, marble, clay and other materials. In our work, we assume a clay-like material for sculpting as three major operations involving sculpting can be performed on it: deforming, wetting and cutting. We provide three sculpting brushes for these three operations.

- **Deforming**: An artist deforms a lump of clay to shape it.
- **Wetting**: Adding water to the clay makes it malleable, which in turn helps to reshape parts of the same model differently. This allows the user to deform the object differently in different parts while applying the same force.
- **Cutting**: Cutting is performed to make different shapes.

We develop a fully functional sculpting framework which includes deformation, wetting and remeshing-free cutting of an object mesh. Our sculpting brushes allow a user to perform these sculpting operations virtually while receiving the appropriate haptic feedback for the same. In Figure 7.1, on the left two images, a Lioness mesh is made wet, deformed and cut using our sculpting brushes. The two rightmost images illustrate an original toy ninja model and a sculpted version of it respectively.

To perform accurate virtual sculpting simulation, at an interactive rate, we have to develop robust and efficient algorithms for object deformation, wetting and cutting which are fast and real-time. For a realistic user experience, we require a high-fidelity visual representation of a virtual scenario coupled with accurate haptics force feedback. Moreover, visual and haptic feedback demand vastly different rate of update for smooth interaction and synchronization between them is a major challenge.

In the course of the work, we first attempt to develop the sculpting framework using the Is-XFEM technique. However, as Is-XFEM does not run in a real-time, interactive manner for high-resolution meshes (5k tetrahedra in our system), the corresponding surface mesh of the low-resolution volumetric mesh affects the visual fidelity in a negative way. Thus, we propose a multi-resolution, multi-timescale sculpting framework coupled with synchronized haptic and visual feedback. The multi-resolution part of our framework denotes that the physical simulation is computed on a low-resolution coarse mesh while a high-resolution surface mesh is embedded inside it for visualization purposes. The demand disparate refresh rate for visual and haptic feedback ($\sim$ 60 and $\sim$ 1000 fps respectively) is met by our multi-timescale framework.

As discussed in Section 3.4 of Chapter 3, Is-XFEM suffers from an explosion of system matrix size; thus, making the simulation non-interactive for a large number of cuts. However, our graph-based FEM integrated with Galerkin multigrid method (see Chapter 6) can smoothly scale to high-resolution meshes containing a large number of cuts. Thus, our final interactive sculpting framework is built using Galerkin multigrid with graph-based FEM. As it is evident, the multi-resolution framework is not required in this method. However, the multi-timescale framework is still used to meet the refresh rate needed for faithful haptic interaction.

7.2 Technical Overview

7.2.1 Wetting Porous Object

Fluid flow and material wetting is a well-studied subject in material physics [Bear, 1988] [Adler, 1992]. In computer graphics, work by Patkar and Chaudhuri [2013] offers a geometrically

modelled solution to the absorption, diffusion and dripping of water in porous materials. In more recent work [Fei et al., 2018], the wetting of different kinds of clothes is explored. For wetting the material, we follow the method proposed in [Patkar and Chaudhuri, 2013] barring the dripping part of the algorithm that reduces the fluid content in the object. Once collision occurs between the wetting tool in our framework and the boundary of the tetrahedral mesh, the fluid content in the tetrahedrons in contact with the tool increases in incremental steps till the saturation value becomes one. The saturation of a tetrahedron is defined as $S_w = m^w/V^e$. Here m^w is the mass of water absorbed and V^e is the volume of the tetrahedron. After the absorption of fluid, diffusion happens between any two neighbouring tetrahedra depending on the saturation gradient between them [Patkar and Chaudhuri, 2013]. A drying tool is provided whose action is complementary to the wetting one.

7.2.2 Variable Elasticity

The change of material properties due to fluid absorption is well investigated in material science [Yoon and Cowin, 2009] [Schraad, 2014]. Here the authors hypothesize a mathematical relationship between object elasticity with fluid content and finally verify their hypothesis with empirical results. In order to formulate a relationship between fluid content and elasticity of the material we follow the line of thought presented in [Yoon and Cowin, 2009].

Given linear Cartesian strain ε_c^e and stress σ_c^e for an element Δ_e, we can write the expression $\sigma_c^e = \mathbf{E}\varepsilon_c^e$ where $\mathbf{E}$ denotes elasticity tensor (see Section 4.2.4.1 for more details). Similarly, compliance tensor $\mathbf{S}$ is defined as $\varepsilon_c^e = \mathbf{S}\sigma_c^e$ with $\mathbf{S} = \mathbf{E}^{-1}$. The Voigt upper bound on the elasticity tensor of a solid-fluid mixture is given by

$$\mathbf{E}^V = (1-\phi)\,\mathbf{E}^s + \phi\mathbf{E}^w \tag{7.1}$$

where $\mathbf{E}^V$, $\mathbf{E}^s$ and $\mathbf{E}^w$ are the elasticity tensor of a solid-fluid mixture, solid and fluid respectively. The quantity ϕ denotes the fluid volume fraction in the solid. The Reuss lower bound on the compliance tensor of a solid-fluid mixture with any kind of solid and fluid is given by

$$\mathbf{S}^R = (1-\phi)\,\mathbf{S}^s + \phi\mathbf{S}^w \tag{7.2}$$

where $\mathbf{S}^R$, $\mathbf{S}^s$ and $\mathbf{S}^w$ are the compliance tensor of a mixture, solid and fluid respectively. The

Voigt and Reuss bounds together give the bound on the effective elasticity tensor as

$$\left(\mathbf{S}^{\mathbf{R}}\right)^{-1} \leq \mathbf{E}^{eff} \leq \mathbf{E}^{V} \tag{7.3}$$

Putting everything together, the effective compliance tensor for the solid-fluid mixture with any kind of solid and fluid is given by

$$\mathbf{S}^{eff} = \left[\mathbf{1} + \phi \left(\mathbf{Q}^{I} - \mathbf{P}^{w}\right)^{-1}\right] \mathbf{S}^{s} \tag{7.4}$$

where $\mathbf{Q}^{I} \equiv \left(\mathbf{E}^{s} - \mathbf{E}^{w}\right)^{-1} \mathbf{E}^{s}$ and $\mathbf{P}^{w}$ is the Eshelby tensor [Eshelby, 1957] for fluid inclusion inside solid. Using this formulation we determine the effective elastic tensor of a solid-fluid mixture system. We always use water for fluid in our framework. As the change of elasticity is dependent on the fraction of water content in the tetrahedral element and water content is dependent on the gradient of saturation, there is never any abrupt change of elasticity in the model, thus maintaining the stability of the system. The effect of wetting is illustrated in Figure 7.12 and Figure 7.13.

7.2.3 Haptic Rendering

Haptic rendering of mesh-based solid objects goes back to seminal work by Zilles and Salisbury [1995]. In this work, the authors present a God Object-based haptic rendering method. In this method, the movement of a god object is constrained up to the outer surface of an object mesh while the corresponding haptic proxy penetrates the outer surface to go inside the object mesh. The difference between the acceleration of the god object and the haptic proxy generates haptic feedback. Ortega et al. [2007] improved the original god object method, which was constrained to three degrees of freedom and extended it to all six degrees of freedom. Another popular method for rendering haptic force is penalty-based rendering [Barbič and James, 2009] [Otaduy and Lin, 2005] [McNeely et al., 1999]. In this method, when two or more colliding objects penetrate each other, we calculate force feedback depending on the depth of penetration among these objects. However, discrete penalty-based force rendering is often discontinuous and jerky especially when the contact stiffness is high. Continuous collision detection (CCD) [Tang et al., 2012] alleviates these problems by integrating the force over the contact time intervals between two colliding meshes. Xu and Barbič [2017] developed a method to compute haptic feedback between points and signed distance field using CCD. In

our work, we use the continuous penalty-based method for our haptic feedback as it produces smoother force feedback [Xu and Barbič, 2017].

We render faithful haptic feedback to implement interactive virtual sculpting. The haptic interaction process while sculpting an object consists of the following components:

- Continuous Collision Detection (CCD) between the haptic proxy and outer surface of the tetrahedral simulation mesh.
- Continuous penalty-based haptic rendering while deforming and cutting the mesh.

All sculpting operations and haptic force feedback for them are performed on volumetric simulation mesh with tetrahedral elements. In order to visually render the sculpt in high quality, we transfer the deformations and all sculpting operations to a higher resolution visualization surface mesh as explained in Section 7.3.

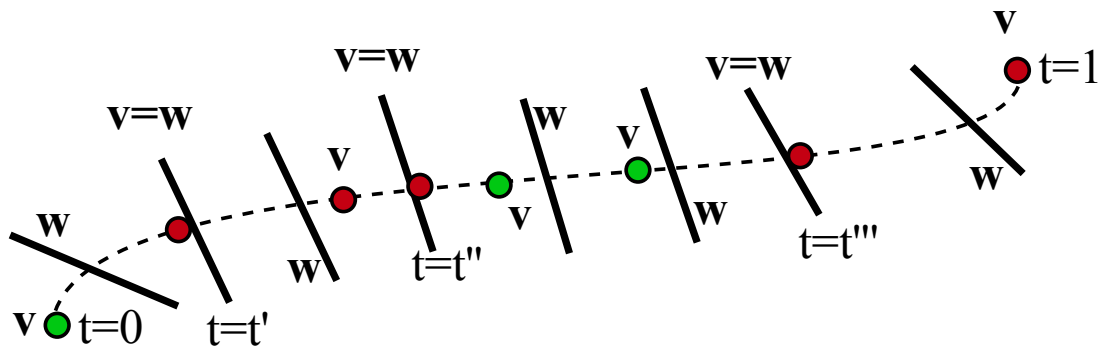


Figure 7.2: Three contact times are $t^{'}, t^{''}, t^{'''}$ when the vertex 'v' comes in contact with outer plane 'w' of an object mesh. Penetration intervals are $[t^{'}, t^{''}]$ and $[t^{'''}, 1]$. During these intervals, vertex 'v' crosses the outer plane 'w' and goes inside the object mesh. Time step Δt is normalized to $[0, 1]$.

7.2.3.1 Continuous Collision Detection

The continuous collision detection method is performed between the outer boundary of the tetrahedral object mesh and haptic proxy, both of which consist of triangular primitives. When two triangular face elements collide, we check for two kinds of collisions: vertex-face and edge-edge. We begin by interpolating the positions of each primitive i.e., vertex, edge and face in the simulation time step, Δt, normalized to $[0, 1]$ (as shown in Figure 7.2). Finally, a 3^{rd}

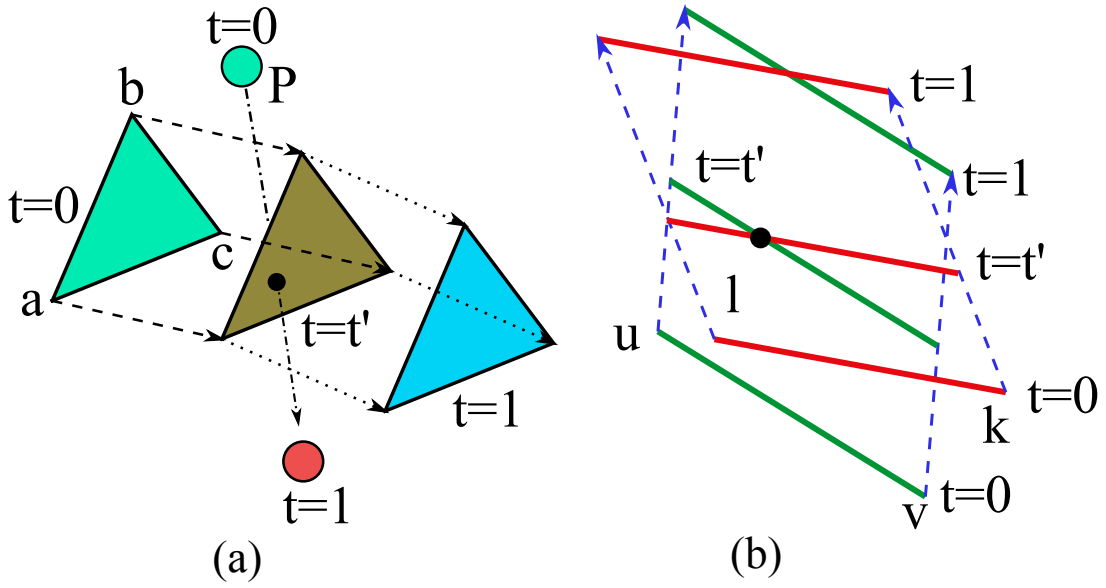


Figure 7.3: Continuous collision between (a) vertex-face and (b)edge-edge

order polynomial equation in t is solved to deduce the number of collisions that occurs during that particular simulation time step. For faster detection of the collisions, we implement Axis Aligned Bounding Box method for each of the triangles along with the non-penetration filter-based algorithm presented in [Tang et al., 2010b].

7.2.3.2 Vertex-face penalty force

A vertex-face collision occurs when any vertex of the haptic proxy collides with any of the triangles of the object mesh boundary or vice-versa. In this case, we calculate a penalty force [Tang et al., 2012] as

$$\mathbf{I}_p^{VF} = k_{vf} \sum_{i=0}^{i<N} \int_{t_a^i}^{t_b^i} \mathbf{n}_t^T \left(\mathbf{p}_t - \mathbf{q}_t\right) \mathbf{n}_t dt \tag{7.5}$$

where k_{vf} is a scalar stiffness constant. Time intervals $[t_a^i, t_b^i] \in [0,1]$ are called penetration time intervals which are defined as time segments while the vertex is inside the object mesh (see Figure 7.3(a)). The parameters i, $\mathbf{n}_t$, $\mathbf{p}_t$ and $\mathbf{q}_t$ denote the number of penetration time intervals, contact normal, position of vertex and contact point on boundary mesh respectively during Δt. Using barycentric coordinates of three vertices of a triangular primitive, the position of the collision point on the triangle can be expressed as $\mathbf{q}_t = w_a\mathbf{a}_t + w_b\mathbf{b}_t + w_c\mathbf{c}_t$. The penalty force $\mathbf{I}_p^{VF}$ is applied to the object mesh. A reaction force of the same magnitude but opposite direction is applied to the proxy.

7.2.3.3 Edge-edge penalty force

Similar to vertex-face penalty force, we calculate penalty force $\mathbf{I}_p^{EE}$ if a collision occurs between the edge of the haptic proxy mesh and the edge of the simulation mesh boundary of the object ((see Figure 7.3(b))). The penalty force is calculated as

$$\mathbf{I}_p^{EE} = k_{ee} \sum_{i=0}^{i<N} \int_{t_a^i}^{t_b^i} \mathbf{n}_{E_t}^T (\mathbf{p}_t - \mathbf{q}_t)\, \mathbf{n}_{E_t} dt \tag{7.6}$$

where k_{ee} is a scalar stiffness constant. Other parameters remain the same as before. The position of the collision point on the two edges can again be expressed using barycentric coordinates of two vertices of the edge as $\mathbf{p}_t = w_a \mathbf{a}_t + w_b \mathbf{b}_t$ and $\mathbf{q}_t = w_c \mathbf{c}_t + w_d \mathbf{d}_t$. Like before, the penalty force and the corresponding reaction force are then applied to the object mesh and haptic proxy respectively.

7.2.3.4 Haptic Rendering for Mesh: Deformation

We classify the deformation of a mesh in two categories: (1) push deformation of mesh and (2) pull deformation of mesh, both of which follow the same principle except the applied force direction which is inward for push and outward for pull. We detect collision using CCD in each time step, Δt, and then integrate over those particular time intervals when penetration depth between collided primitives is positive, implying that they are in a colliding state (see Figure 7.2).

Local deformation of the mesh: clay-like behaviour— While deforming the object we want to replicate a clay-like behaviour [Bergaya and Lagaly, 2013] in our model, i.e., the object should be malleable near the point where an external force is applied but the movement of the whole structure of the object should be negligible due to this external force. To this end, we define a kernel function G_d in Equation 7.7 around the position of the haptic proxy. The velocities of the object mesh are scaled with the weights of the kernel. If $r = ||\mathbf{x} - \mathbf{x}_c||_2$, then,

$$G_d(\mathbf{x}) = \begin{cases} \dfrac{1}{1 + k_1 r} & \text{if} \quad r < R_D \\ \dfrac{1}{1 + k_1 r + \exp(k_2 r)} & \text{if} \quad r \geq R_D \end{cases} \tag{7.7}$$

where $k_1, k_2 \in [1, 100]$ are stiffness constants, $\mathbf{x}_c$ denotes the position of haptic proxy and $||\cdot||_2$ denotes the l_2 norm. R_D is the influence radius of the damping kernel. As a result, velocities of points further away from the haptic proxy are more damped than closer points.

7.2.3.5 Haptic Rendering for Mesh: Wetting

Haptic force in the wetting tool is rendered using the same technique we employ for deforming a mesh. But unlike deformation, the mesh geometry remains unchanged.

7.2.3.6 Haptic Rendering for Mesh: Cutting

We employ the continuous penalty-based haptic force feedback rendering method for cutting the object. We consider the collision between the tetrahedral element and the 2D cut brush mesh (see Figure 7.14). We use a direct coupling method for haptic rendering, i.e., we let the cut brush, controlled by a haptic device, penetrate into the object simulation mesh. As the penalty force is directly proportional to the depth of penetration, the more the cut brush overlaps the object, the more the penalty force. When the penalty force increases beyond a certain threshold, the overlapping portion between the cut brush and the tetrahedral mesh is considered cut. The overlapping portion is computed using a fast triangle-triangle intersection algorithm [Möller, 1997] between the triangles of the cut brush and the surface triangles of a tetrahedron. For convenient user interaction, the entire cutting simulation runs in two distinct phases repeatedly.

- Marking the cut boundaries with the haptic cut brush. The movement of the object mesh is frozen during this interaction to get a clean-cut boundary.

- Letting the object mesh move due to some kind of external force e.g. gravity, without damping kernel, as directed by the underlying physical simulation.

This is done because if the object moves considerably while cutting, the penalty force threshold to trigger the cut action is never reached.

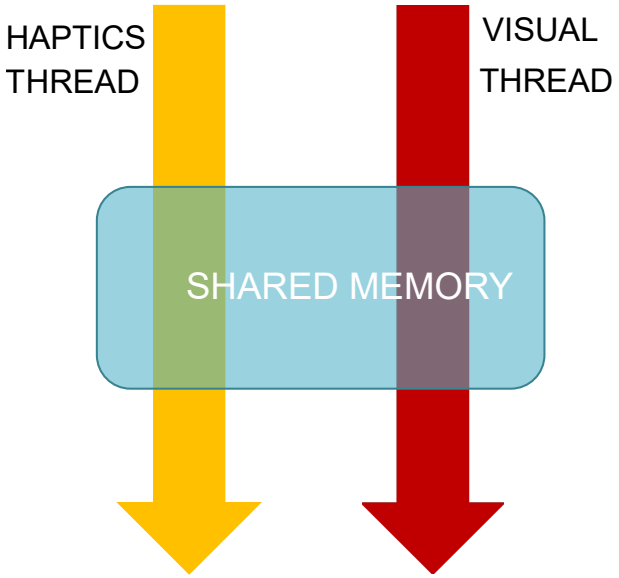


Figure 7.4: Multi-Timescale framework.

7.3 Simulation Framework

7.3.1 Multi-Timescale Haptic and Visual Feedback

For smooth haptic force feedback, a minimum refresh rate of 1000 frames/sec is required. On the other hand for smooth visual feedback, a refresh rate of 60 frames/sec is sufficient. To achieve both these requirements, the whole simulation is run in two distinct threads. We use MS Windows's in-built WINAPI package to create them. On one thread, physical simulations along with graphics rendering are performed while another thread is used for rendering haptic feedback. We have kept the haptic thread running at 1000 frames/sec all time using HAPI API which samples the haptic force feedback at the required rate. The refresh rate of the visual thread varies from $70 - 900$ frames/sec, depending on the underlying object mesh. So, the haptic thread keeps rendering the same old force feedback at the higher frame rate, until it gets a force update from the visual thread which runs at a much slower rate. The synchronization between the two threads is obtained implicitly due to the rapid update rate of the haptic thread, instead of using a blocking, explicit synchronization construct as depicted in Figure 7.4. This is a common practice followed by many haptics literature [Barbič and James, 2009][Xu and Barbič, 2017]. Because of this construct, our framework can work on high-resolution mesh

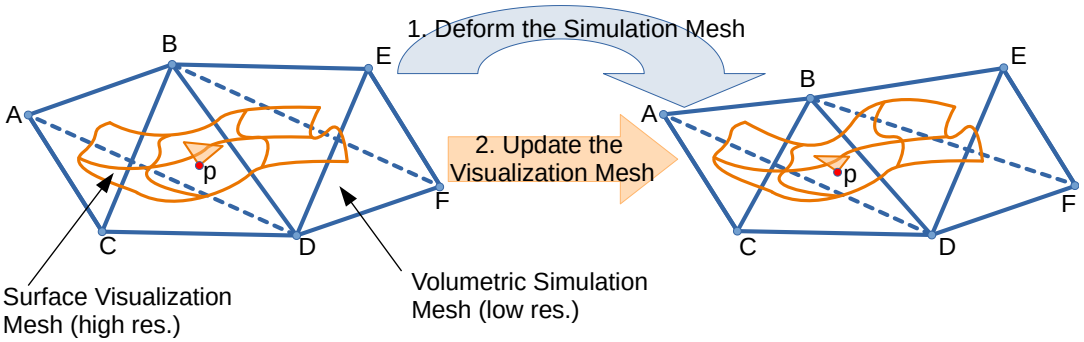


Figure 7.5: Multi-Resolution framework.

models with intricate details without degrading the quality of a user's tactile experience.

7.3.2 Multi-Resolution framework for Is-XFEM

Physics-based simulation is computationally expensive and cannot be performed on extremely high-resolution meshes at interactive rates. However, visual fidelity suffers a lot when low-resolution meshes are used. On the other hand, haptic fidelity requires simulations to run at very high frame rates. Our framework allows us to find common ground between all these disparate goals.

Our simulation runs on a coarse, low-resolution volumetric mesh with tetrahedral elements that encloses a high-resolution surface mesh with triangle elements like a cage. As shown in Figure 7.5, the vertices of the surface mesh are expressed in the local space of the simulation mesh using barycentric coordinates. When the simulation mesh is deformed, the barycentric coordinates of the surface mesh vertices in the local space of the simulation mesh do not change. This lets us calculate new coordinates of the surface mesh vertices in a global coordinate system. Similar ideas can be found in [Ju et al., 2005] [Xian et al., 2009]. In Figure 7.6 two high-resolution surface meshes, T-Rex (left) & Pink-Panther (right), and their corresponding low-resolution volumetric simulation meshes are depicted. Any manipulation performed on the simulation mesh gets transferred to the visualization surface mesh using a weighted kernel. This sets up the multi-resolution component of our framework.

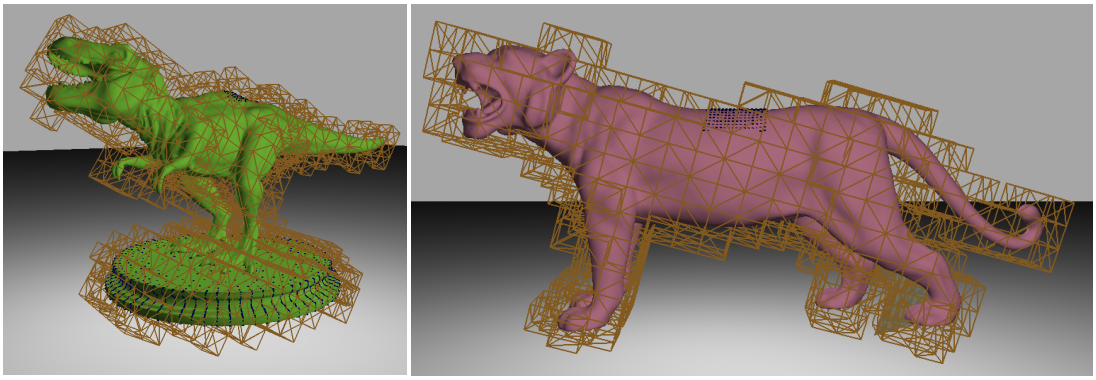


Figure 7.6: Surface visualization mesh embedded inside volumetric simulation mesh: T-Rex (left) and Pink-Panther (right).

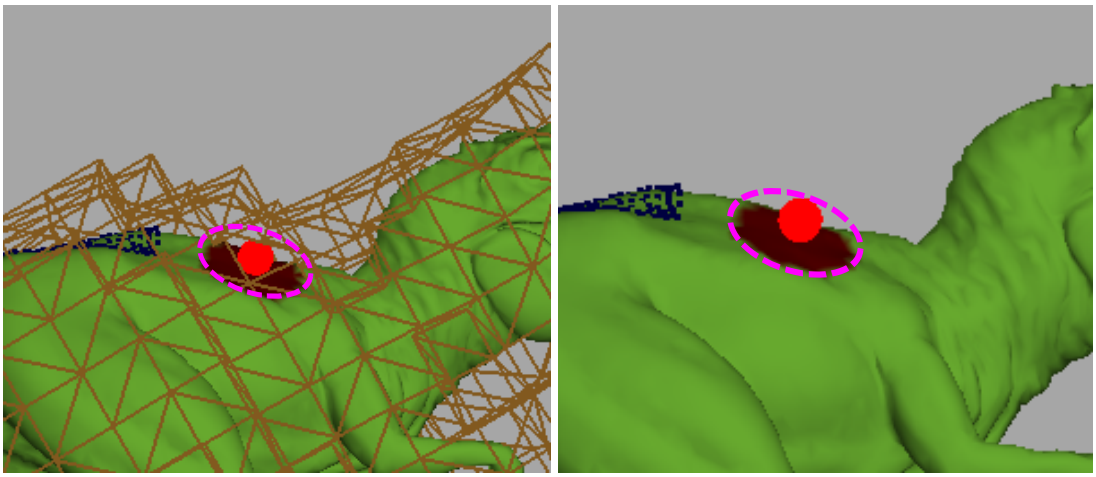


Figure 7.7: A deformation tool is colliding with the simulation mesh (left). The circled colour gradient indicates the region of deformation on the visualization mesh. The deformation is projected onto the surface mesh for visualization (right).

7.3.2.1 Transfer of Deformation

When a deformation (push/pull) tool collides with the outer surface of the tetrahedral simulation mesh, we visualize it by projecting a region with a colour gradient on the surface mesh to denote the deformation region (Figure 7.7 left). The deformations are performed on the simulation mesh. Using barycentric coordinates this deformation is then transferred to the surface mesh for visualization (Figure 7.7 right).

7.3.2.2 Transfer of Wetting

During wetting any node of the triangular surface mesh gets the same saturation value as the tetrahedron which contains it.

7.3.2.3 Transfer of Cut

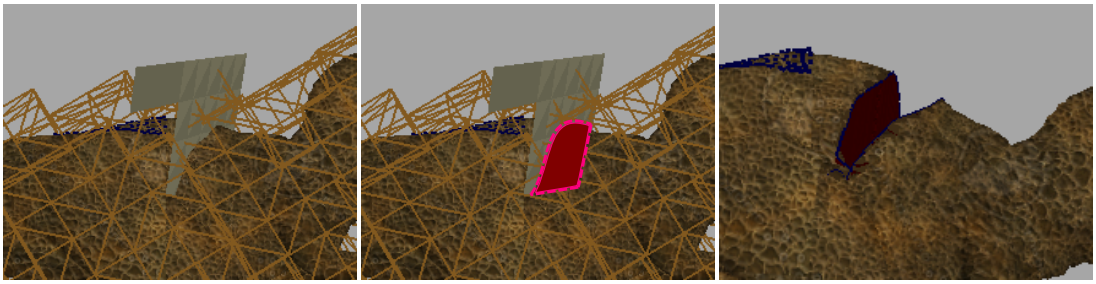


Figure 7.8: The cut brush penetrates inside the mesh (left). The cut is projected inside the surface mesh for visualization (middle). The final cut without simulation mesh (right).

When the cut brush penetrates inside the tetrahedral simulation mesh as well as the surface mesh, the cut gets initiated (Figure 7.8 left). Then the cut on the tetrahedral mesh gets projected only inside the triangular outer surface mesh for visualization purposes (Figure 7.8 middle). A user visualizes the cut-mesh without a cage (Figure 7.8 right). Finally, at the end of the sculpting operations, the tetrahedral simulation and surface visualization mesh get affected. Users can save or export any of these sculpted meshes, whether simulation or visualization, for use in other applications.

Note that, in Galerkin multigrid method, the multi-resolution framework of Is-XFEM is unnecessary. Consequently, the transfer of deformation and cut from low-resolution to high-resolution mesh is not required in Galerkin multigrid method. Moreover, while the maximum resolution of meshes on which Is-XFEM can run a simulation at an interactive rate is limited to 5k tetrahedra, in the case of the Galerkin multigrid method it can scale up to 100k tetrahedra.

7.3.3 Graphical User Interface

Figure 7.9 illustrates our Graphical User Interface (GUI) for performing interactive sculpting operations. An enlarged GUI menu is depicted on the right of Figure 7.9. Our GUI provides a user with an array of tools for various kinds of mesh manipulation and simple navigational components to orient those. Additionally, our GUI facilitates the exportation of object mesh and screenshots of frames. We have used our application for creating various sculpting examples and conducted a user study to gauge its efficacy. We present these results in the next section.

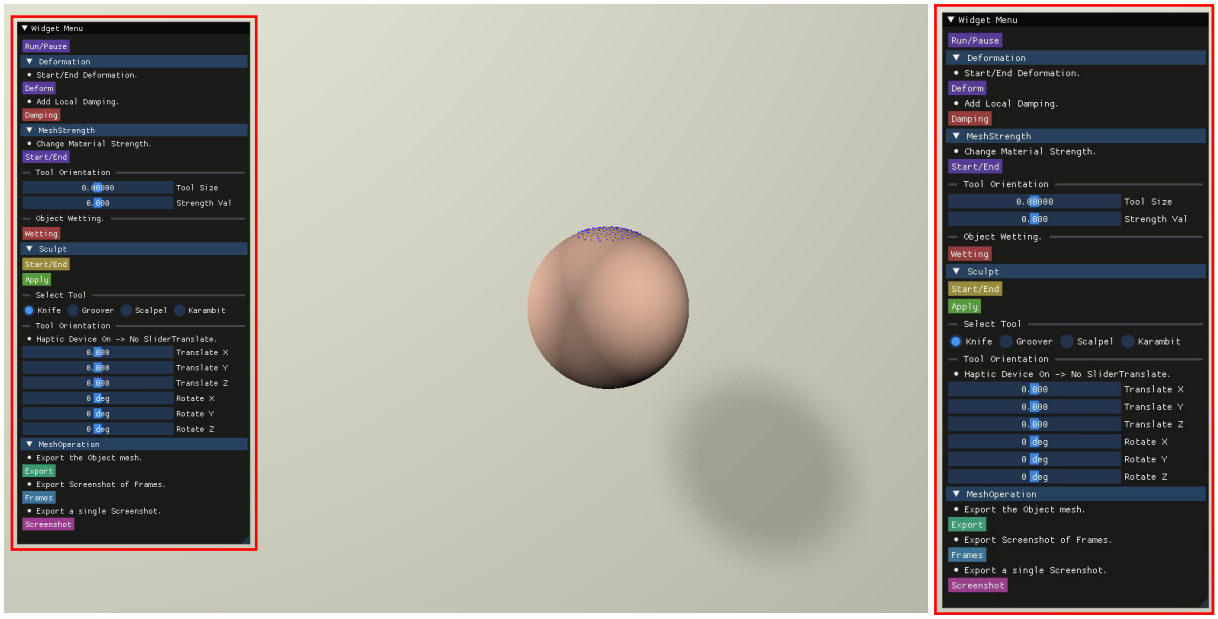


Figure 7.9: Graphical User Interface for interactive sculpting operations.

7.4 Results

In this section, we present results that help evaluate the performance of our sculpting solution and all its functionalities. First, we present a simulation overview of generating the cut surface for the purpose of visual rendering while cutting the mesh object, even though the original model is not remeshed. Further, we conduct a user study to evaluate the qualitative performance of our solution. A quantitative evaluation of our framework is also presented to affirm that we satisfy real-time interaction constraints.

All the experiments presented here are carried out in a Windows 7 operating system with Intel i7-9750H octa-core processor, 16GB DDR4 RAM, a single Nvidia RTX 2060 GPU with 6 GB of graphics memory and a 6-DOF haptic device from Geomagic Touch. We do not handle self-collision in our work.

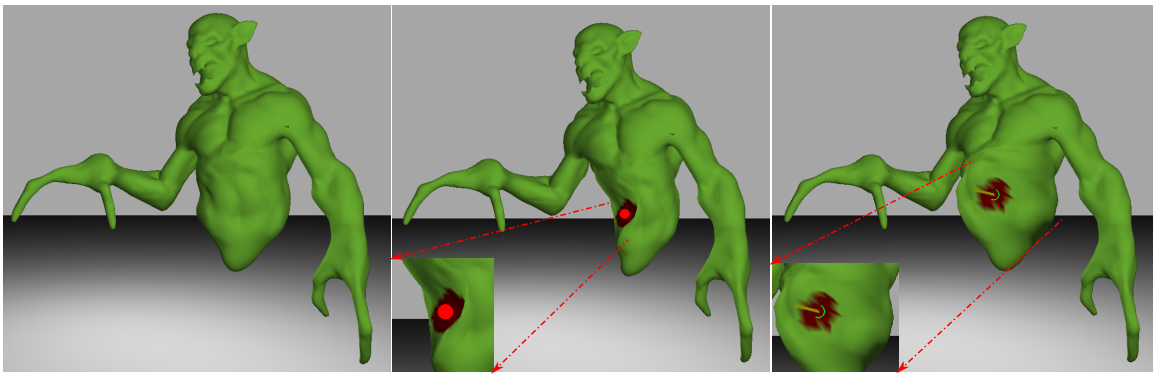


Figure 7.10: Illustration of the original model (left), the interaction of the push tool with the model (middle) and the interaction of the pull tool with the model (right).

7.4.1 Deforming

As shown in Figures 7.10, whenever a haptic proxy touches the simulation mesh, a colour gradient gets projected onto the surface visualization mesh near proxy within radius R_D as mentioned in Equation 7.7. This helps the user to get a better perception of the deformation region. In Figure 7.10, a zombie object mesh with a push deformation (middle) and a pull deformation (right) is shown. For push deformation, the mesh collapses and bulging takes place for pull deformation. The zombie model which consists of 1.2k tetrahedra is rendered using the Is-XFEM-based framework and runs at 87.9 frames/sec. Similar deformation on a human body mesh which runs the Galerkin multigrid method for physical simulation is depicted in Figure 7.11 (right). The human body mesh comprises 40k tetrahedra and runs at 127.2 frames/sec with a grid set up of 50/all. When similar deformation is rendered using Is-XFEM, it no longer remains interactive and runs at 2.7 frames/sec.

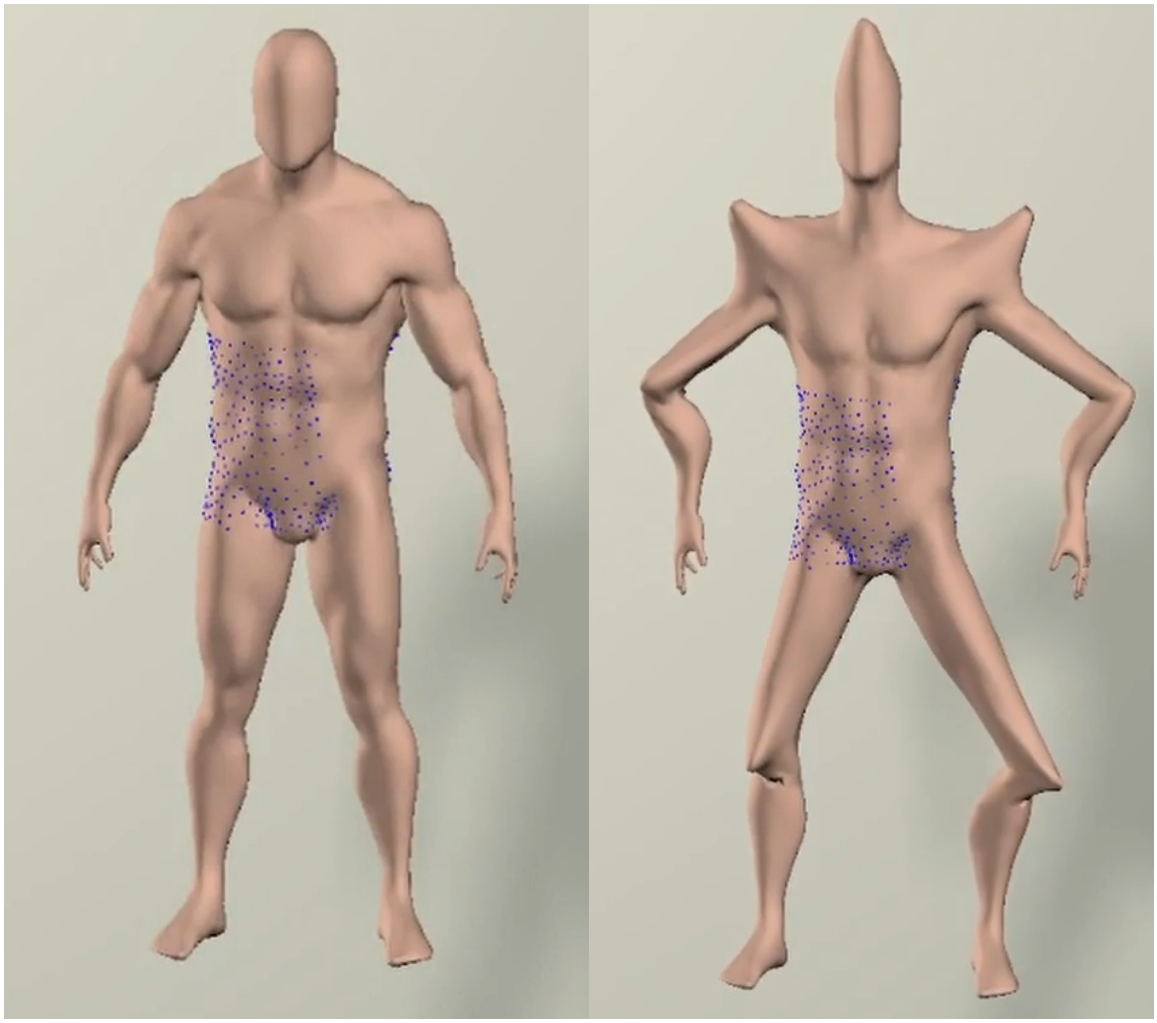


Figure 7.11: Illustration of the original (left) and deformed human body model for Galerkin multigrid method (right).

7.4.2 Wetting

Using a wetting tool we can wet material by transfer of fluid. Any vertex of the surface mesh gets the same saturation value as the tetrahedron from the simulation mesh that contains the vertex. In Figure 7.12 a user is shown interacting with a dry and a partially wet T-Rex model. Except for the effect of wetting on elasticity, the other material properties of the object mesh and the area where the user interacts remain the same in both cases. The haptic feedback during this interaction is shown in Figure 7.13. The perceptive change of haptic force feedback after fluid absorption is evident from the plot which indicates that the wet model offers less resistance compared to a dry one. Moreover, as shown in Figure 7.12, the wet portion of the mesh exhibits

more deformation due to a change in the material property after water absorption.

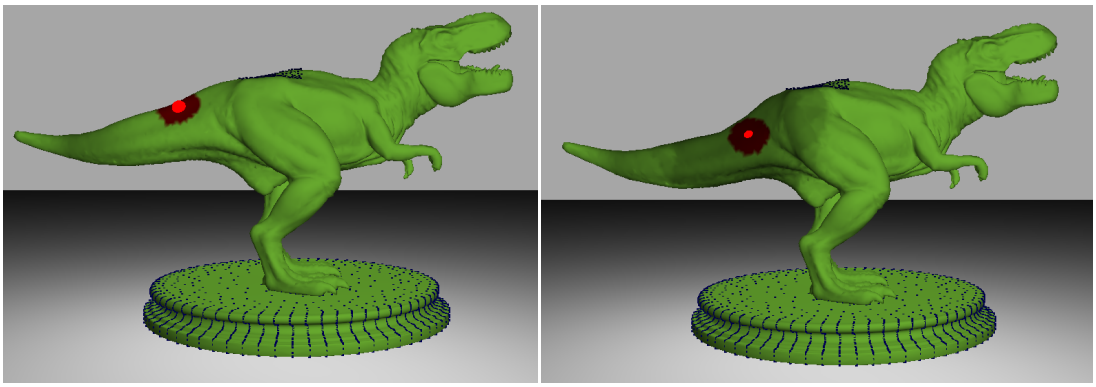


Figure 7.12: Deforming a dry (left) and partially wet (right) T-Rex model with a haptic push tool.

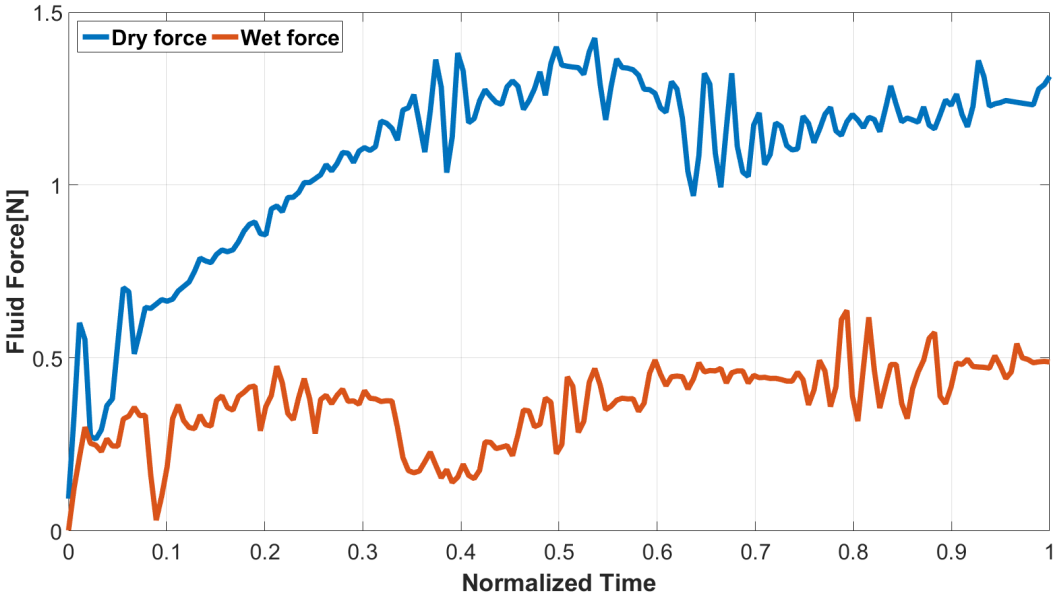


Figure 7.13: Illustration of haptic feedback force while interacting with dry and wet objects. As expected the force is much less for the wet case.

7.4.3 Cutting and Visualizing the Object Mesh

While cutting the object mesh, in order to get proper haptic feedback, we restrict the movement of the object. After the object is cut, i.e., the cut surface plane is generated, physical simulation is resumed which allows the object to move under external forces. The reason for doing so is

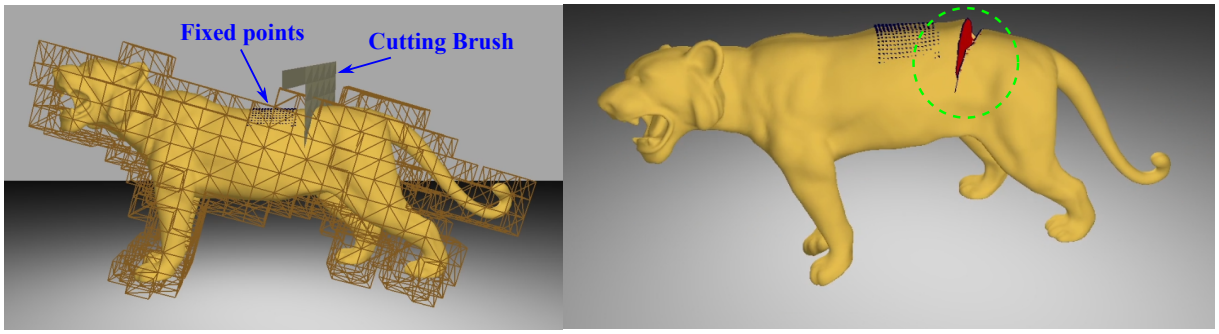


Figure 7.14: The cut brush colliding with the object mesh (left). The opening of the cut (circled in green) on the mesh (right).

that if the object moves while colliding with the cut brush then we do not get any proper overlap between the cut brush and object mesh. This closely resembles interaction in real life where we need to hold onto an object firmly in order to cut it with a tool.

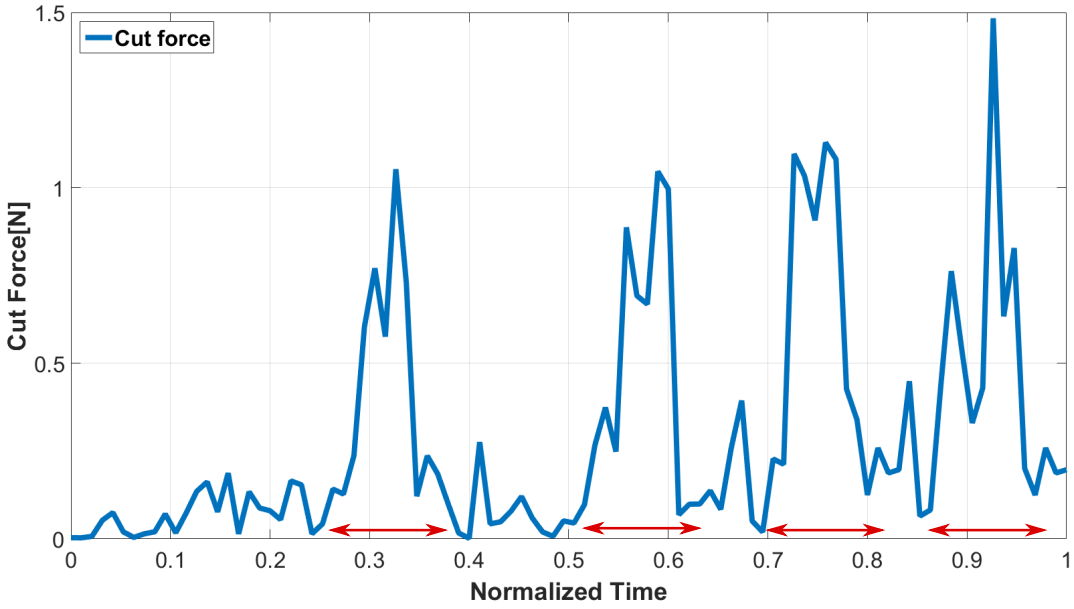


Figure 7.15: Illustration of haptic force feedback while cutting the object.

When the haptic force threshold is reached, the cut plane boundary gets generated in the overlapping region between the cut brush and object mesh. The boundary then gets projected on the surface mesh for proper visualization. This can be seen in Figure 7.14 (left). Subsequently due to the effect of gravity the cut part of the object mesh dangles (shown in Figure 7.14 (right)). In Figure 7.1 (leftmost image) we have shown multiple cuts on a Lioness model. The haptic force feedback for cutting the object mesh is shown in Figure 7.15. It can be seen, the feedback force is low when the cut brush is not interacting with the object mesh. As it penetrates inside

the object, marked by red double-headed arrows in the figure, the force increases sharply and once the force threshold is crossed, the object gets cut. The force feedback falls to a low value again after that. In Figure 7.15 haptic force feedback of four cuts is shown. In the above two examples, the cut is simulated using Is-XFEM. In Figure 7.14 Lioness mesh consisting of 1.3k tetrahedra runs at 122.5 frames/sec when no cut is present and the simulation rate drops to 109.7 frames/sec when a single cut is introduced.

Using the Galerkin multigrid method with graph-based FEM, a similar cutting example of a human body mesh is depicted in Figure 7.16. The human body mesh consists of 40k tetrahedra and runs at 132.3 frames/sec with a grid set up of 50/all, irrespective of the number of cuts introduced. On the other hand, while using Is-XFEM, the whole simulation runs at 4.1 frames/sec when no cut is present and at 3.5 frames/sec when one cut is introduced. The frame rate keeps on decreasing as the number of cuts increases.

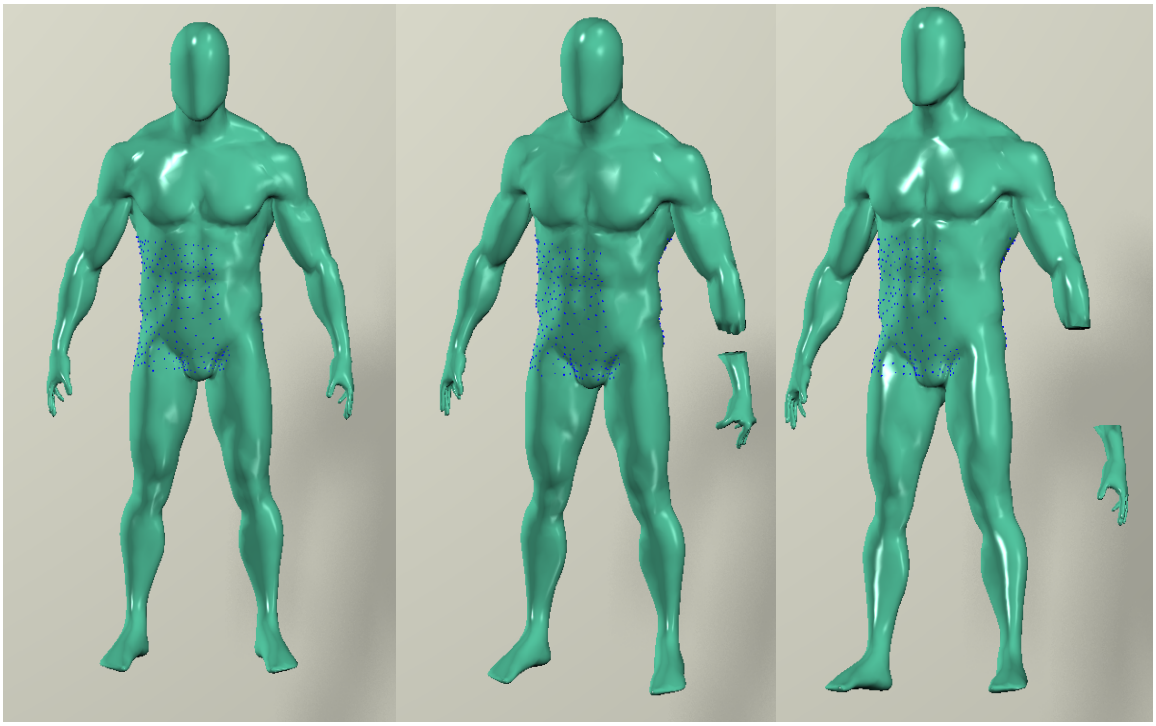


Figure 7.16: Illustration of original mesh (left) and one hand of the mesh is cut and detached from the body. The whole simulation runs using Galerkin multigrid combined with graph-based Finite Element Method Is-XFEM (right).

For proper visualization, the intersection of the boundary mesh of both the simulation and visualization meshes and the cutting tool mesh is determined first. The intersection plane is then duplicated to generate both sides of the cut. Finally, the boundary of the visualization mesh is

re-meshed with these cut planes embedded into it. This ensures that we always have a closed boundary mesh.

Table 7.1: Resolution of object meshes.

Model	***Tetrahedra Number***
T-Rex	2.5k
Zombie	1.2k
Lioness	1.3k
Spider	1.1k
Elephant	1.0k
Toy Ninja	0.9k
Sphere	1.5k
Human Body	40.3k

7.4.4 Sculpting

Our framework is capable of rendering the effects of multiple sculpting operations on the same model. In Figure 7.1 (second image from left) cutting, deforming and wetting operations are performed on a Lioness model. In Figure 7.17, Figure 7.18 and Figure 7.19 we present an Elephant, a Lioness and a Spider model respectively sculpted by different volunteers. Similarly in Figure 7.1, the last two images on the right show an original and a sculpted Toy Ninja model. In Figure 7.20 an amateur volunteer sculpted a scary mask starting from a simple sphere model using Is-XFEM framework. A similar sphere model is sculpted using Galerkin multigrid method in Figure 7.21. As evident from both these figures, using Galerkin multigrid framework an artist can sculpt more complicated and expressive models. The number of tetrahedra in each model is reported in Table 7.1.

Our interactive framework can handle multiple sculpting operations being performed on an object mesh in real-time. At the end of the sculpting operations, both the volumetric tetrahedral mesh and visualization triangular mesh gets affected. Users can save, export or import any of these sculpted meshes, whether tetrahedral or triangular, for using them in the same or other

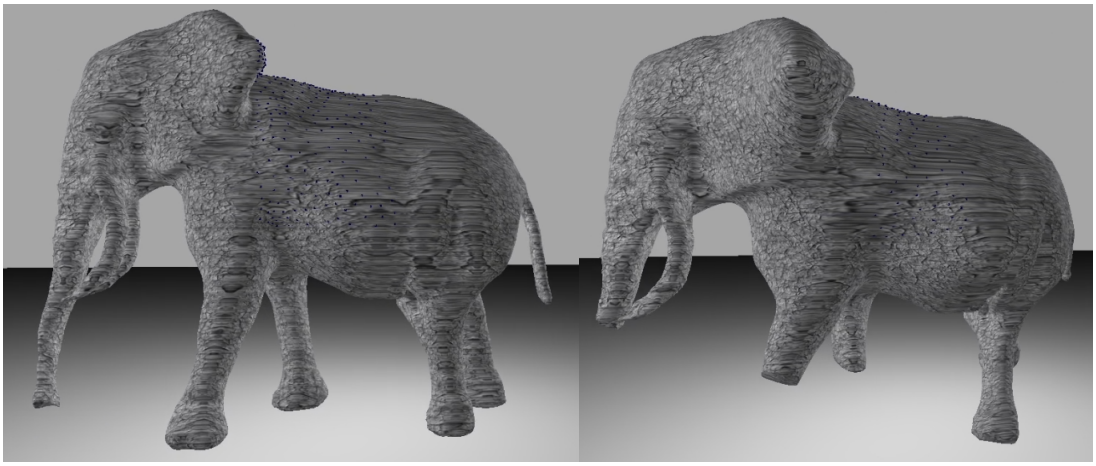


Figure 7.17: Original (left) and sculpted (right) Elephant model.

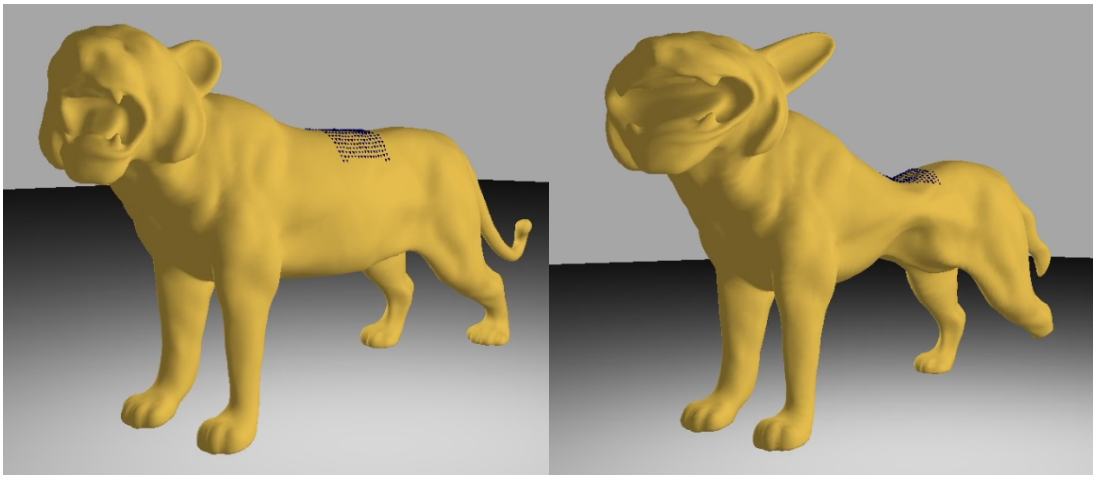


Figure 7.18: Original (left) and sculpted (right) Lioness model.

applications. Moreover, after sculpting, our method always generates a water-tight visualization mesh which is convenient for further use in different applications.

Figure 7.19: Original (left) and sculpted (right) Spider model.

In Figure 7.22, starting from a sphere mesh, another user sculpted a model of *Grogu* from television series *The Mandalorian*.

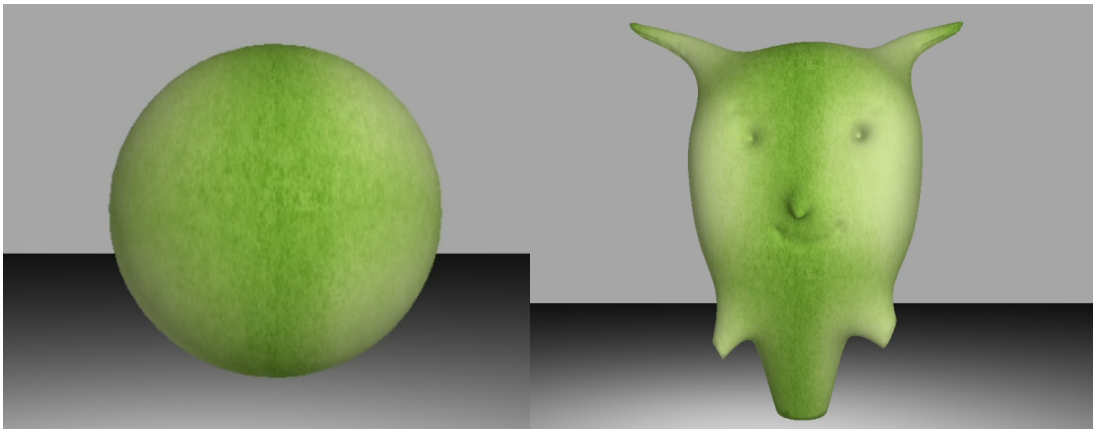


Figure 7.20: Original (left) and sculpted (right) Sphere model using Is-XFEM framework.

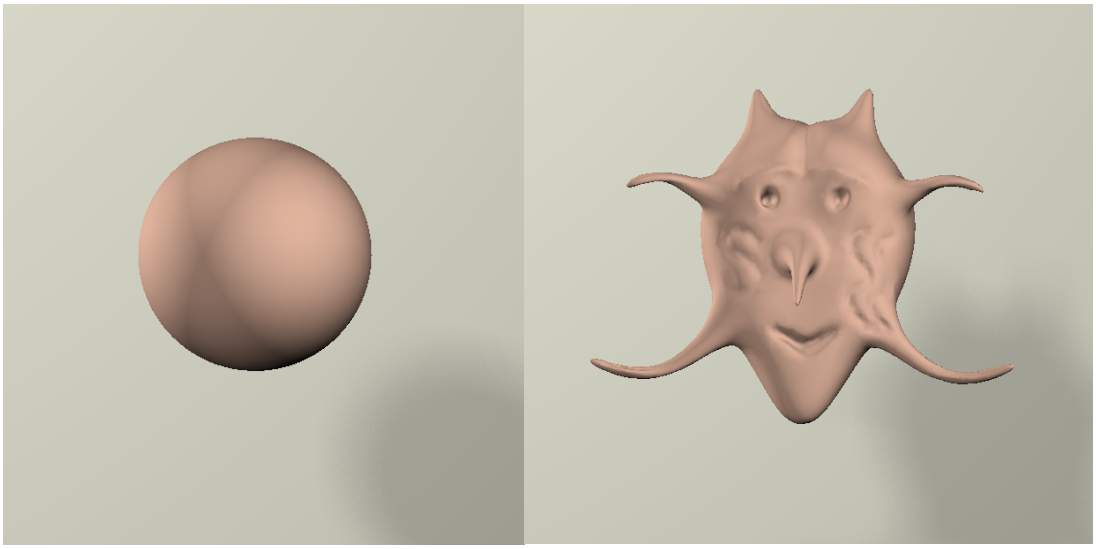


Figure 7.21: Original (left) and sculpted (right) Sphere model using Galerkin multigrid framework.

7.4.5 User Study

We perform a preliminary user study for all the sculpting operations of our framework, i.e., cutting of mesh along with deformation and wetting from [Mandal et al., 2022a]. Our user study consists of the following two experiments.

- **Haptics-Visual Feedback Study** to analyse the effect of haptics and visual feedback in virtual sculpting. We perform an ANOVA analysis for this study.

- **Double Stimulus Comparison Study** to determine how close the virtual sculpting expe-

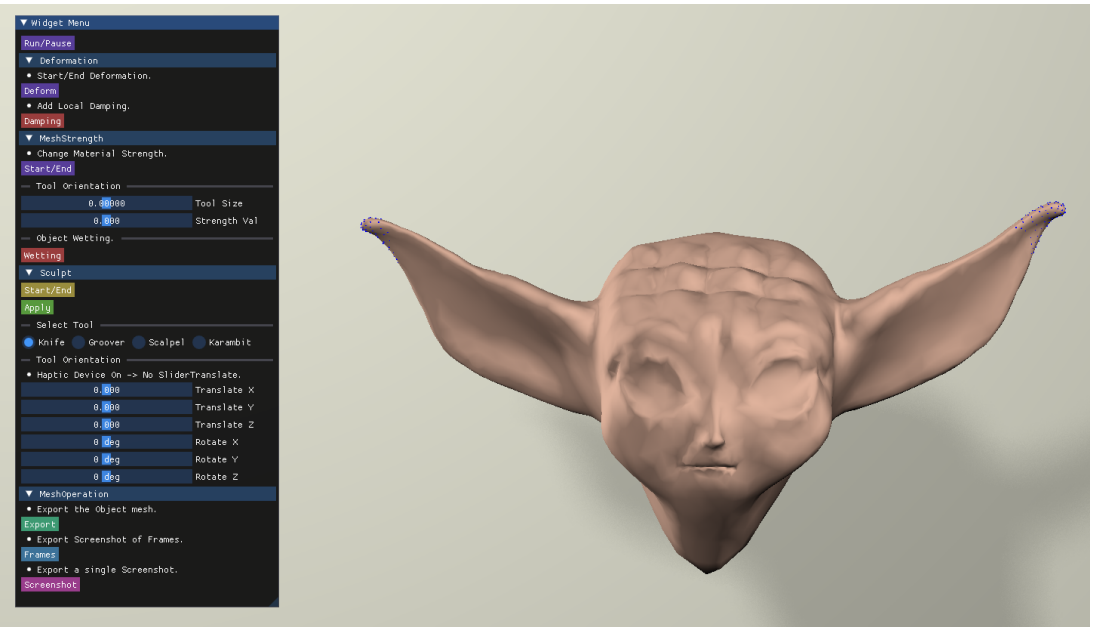


Figure 7.22: Sculpted a model of *Grogu* from television series *The Mandalorian*.

rience is compared to real-world sculpting.

7.4.5.1 Study Subjects

Our user study is conducted with 20 subjects with 16 males and 4 females. All the subjects are between 22 and 40 years old. All the participants confirm that they are in sound health both physically and mentally at the time of the experiment. None of the participants has any experience using a haptic setup before.

7.4.5.2 Experimental Setup

Our experimental setup is shown in Figure 7.23. A user sculpting a virtual Lioness model is shown in the figure. Figure 7.25 visually compares the result of deforming a virtual clay sphere (3D volumetric mesh) with a real clay sphere. Similarly, Figure 7.24 visually compares the result of cutting a virtual clay cylinder (3D volumetric mesh) with a real clay cylinder. Similar results are depicted in Figure 7.26.

Figure 7.23: Set up for our experiment.

7.4.5.3 Haptics-Visual Feedback Study

The Analysis of Variance (ANOVA) [Fisher, 1925] is a commonly used tool to analyze whether the differences between groups of data are statistically significant. In our work, we use ANOVA to determine the relevance and usefulness of rendering haptics and visual feedback during virtual sculpting. First, we form three groups consisting of six subjects in each group. The users of each group are asked to use the sculpting tools available in our framework on a sphere mesh and rate their immersive experience of sculpting on a scale of 1 (very poor) to 5 (very good) for each case. Each group is assigned to perform sculpting under one of the following strategies.

- **Strategy 1 - Visual On & Haptics On**: Subjects get both visual and haptic feedback.
- **Strategy 2 - Visual On & Haptics Off**: Subjects get visual feedback but no haptic feedback.
- **Strategy 3 - Visual Off & Haptics On**: Subjects get haptic feedback but no visual feedback.

Note that "without visual feedback" does not denote a complete absence of the virtual scene. It means that the effect of the sculpting operations (e.g., deformation, opening up of the cut plane) is not rendered on the high-resolution visualization mesh. All the sculpting operations are performed only on the outer cage mesh.

Table 7.2: p-value for ANOVA study.

Compared strategies	***p-value***
1 vs 2	0.00065
1 vs 3	0.00001

The null hypothesis in ANOVA is that all groups are random samples from the same population, which in our work means that all the sculpting strategies are equally effective. Thus, any observed difference between them is due to random noise. Now, the p-value is the probability of obtaining results at least as extreme as the observed results of a statistical hypothesis test, assuming that the null hypothesis is correct. So, if the p-value is below a certain threshold, the null hypothesis is rejected. In our work, we use p-value of 0.05, which is a widely accepted choice. The p-value (see Table 7.2) for Strategy 1 vs Strategy 2 is $0.00065 < 0.05$, which denotes a significant difference between the two strategies rejecting the null hypothesis. However, as reported in the top half of Table 7.3, the higher mean and median score for Strategy 1 compared to Strategy 2 denotes that user experience for virtual sculpting improves when haptic feedback is on. Similarly, the p-value for Strategy 1 vs Strategy 3 is $0.00001 < 0.05$, which again rejects the null hypothesis. The much higher mean and median score for Strategy 1 compared to Strategy 3 proves that visual feedback is paramount for a faithful user experience.

Table 7.3: Mean and standard deviation of user feedback.

Parameter	***Mean***	***Median***	***Std***
Visual On, Haptics On	4.71	4.65	0.23
Visual On, Haptics Off	4.27	4.35	0.26
Visual Off, Haptics On	2.79	2.85	0.43
Realism	4.68	4.70	0.34
Visual-haptic sync	4.85	5.00	0.22
Physical consistency	4.51	4.50	0.29

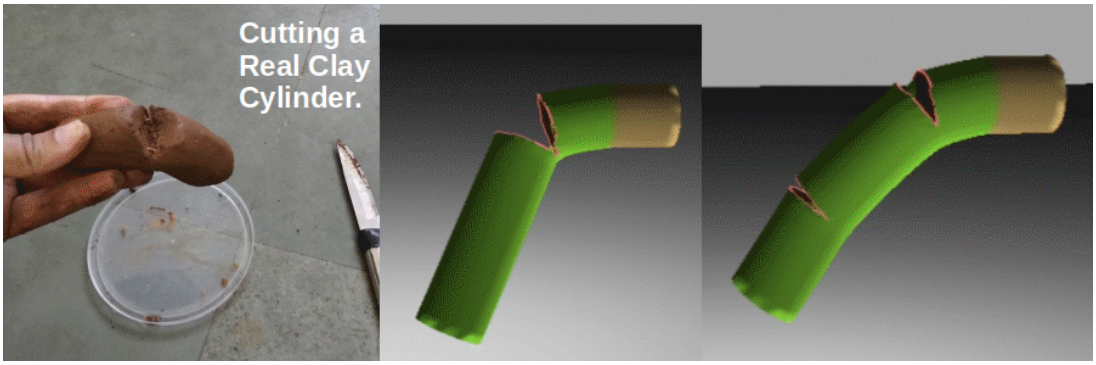


Figure 7.24: The three images (from left to right) show cutting a cylinder made of real clay, simulating single and multiple cuts on a virtual clay cylinder.

Therefore if we compare the various strategies then based on the results of the above study we can make the following observations.

Strategy 1 vs Strategy 2: According to our data, users rate their virtual sculpting experience most favourably for Strategy 1. It reveals that if the visual feedback remains the same, turning on the haptic feedback while sculpting improves user experience.

Strategy 1 vs Strategy 3: As the results show Strategy 3 performs very poorly compared to Strategy 1 which asserts the utmost importance of appropriate visual feedback for virtual sculpting. This is not surprising given that in real life also, visual cues and motions are most dominant among all the six sensory senses used for perception.

7.4.5.4 Double Stimulus Comparison Study

We perform a double stimulus comparison [Union, 2013] study to compare our framework with real-world sculpting experience. For this study, we use the following parameters.

- **Realism**: The participants are asked to evaluate how realistic their virtual sculpting experience is compared to real-world sculpting.
- **Visual-haptic synchronization**: The users are asked to report if they experienced any delay between visual change and haptic force feedback while using our framework.
- **Physical consistency**: Finally, the users are asked to rate how consistent is the visual simulation of our framework with the real world.

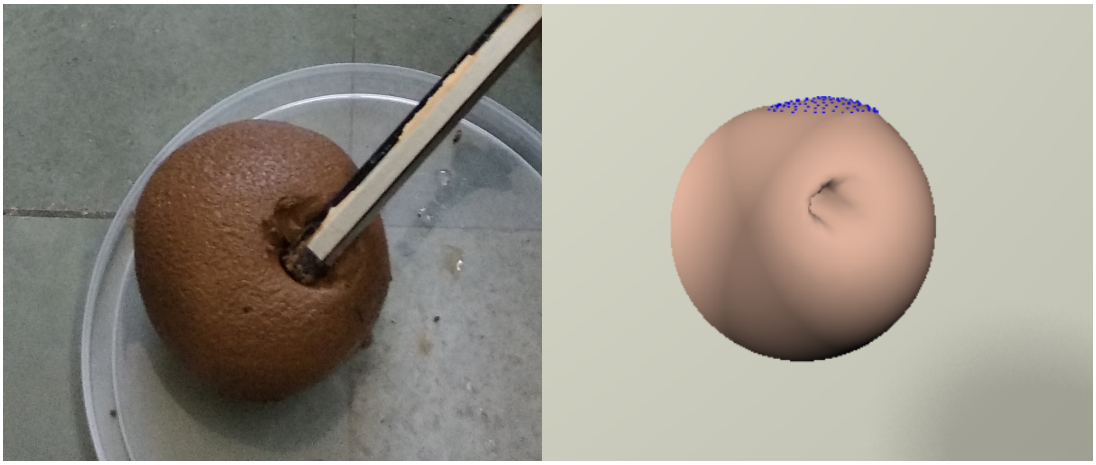


Figure 7.25: User deforming a cylinder made of real clay (left) and a virtual clay (right).

In the study, each subject experiences two stimuli one after the other.

- **First stimulus**: First the subjects are asked to mould a ball of clay with a small stick (see Figure 7.25(left)) and cut a clay cylinder with a knife (see Figure 7.24(left)).
- **Second stimulus**: The subjects are then asked to perform the same two operations using the virtual sculpting tools available in our framework (see Figure 7.25(right) and Figure 7.24(middle, right)).

Then the subjects are asked to rate their experience on a scale of 1 (very poor) to 5 (very good) after finishing the experiment.

The mean and standard deviation of the scores of the user feedback opinions for our framework are listed in the bottom half of Table 7.3. As evident from the Table, the users rate positively for their virtual sculpting experience. Moreover, the low value of standard deviation denotes little difference of opinion among users.

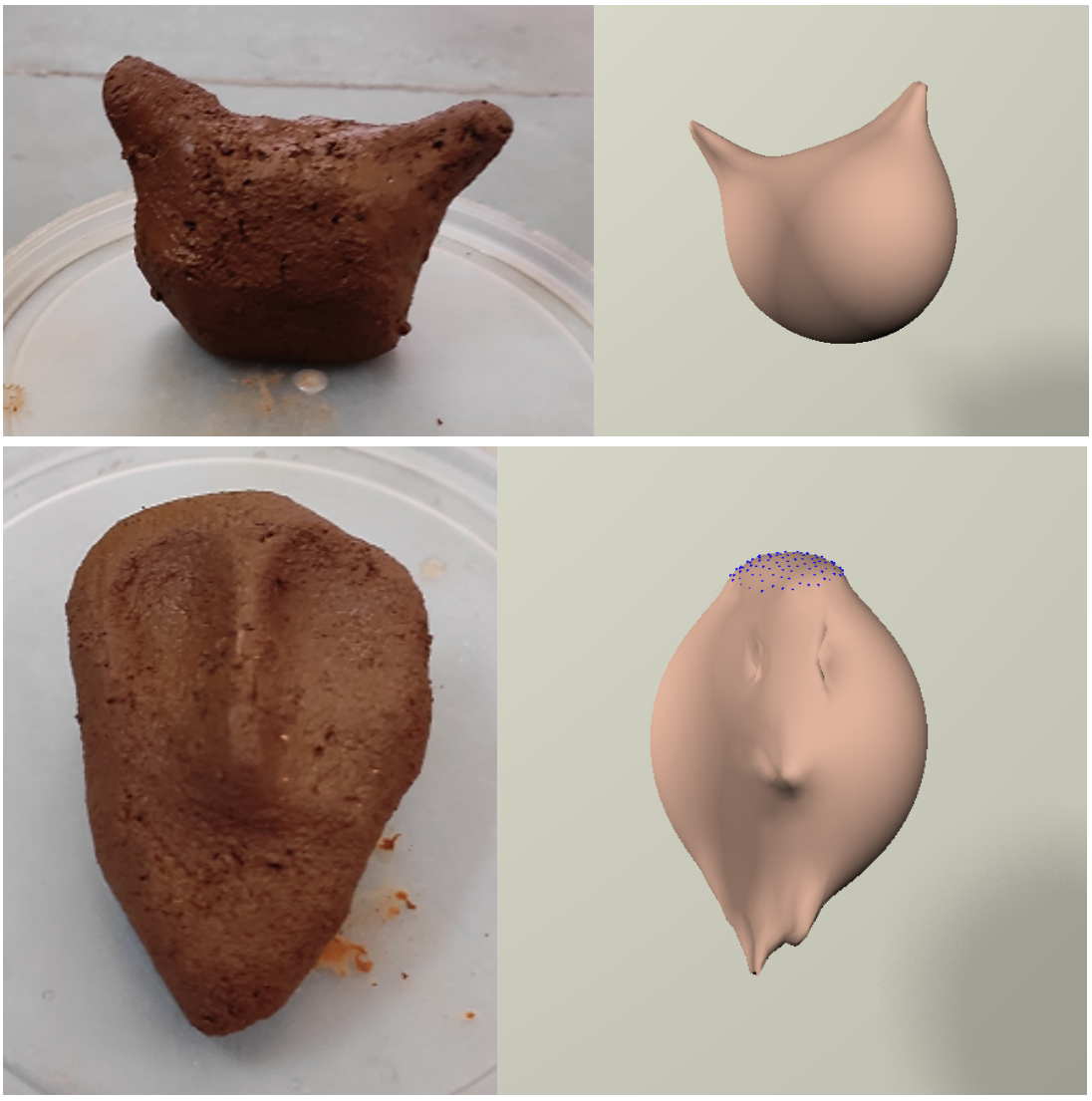


Figure 7.26: Shapes made by participants using a real clay ball (left) and a virtually simulated ball (right).

Chapter 8

Summary and Future Work

8.1 Summary

In this dissertation, we propose multiple methods to simulate complex cuts and fractures on an object mesh. We first build an Improved stable XFEM to simulate complex cuts on deformable objects at an interactive rate. We validate our framework with a preliminary user study.

The limitations of Is-XFEM motivate us to develop a novel graph-based FEM for fracture simulation in both brittle and ductile materials. We derive theoretical proof to show that our method extends to non-linear hyper-elastic strain energy densities. The high stability, speed and robustness of our method are illustrated and established via multiple dynamic experiments with different materials. We evaluate the appeal and realism of our fracture simulations through results on objects made of a wide variety of materials. Compared with benchmark fracture experiments, we validate the correctness and accuracy of our method. We also compare with existing XFEM simulation methods, providing quantitative and visual evidence about the better

performance of our method.

We extend our graph-based FEM to introduce a novel probabilistic damage mechanics formulation to incorporate the uncertainties during fracture, caused by material impurities. We leverage this formulation to create a framework that allows an artist-controlled design of fracture. Moreover, we prove using a Markov random process abstraction that even after the introduction of impurities, no sharp discontinuities or spurious divergences arise in the value of various node parameters of the FEM simulation mesh. We demonstrate the appeal and accuracy of this method for ductile and brittle fracture of various objects for anisotropic materials and materials with or without impurities. We also provide an alternate theoretical representation of damage as an edge variable using a smooth mapping from Euclidean to Riemannian manifolds, which better explains the edge-based damage captured by graph-based FEM.

Finally, we integrate our graph-based FEM with a Galerkin multigrid method to simulate complex cuts and fractures on a high-resolution mesh at an interactive rate. Galerkin multigrid method can simulate the deformation of a high-resolution mesh in real-time. Our graph-based FEM provides the cutting framework on top of it without any extra computational overhead.

8.2 Limitations

- Currently, in our fracture simulation algorithm, an edge of an element in the object mesh can split into a maximum of two parts, thus limiting the fracture of tetrahedral elements to a maximum of four parts.

- Even though our surface remeshing algorithm for graph-based FEM uses floating point operations for element splitting, we do not employ any safeguard to resolve rare errors that might occur due to floating point operations. Despite this, our algorithm is very stable, even when simulating highly complex fracture patterns.

8.3 Future Work

We first explain a few possible approaches to overcome the limitations. Next, we explain some interesting future directions of work.

8.3.1 Addressing the Limitations

- One important approach for future work is to incorporate a high number of fracture segments in a single element. Adaptive remeshing schemes [Pfaff et al., 2014] [Mercier-Aubin et al., 2022a] combined with graph-based FEM could be one possible direction.

- To remove the floating point errors, some possible solutions could be employing the methods from [Wang et al., 2015] [Wang et al., 2021] where authors resolve floating point errors by providing an appropriate error bound or by looking for increasingly tighter subintervals.

8.3.2 New Avenues of Work

- Exploring other areas related to fractures, e.g. fractures caused by fatigue which is a longer-term phenomenon that is crucial to the safety of many structures like dams and aircraft, may lead us to some fascinating results.

- Similarly, another exciting avenue of our work would be to simulate fracture with a learning-based approach, i.e., employ state-of-the-art deep learning techniques such as Graph Neural Network or Convolutional Neural Network to learn the physics behind fracture. On one hand, this may also enable us to easily render fracture for a large array of materials and on the other, relieve the artist from the burden of tuning the vast number of material parameters.

- Another fascinating future direction of our work would be to investigate wave propagation in fractured media. These algorithms may have huge potential applications in detecting

micro-cracks inside any high-value assets in an inexpensive way.

- Fracture occurs due to shockwave as well. A large blast or high pitch sound is capable of shattering glass or damaging the membrane of our eardrums. We would like to explore this kind of wave-induced fractures in future.
- In continuation with the above point, devising the mathematical model for sound synthesis due to fracture simulation may be another potential direction of future work.
- While simulating fracture, we often neglect the contribution of strain energy generated due to surface curvature. However, this energy, dominant in elastic polymers, plays a vital role in crack initiation and propagation. This could also be a potential avenue for future work.

Appendix A

Derivation of Internal Elastic Forces along the Edges

As introduced in Equation 4.1, the internal elastic force for graph-based FEM can be written as

$$\mathbf{f}_{e_i}^{int} = V_e \frac{\partial \mathbf{\Psi}^e}{\partial \mathbf{r}_i^e} = 2V_e \sum_{\substack{j=1 \\ j \neq i}}^{n_e} \frac{\partial \mathbf{\Psi}^e}{\partial \left(d_{ij}^e\right)^2} d_{ij}^e \hat{\mathbf{d}}_{ij}^e \tag{A.1}$$

where $\mathbf{r}_i^e$ is the world position vector for node i of element Δ_e. The parameter $d_{ij}^e = ||\mathbf{r}_i^e - \mathbf{r}_j^e||$ denotes the distance between the i^{th} and j^{th} nodes of Δ_e and $\hat{\mathbf{d}}_{ij}^e$ is unit vector along d_{ij}^e. Moreover, V_e and n_e represent the volume and the number of nodes of Δ_e respectively. Here we give the full derivation of the above equation.

$$\mathbf{f}_{e_i}^{int} = V_e \frac{\partial \mathbf{\Psi}^e}{\partial \mathbf{r}_i^e}$$

$$= V_e \sum_{\substack{j=1 \\ j \neq i}}^{n_e} \frac{\partial \boldsymbol{\Psi}^e}{\partial \left(d_{ij}^e\right)^2} \frac{\partial \left(d_{ij}^e\right)^2}{\partial \mathbf{r}_i^e} \quad \text{(Using chain rule of differentiation)}$$

$$= V_e \sum_{\substack{j=1 \\ j \neq i}}^{n_e} \frac{\partial \boldsymbol{\Psi}^e}{\partial \left(d_{ij}^e\right)^2} \frac{\partial d_{ij}^e}{\partial \mathbf{r}_i^e} 2d_{ij}^e$$

$$= V_e \sum_{\substack{j=1 \\ j \neq i}}^{n_e} \frac{\partial \boldsymbol{\Psi}^e}{\partial \left(d_{ij}^e\right)^2} \frac{\partial \sqrt{\left(\mathbf{r}_i^e - \mathbf{r}_j^e\right)^T \left(\mathbf{r}_i^e - \mathbf{r}_j^e\right)}}{\partial \mathbf{r}_i^e} 2d_{ij}^e$$

$$= V_e \sum_{\substack{j=1 \\ j \neq i}}^{n_e} \frac{\partial \boldsymbol{\Psi}^e}{\partial \left(d_{ij}^e\right)^2} \frac{1}{2\sqrt{\left(\mathbf{r}_i^e - \mathbf{r}_j^e\right)^T \left(\mathbf{r}_i^e - \mathbf{r}_j^e\right)}} \frac{\partial \left[\left(\mathbf{r}_i^e - \mathbf{r}_j^e\right)^T \left(\mathbf{r}_i^e - \mathbf{r}_j^e\right)\right]}{\partial \mathbf{r}_i^e} 2d_{ij}^e$$

$$= V_e \sum_{\substack{j=1 \\ j \neq i}}^{n_e} \frac{\partial \boldsymbol{\Psi}^e}{\partial \left(d_{ij}^e\right)^2} \frac{1}{\cancel{2d_{ij}^e}} \frac{\partial \left[(\mathbf{r}_i^e)^T \mathbf{r}_i^e - (\mathbf{r}_i^e)^T \mathbf{r}_j^e - (\mathbf{r}_j^e)^T \mathbf{r}_i^e + (\mathbf{r}_j^e)^T \mathbf{r}_j^e\right]}{\partial \mathbf{r}_i^e} \cancel{2d_{ij}^e}$$

$$= V_e \sum_{\substack{j=1 \\ j \neq i}}^{n_e} \frac{\partial \boldsymbol{\Psi}^e}{\partial \left(d_{ij}^e\right)^2} \frac{\partial \left[(\mathbf{r}_i^e)^T \mathbf{r}_i^e - 2(\mathbf{r}_i^e)^T \mathbf{r}_j^e + (\mathbf{r}_j^e)^T \mathbf{r}_j^e\right]}{\partial \mathbf{r}_i^e} \quad \text{(As } (\mathbf{r}_i^e)^T \mathbf{r}_j^e = (\mathbf{r}_j^e)^T \mathbf{r}_i^e \text{, both being scalar values)}$$

$$= V_e \sum_{\substack{j=1 \\ j \neq i}}^{n_e} \frac{\partial \boldsymbol{\Psi}^e}{\partial \left(d_{ij}^e\right)^2} 2\left(\mathbf{r}_i^e - \mathbf{r}_j^e\right)$$

$$= 2V_e \sum_{\substack{j=1 \\ j \neq i}}^{n_e} \frac{\partial \boldsymbol{\Psi}^e}{\partial \left(d_{ij}^e\right)^2} d_{ij}^e \hat{\mathbf{d}}_{ij}^e \tag{A.2}$$

Appendix B

Full expressions for A_1 and A_2 from Equation 4.23

B.1 Full expression of $\mathbf{A}_1$

$$\mathbf{A}_1 = \left[\frac{\partial \mathbf{l}_d^e}{\partial \mathbf{u}_e}\right]^T = \begin{bmatrix} \frac{\partial l_{12}^e}{\partial u_{1x}^e} & \frac{\partial l_{13}^e}{\partial u_{1x}^e} & \frac{\partial l_{14}^e}{\partial u_{1x}^e} & \frac{\partial l_{23}^e}{\partial u_{1x}^e} & \frac{\partial l_{24}^e}{\partial u_{1x}^e} & \frac{\partial l_{34}^e}{\partial u_{1x}^e} \\ \frac{\partial l_{12}^e}{\partial u_{1y}^e} & \frac{\partial l_{13}^e}{\partial u_{1y}^e} & \frac{\partial l_{14}^e}{\partial u_{1y}^e} & \frac{\partial l_{23}^e}{\partial u_{1y}^e} & \frac{\partial l_{24}^e}{\partial u_{1y}^e} & \frac{\partial l_{34}^e}{\partial u_{1y}^e} \\ \frac{\partial l_{12}^e}{\partial u_{1z}^e} & \frac{\partial l_{13}^e}{\partial u_{1z}^e} & \frac{\partial l_{14}^e}{\partial u_{1z}^e} & \frac{\partial l_{23}^e}{\partial u_{1z}^e} & \frac{\partial l_{24}^e}{\partial u_{1z}^e} & \frac{\partial l_{34}^e}{\partial u_{1z}^e} \\ \frac{\partial l_{12}^e}{\partial u_{2x}^e} & \frac{\partial l_{13}^e}{\partial u_{2x}^e} & \frac{\partial l_{14}^e}{\partial u_{2x}^e} & \frac{\partial l_{23}^e}{\partial u_{2x}^e} & \frac{\partial l_{24}^e}{\partial u_{2x}^e} & \frac{\partial l_{34}^e}{\partial u_{2x}^e} \\ \frac{\partial l_{12}^e}{\partial u_{2y}^e} & \frac{\partial l_{13}^e}{\partial u_{2y}^e} & \frac{\partial l_{14}^e}{\partial u_{2y}^e} & \frac{\partial l_{23}^e}{\partial u_{2y}^e} & \frac{\partial l_{24}^e}{\partial u_{2y}^e} & \frac{\partial l_{34}^e}{\partial u_{2y}^e} \\ \frac{\partial l_{12}^e}{\partial u_{2z}^e} & \frac{\partial l_{13}^e}{\partial u_{2z}^e} & \frac{\partial l_{14}^e}{\partial u_{2z}^e} & \frac{\partial l_{23}^e}{\partial u_{2z}^e} & \frac{\partial l_{24}^e}{\partial u_{2z}^e} & \frac{\partial l_{34}^e}{\partial u_{2z}^e} \\ \frac{\partial l_{12}^e}{\partial u_{3x}^e} & \frac{\partial l_{13}^e}{\partial u_{3x}^e} & \frac{\partial l_{14}^e}{\partial u_{3x}^e} & \frac{\partial l_{23}^e}{\partial u_{3x}^e} & \frac{\partial l_{24}^e}{\partial u_{3x}^e} & \frac{\partial l_{34}^e}{\partial u_{3x}^e} \\ \frac{\partial l_{12}^e}{\partial u_{3y}^e} & \frac{\partial l_{13}^e}{\partial u_{3y}^e} & \frac{\partial l_{14}^e}{\partial u_{3y}^e} & \frac{\partial l_{23}^e}{\partial u_{3y}^e} & \frac{\partial l_{24}^e}{\partial u_{3y}^e} & \frac{\partial l_{34}^e}{\partial u_{3y}^e} \\ \frac{\partial l_{12}^e}{\partial u_{3z}^e} & \frac{\partial l_{13}^e}{\partial u_{3z}^e} & \frac{\partial l_{14}^e}{\partial u_{3z}^e} & \frac{\partial l_{23}^e}{\partial u_{3z}^e} & \frac{\partial l_{24}^e}{\partial u_{3z}^e} & \frac{\partial l_{34}^e}{\partial u_{3z}^e} \\ \frac{\partial l_{12}^e}{\partial u_{4x}^e} & \frac{\partial l_{13}^e}{\partial u_{4x}^e} & \frac{\partial l_{14}^e}{\partial u_{4x}^e} & \frac{\partial l_{23}^e}{\partial u_{4x}^e} & \frac{\partial l_{24}^e}{\partial u_{4x}^e} & \frac{\partial l_{34}^e}{\partial u_{4x}^e} \\ \frac{\partial l_{12}^e}{\partial u_{4y}^e} & \frac{\partial l_{13}^e}{\partial u_{4y}^e} & \frac{\partial l_{14}^e}{\partial u_{4y}^e} & \frac{\partial l_{23}^e}{\partial u_{4y}^e} & \frac{\partial l_{24}^e}{\partial u_{4y}^e} & \frac{\partial l_{34}^e}{\partial u_{4y}^e} \\ \frac{\partial l_{12}^e}{\partial u_{4z}^e} & \frac{\partial l_{13}^e}{\partial u_{4z}^e} & \frac{\partial l_{14}^e}{\partial u_{4z}^e} & \frac{\partial l_{23}^e}{\partial u_{4z}^e} & \frac{\partial l_{24}^e}{\partial u_{4z}^e} & \frac{\partial l_{34}^e}{\partial u_{4z}^e} \end{bmatrix}$$

$$= \begin{bmatrix} \cos\phi_x^{12} & \cos\phi_x^{13} & \cos\phi_x^{14} & 0 & 0 & 0 \\ \cos\phi_y^{12} & \cos\phi_y^{13} & \cos\phi_y^{14} & 0 & 0 & 0 \\ \cos\phi_z^{12} & \cos\phi_z^{13} & \cos\phi_z^{14} & 0 & 0 & 0 \\ -\cos\phi_x^{12} & 0 & 0 & \cos\phi_x^{23} & \cos\phi_x^{24} & 0 \\ -\cos\phi_y^{12} & 0 & 0 & \cos\phi_y^{23} & \cos\phi_y^{24} & 0 \\ -\cos\phi_z^{12} & 0 & 0 & \cos\phi_z^{23} & \cos\phi_z^{24} & 0 \\ 0 & -\cos\phi_x^{13} & 0 & -\cos\phi_x^{23} & 0 & \cos\phi_x^{34} \\ 0 & -\cos\phi_y^{13} & 0 & -\cos\phi_y^{23} & 0 & \cos\phi_y^{34} \\ 0 & -\cos\phi_z^{13} & 0 & -\cos\phi_z^{23} & 0 & \cos\phi_z^{34} \\ 0 & 0 & -\cos\phi_x^{14} & 0 & -\cos\phi_x^{24} & -\cos\phi_x^{34} \\ 0 & 0 & -\cos\phi_y^{14} & 0 & -\cos\phi_y^{24} & -\cos\phi_y^{34} \\ 0 & 0 & -\cos\phi_z^{14} & 0 & -\cos\phi_z^{24} & -\cos\phi_z^{34} \end{bmatrix} \tag{B.1}$$

where $\cos\phi_\alpha^{ij}$ denotes angle made by the edge formed by node i and j with $\alpha \in \{x, y, z\}$ axis.

B.2 Full expression of $\mathbf{A}_2$

$$
\begin{aligned}
\mathbf{A}_2 &= \left[\frac{\partial \boldsymbol{\varepsilon}_c^e}{\partial \mathbf{l}_d^e}\right]^T = \left[\frac{\partial \mathbf{T}^{-1}\boldsymbol{\varepsilon}_d^e}{\partial \mathbf{l}_d^e}\right]^T = \left[\frac{\partial \boldsymbol{\varepsilon}_d^e}{\partial \mathbf{l}_d^e}\right]^T \mathbf{T}^{-T} \\
&= \begin{bmatrix} \frac{\partial \varepsilon_{12}^e}{\partial \mathbf{l}_d^e} & \frac{\partial \varepsilon_{13}^e}{\partial \mathbf{l}_d^e} & \frac{\partial \varepsilon_{14}^e}{\partial \mathbf{l}_d^e} & \frac{\partial \varepsilon_{23}^e}{\partial \mathbf{l}_d^e} & \frac{\partial \varepsilon_{24}^e}{\partial \mathbf{l}_d^e} & \frac{\partial \varepsilon_{34}^e}{\partial \mathbf{l}_d^e} \end{bmatrix}^T \mathbf{T}^{-T} \\
&= \begin{bmatrix} \frac{1}{L_{12}^e} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{L_{13}^e} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{L_{14}^e} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{L_{23}^e} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{L_{24}^e} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{L_{34}^e} \end{bmatrix} \begin{bmatrix} \frac{\partial l_{12}^e}{\partial l_{12}^e} & \frac{\partial l_{12}^e}{\partial l_{13}^e} & \frac{\partial l_{12}^e}{\partial l_{14}^e} & \frac{\partial l_{12}^e}{\partial l_{23}^e} & \frac{\partial l_{12}^e}{\partial l_{24}^e} & \frac{\partial l_{12}^e}{\partial l_{34}^e} \\ \frac{\partial l_{13}^e}{\partial l_{12}^e} & \frac{\partial l_{13}^e}{\partial l_{13}^e} & \frac{\partial l_{13}^e}{\partial l_{14}^e} & \frac{\partial l_{13}^e}{\partial l_{23}^e} & \frac{\partial l_{13}^e}{\partial l_{24}^e} & \frac{\partial l_{13}^e}{\partial l_{34}^e} \\ \frac{\partial l_{14}^e}{\partial l_{12}^e} & \frac{\partial l_{14}^e}{\partial l_{13}^e} & \frac{\partial l_{14}^e}{\partial l_{14}^e} & \frac{\partial l_{14}^e}{\partial l_{23}^e} & \frac{\partial l_{14}^e}{\partial l_{24}^e} & \frac{\partial l_{14}^e}{\partial l_{34}^e} \\ \frac{\partial l_{23}^e}{\partial l_{12}^e} & \frac{\partial l_{23}^e}{\partial l_{13}^e} & \frac{\partial l_{23}^e}{\partial l_{14}^e} & \frac{\partial l_{23}^e}{\partial l_{23}^e} & \frac{\partial l_{23}^e}{\partial l_{24}^e} & \frac{\partial l_{23}^e}{\partial l_{34}^e} \\ \frac{\partial l_{24}^e}{\partial l_{12}^e} & \frac{\partial l_{24}^e}{\partial l_{13}^e} & \frac{\partial l_{24}^e}{\partial l_{14}^e} & \frac{\partial l_{24}^e}{\partial l_{23}^e} & \frac{\partial l_{24}^e}{\partial l_{24}^e} & \frac{\partial l_{24}^e}{\partial l_{34}^e} \\ \frac{\partial l_{34}^e}{\partial l_{12}^e} & \frac{\partial l_{34}^e}{\partial l_{13}^e} & \frac{\partial l_{34}^e}{\partial l_{14}^e} & \frac{\partial l_{34}^e}{\partial l_{23}^e} & \frac{\partial l_{34}^e}{\partial l_{24}^e} & \frac{\partial l_{34}^e}{\partial l_{34}^e} \end{bmatrix}^T \mathbf{T}^{-T} \\
&= \begin{bmatrix} \frac{1}{L_{12}^e} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{L_{13}^e} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{L_{14}^e} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{L_{23}^e} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{L_{24}^e} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{L_{34}^e} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{T}^{-T} \\
&= \begin{bmatrix} \frac{1}{L_{12}^e} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{L_{13}^e} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{L_{14}^e} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{L_{23}^e} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{L_{24}^e} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{L_{34}^e} \end{bmatrix} \mathbf{T}^{-T}
\end{aligned}
\tag{B.2}
$$

This page was intentionally left blank.

Appendix C

Failed Approaches For Reformulating Energy Density for Non-linear Graph-based FEM

C.1 First Approach - Optimization based

Plugging in the optimized strain tensor $\hat{\mathbf{C}} = \mathscr{T} \cdot [\boldsymbol{Q} \otimes \boldsymbol{Q}]^{\dagger} \cdot \boldsymbol{Q}$ from Theorem E.1 to the set of invariants and subsequently calculating elastic strain energy density using these updated invariants, may seem a promising direction for internal force and tangent stiffness matrix calculation after fracture. However, the major problem of the optimization procedure is that it produces odd values of $\hat{\mathbf{C}}$ which are not practically consistent with the fractured state of the element. These odd values in turn make the movement of disjoint fractured segments erratic and render the simulation unstable. In Figure C.1 we show the results for this approach on a 2D bar on

which external force applied is applied at one end while keeping the other end hinged. Thus this approach does not work. Moreover, we notice that performing the optimization operation in every time step is extremely time consuming.

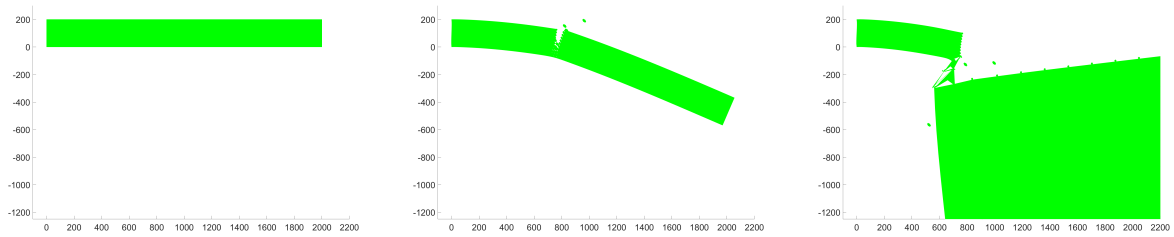


Figure C.1: First failed approach for simulating fracture on a 2D bar.

C.2 Second Approach - Coordinate transform

While using non-linear hyper-elastic strain energy density, the internal force vector, $\mathbf{f}_e^{int}$ and tangent stiffness matrix, $\mathbf{k}_e$ can be defined as

$$\begin{aligned}\mathbf{f}_e^{int} &= \int_{\Delta_e} \frac{\partial \boldsymbol{\Psi}^e}{\partial \mathbf{u}} d\xi = \int_{\Delta_e} \left[\frac{\partial \boldsymbol{\Psi}^e}{\partial \mathbf{F}}\right] \left[\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right] d\xi \\ &= V_e \mathbf{P}(\mathbf{F}) \left[\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right]\end{aligned} \tag{C.1}$$

$$\begin{aligned}\mathbf{k}_e &= \int_{\Delta_e} \frac{\partial \mathbf{f}_e^{int}}{\partial \mathbf{u}} d\xi = \int_{\Delta_e} \left[\frac{\partial \mathbf{f}_e^{int}}{\partial \mathbf{F}}\right] \left[\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right] d\xi \\ &= V_e \left[\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right]^T \left[\frac{\partial \mathbf{P}}{\partial \mathbf{F}}\right] \left[\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right]\end{aligned} \tag{C.2}$$

Now one intuitive way to incorporate fractured stress value to Equation C.1 is to transform the Cartesian component of Piola-Kirchhoff stress tensor, $\mathbf{P}(\mathbf{F})$ to the normal stress tensor along the edges, then apply zero stress values to the edges that are broken using fracture threshold criteria and finally transform them back to the Cartesian space again.

$$\left[\boldsymbol{\sigma}^e_{c_{frac}}\right]^T = \mathbf{T}^{-1} \boldsymbol{\zeta} \mathbf{T} [\boldsymbol{\sigma}^e_c]^T \tag{C.3}$$

Finally, plugging in the values of $\boldsymbol{\sigma}^e_{c_{frac}}$ in Equation C.1 it may be assumed that we get back the fractured internal forces. Similarly, the same procedure can be followed for tangent stiffness matrix, $\mathbf{k}_e$ in Equation C.2.

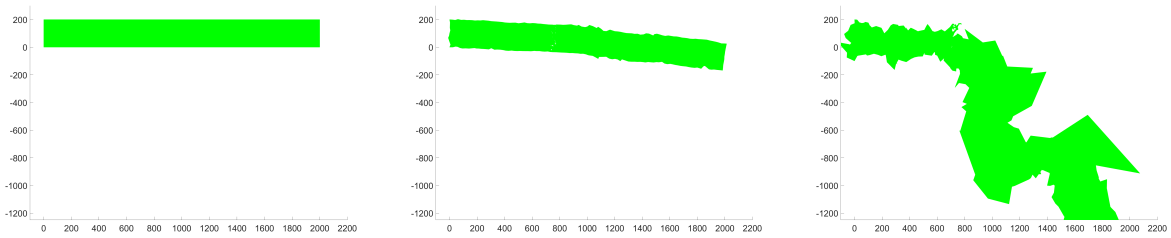


Figure C.2: Second failed approach for simulating fracture on a 2D bar.

However, following this line of thought does not work and it blows up the simulation as shown in Figure C.2. The reason for this can be seen from the derivation of internal force and tangent stiffness matrix for the linear case of graph-based FEM. As shown in Equation 4.23, the matrix $\mathbf{B}$ is denoted by $\mathbf{A}_2^T\mathbf{A}_1^T$ in linear case. However, for non-linear case in Equation C.1 and Equation C.2 the matrix $\mathbf{B}$ is $\dfrac{\partial \mathbf{F}}{\partial \mathbf{u}}$ which is not same as linear case. Thus if we follow this approach, we have to derive the matrix $\mathbf{B}$ from the fractured hyper-elastic strain energy density equation similar to the linear case. However, for non-linear case the hyper-elastic strain energy density is not a simple function of multiplication of stress and strain $\boldsymbol{\Psi}^e = \dfrac{1}{2}\boldsymbol{\sigma}_c^e \cdot \boldsymbol{\varepsilon}_c^e$ but rather depends on lower order invariants e.g., $\{I_C, II_C, III_C\}$ of Cauchy-Green deformation tensor $\mathbf{C}$. We do not find a suitable way of rewriting the invariants $\{I_C, II_C, III_C\}$ in a damaged state due to the problematic pseudo-inverse operation as discussed previously. Thus, we are not able to derive $\mathbf{B}$ and damaged stress $\boldsymbol{\sigma}^e_{c_{frac}}$ using edge-based criteria similar to the linear case. Moreover, note that in linear case the damaged internal elastic force $\mathbf{f}^{int}_{e_{frac}}$ (Equation 4.26) and tangent stiffness matrix $\mathbf{k}_{e_{frac}}$ (Equation 4.27) depend on the damage matrix $\boldsymbol{\zeta}$ quadratically. If we just plug in damaged stress $\boldsymbol{\sigma}^e_{c_{frac}}$ from Equation C.3 into Equation C.1 and Equation C.2, the dependence on $\boldsymbol{\zeta}$ will be only linear, which is incorrect.

This page was intentionally left blank.

Appendix D

Linearization of Hyperelastic Strain Energy Density

D.1 Saint–Venant Kirchhoff energy density

The expression for StVK energy density is as follows.

$$\mathbf{\Psi}_{stvk} = \frac{\lambda}{8}\left(I_C - 3\right)^2 + \frac{\mu}{4}\left(II_C - 2I_C + 3\right) \tag{D.1}$$

The energy expression can be rewritten [Sin et al., 2013] in terms of Lagrangian finite strain tensor $\mathbf{E} = 1/2\left(\mathbf{F}^T\mathbf{F} - \mathbf{I}\right)$ as

$$\mathbf{\Psi}_{stvk} = \frac{\lambda}{2}\left(\text{trace}\left(\mathbf{E}\right)\right)^2 + \mu\mathbf{E} : \mathbf{E} \tag{D.2}$$

The PK1 stress for the StVK energy density is

$$\mathbf{P}(\mathbf{F}) = 2\mu\mathbf{E} + \lambda\,\text{trace}\left(\mathbf{E}\right)\mathbf{I} \tag{D.3}$$

Replacing Lagrangian finite strain tensor, $\mathbf{E}$ with infinitesimal linearized strain tensor $\varepsilon = 1/2(\mathbf{F}+\mathbf{F}^T)-\mathbf{I}$ and PK1 stress with linearized stress, $\boldsymbol{\sigma}$, we get

$$\boldsymbol{\sigma} = 2\mu\boldsymbol{\varepsilon} + \lambda \text{trace}\,(\boldsymbol{\varepsilon})\,\mathbf{I} \tag{D.4}$$

Thus for StVK energy density $\mu = \mu_{\text{Lamé}}$ and $\lambda = \lambda_{\text{Lamé}}$.

D.2 Neo-Hookean energy density

The Neo-Hookean energy density can be expressed as

$$\boldsymbol{\Psi}_{neo} = \frac{\mu}{2}(I_C - 3) + \frac{\lambda}{2}(J-\alpha)^2 - \frac{\mu}{2}\log(I_C+1) \tag{D.5}$$

where $\alpha = 1 + \dfrac{\mu}{\lambda} - \dfrac{\mu}{4\lambda}$.

The PK1 stress for the Neo-Hookean energy density is

$$\mathbf{P}(\mathbf{F}) = \mu\left(1 - \frac{1}{I_C+1}\right)\mathbf{F} + \lambda\,(J-\alpha)\,\frac{\partial J}{\partial \mathbf{F}} \tag{D.6}$$

Like before replacing PK1 stress with linearized stress, $\boldsymbol{\sigma}$,

$$\begin{aligned}
\boldsymbol{\sigma} &= \mu\left(1 - \frac{1}{I_C+1}\right)\mathbf{F} + \lambda\,(J-\alpha)\,\frac{\partial J}{\partial \mathbf{F}} \\
&= \mu\left(1 - \frac{1}{I_C+1}\right)\mathbf{F} + \lambda\left(J - 1 - \frac{\mu}{\lambda} + \frac{\mu}{4\lambda}\right)J\mathbf{F}^{-T}, \quad \text{using } \frac{\partial J}{\partial \mathbf{F}} = J\mathbf{F}^{-T} \\
&= \mu\mathbf{F} - \frac{\mu\mathbf{F}}{I_C+1} + \left(\lambda\,\text{trace}(\boldsymbol{\varepsilon}) - \frac{3\mu}{4}\right)(1+\text{trace}(\boldsymbol{\varepsilon}))\,\mathbf{F}^{-T}, \quad \text{using } J \approx 1+\text{trace}(\boldsymbol{\varepsilon}) \\
&= \mu\left(\mathbf{F} - \frac{3}{4}\mathbf{F}^{-T}\right) - \frac{\mu\mathbf{F}}{I_C+1} + \left(\lambda - \frac{3\mu}{4}\right)\text{trace}(\boldsymbol{\varepsilon})\mathbf{F}^{-T}, \quad \text{ignoring } \text{trace}(\boldsymbol{\varepsilon})^2 \\
&= \mu\left(\mathbf{F}\mathbf{F}^T - \frac{3}{4}\mathbf{I}\right)\mathbf{F}^{-T} - \frac{\mu\mathbf{F}}{I_C+1} + \left(\lambda - \frac{3\mu}{4}\right)\text{trace}(\boldsymbol{\varepsilon})\mathbf{F}^{-T} \\
&= \mu\left(\mathbf{I} + 2\boldsymbol{\varepsilon} - \frac{3}{4}\mathbf{I}\right)\mathbf{F}^{-T} - \frac{\mu\mathbf{F}}{I_C+1} + \left(\lambda - \frac{3\mu}{4}\right)\text{trace}(\boldsymbol{\varepsilon})\mathbf{F}^{-T}, \quad \text{using } \mathbf{F}\mathbf{F}^T \approx \mathbf{I} + 2\boldsymbol{\varepsilon} \\
&= 2\mu\boldsymbol{\varepsilon}\mathbf{F}^{-T} + \mu\left(\frac{\mathbf{I}}{4} - \frac{\mathbf{F}\mathbf{F}^T}{I_C+1}\right)\mathbf{F}^{-T} + \left(\lambda - \frac{3\mu}{4}\right)\text{trace}(\boldsymbol{\varepsilon})\mathbf{F}^{-T} \\
&= 2\mu\boldsymbol{\varepsilon} + \mu\left(\frac{\mathbf{I}}{4} - \frac{\mathbf{I}+2\boldsymbol{\varepsilon}}{I_C+1}\right) + \left(\lambda - \frac{3\mu}{4}\right)\text{trace}(\boldsymbol{\varepsilon})\mathbf{I}, \quad \text{using } \mathbf{F}^{-T} \approx \mathbf{I} \text{ and } \mathbf{F}\mathbf{F}^T \approx \mathbf{I} + 2\boldsymbol{\varepsilon} \\
&= 2\mu\boldsymbol{\varepsilon} + \mu\left(\frac{\mathbf{I}}{4} - \frac{\mathbf{I}+2\boldsymbol{\varepsilon}}{4+2\text{trace}(\boldsymbol{\varepsilon})}\right) + \left(\lambda - \frac{3\mu}{4}\right)\text{trace}(\boldsymbol{\varepsilon})\mathbf{I}, \quad \text{using } I_C \approx 3 + 2\text{trace}(\boldsymbol{\varepsilon}) \\
&= 2\mu\boldsymbol{\varepsilon} + \mu\left(\frac{\mathbf{I}}{4} - \frac{1}{4}(\mathbf{I}+2\boldsymbol{\varepsilon})\left(1 - \frac{\text{trace}(\boldsymbol{\varepsilon})}{2}\right)\right) + \left(\lambda - \frac{3\mu}{4}\right)\text{trace}(\boldsymbol{\varepsilon})\mathbf{I},
\end{aligned}$$

$$
\begin{aligned}
&\text{using } \left(1+\frac{\text{trace}(\boldsymbol{\varepsilon})}{2}\right)^{-1} \approx \left(1-\frac{\text{trace}(\boldsymbol{\varepsilon})}{2}\right) \\
&= 2\mu\boldsymbol{\varepsilon} - \frac{\mu\boldsymbol{\varepsilon}}{2} + \frac{\mu\text{trace}(\boldsymbol{\varepsilon})\mathbf{I}}{8} + \frac{\mu\text{trace}(\boldsymbol{\varepsilon})\boldsymbol{\varepsilon}}{4} + \left(\lambda - \frac{3\mu}{4}\right)\text{trace}(\boldsymbol{\varepsilon})\mathbf{I} \\
&= \frac{3}{2}\mu\boldsymbol{\varepsilon}\left(1+\frac{\text{trace}(\boldsymbol{\varepsilon})}{6}\right) + \left(\lambda - \frac{5\mu}{8}\right)\text{trace}(\boldsymbol{\varepsilon})\mathbf{I} \\
&= \frac{3}{2}\mu\boldsymbol{\varepsilon} + \left(\lambda - \frac{5\mu}{8}\right)\text{trace}(\boldsymbol{\varepsilon})\mathbf{I} \quad \text{using } \left(1+\frac{\text{trace}(\boldsymbol{\varepsilon})}{6}\right) \approx 1
\end{aligned} \tag{D.7}
$$

So comparing them with $2\mu_{\text{Lamé}} = \frac{3}{2}\mu \Longrightarrow \mu = \frac{4}{3}\mu_{\text{Lamé}}$ and $\lambda_{\text{Lamé}} = \lambda - \frac{5}{8}\mu \Longrightarrow \lambda = \lambda_{\text{Lamé}} + \frac{5}{8}\mu \Longrightarrow \lambda = \lambda_{\text{Lamé}} + \frac{5}{6}\mu_{\text{Lamé}}$. The approximations we use can be found in [Wikipedia, 2022].

This page was intentionally left blank.

Appendix E

Geometric Interpretation of Strain Energy and Damage

E.1 Edge Length Dependence of Isotropic Strain Energy Density

Here we prove that the strain energy can be represented as a function of the edge lengths of the simulation mesh.

Theorem E.1. *Every element of the set of invariants* $\mathbf{I}_V = \{I_C, II_C, III_C, IV_C, V_C\}$ *can be expressed in closed form using only the length of the edges of a mesh used in FEM.*

Proof. Let us assume an element, $\mathbf{\Delta}_e$, of a mesh used in FEM in k-dimensional space, which consists of Z_l edges. Let τ_{ij} be the stretch ratio of an edge formed by nodes i and j of the same element,

$$\tau_{ij} = \frac{d^e_{ij}}{D^e_{ij}} \tag{E.1}$$

where d^e_{ij} and D^e_{ij} denote current and initial length of the edge respectively.

Now, projecting the Right Cauchy-Green deformation tensor $\mathbf{C}$ along the edges, we can write,

$$\tau^2_{ij} = \mathbf{C} \cdot \left(\mathbf{q}^e_{ij} \otimes \mathbf{q}^e_{ij}\right) = \mathbf{C} \cdot \mathbf{Q}^e_{ij} \tag{E.2}$$

where $\otimes$ denotes tensor product and $\mathbf{Q}^e_{ij} \in \mathbb{R}^{k \times k}$. The projection vector, $\mathbf{q}^e_{ij}$, can be written in a k-dimensional space as

$$\mathbf{q}^e_{ij} = \begin{bmatrix} \hat{\mathbf{d}}^e_{ij} \cdot \hat{\mathbf{i}}_1 & \hat{\mathbf{d}}^e_{ij} \cdot \hat{\mathbf{i}}_2 & \dots & \hat{\mathbf{d}}^e_{ij} \cdot \hat{\mathbf{i}}_k \end{bmatrix}^T$$

where $\{\hat{\mathbf{i}}_1, \hat{\mathbf{i}}_2 \dots \hat{\mathbf{i}}_k\}$ denotes a set of orthogonal basis vectors of the k-dimensional space. Extending this formulation to all Z_l edges, we get

$$\mathscr{T} = \mathbf{C} \cdot \boldsymbol{\mathcal{Q}} \tag{E.3}$$

where $\mathscr{T} \in \mathbb{R}^{Z_l}$ and $\boldsymbol{\mathcal{Q}} \in \mathbb{R}^{k \times k \times Z_l}$, whose entries are equal to τ^2_{ij} and $\mathbf{Q}^e_{ij}$ respectively.

In order to express $\mathbf{C}$ in terms of τ^e_{ij} we need to invert $\boldsymbol{\mathcal{Q}}$, which is not always possible. However, we can determine $\mathbf{C}$ as the solution to the following optimization problem

$$\mathbf{C} = \underset{\hat{\mathbf{C}}}{\operatorname{argmin}} \ ||\hat{\mathbf{C}} \cdot \boldsymbol{\mathcal{Q}} - \mathscr{T}||^2_2 \tag{E.4}$$

The solution to this optimization problem allows us to express $\mathbf{C}$ in terms of $\mathscr{T}$ and $\boldsymbol{\mathcal{Q}}$.

The solution to the problem is given in closed form by

$$\hat{\mathbf{C}} = \mathscr{T} \cdot [\boldsymbol{\mathcal{Q}} \otimes \boldsymbol{\mathcal{Q}}]^{\dagger} \cdot \boldsymbol{\mathcal{Q}} \tag{E.5}$$

where † denotes pseudo-inverse.

It is evident from Equation E.5 that Right Cauchy-Green deformation tensor, $\mathbf{C}$, is a function of the length of the edges, d^e_{ij}. Hence, so are all elements of $\mathbf{I}_V$ and therefore, so is the hyper-elastic strain energy density, $\boldsymbol{\Psi}^e$. □

However, the anisotropic energy we use in Equation 4.36 contains a new invariant I_4 which is not based on right Cauchy-Green deformation tensor $\mathbf{C}$, but on the stretch matrix $\mathbf{S}$. However, Theorem E.1 deals only with $\mathbf{C}$-based invariants. Thus arguments presented in Theorem E.1 are

not sufficient for incorporating the anisotropic invariant I_4 and the anisotropic strain energy density, $\mathbf{\Psi}^e_{AA}$. Therefore, next we derive $\mathbf{S}$ in terms of the length of the edges of a mesh used in FEM simulation.

E.1.1 Discussion on the Proof

First let us write Equation E.5 in summation notation for a 2D triangular element for more coherency. To do that first we rewrite $\mathscr{T}$, $\mathbf{C}$ and $\mathbf{Q}^e_{ij}$ in vector format.

$$\begin{aligned}
\mathscr{T} &= \left[\mathscr{T}_1^2, \mathscr{T}_2^2, \mathscr{T}_3^2\right]^T \\
\mathbf{C} &= \left[\mathbf{c}_{xx}, \mathbf{c}_{yy}, \sqrt{2}\mathbf{c}_{xy}\right]^T \\
\mathbf{Q}_l &= \left[\mathbf{q}_{lx}^2, \mathbf{q}_{ly}^2, \sqrt{2}\mathbf{q}_{lx}\mathbf{q}_{ly}\right]^T \quad l \in Z_l \\
\mathscr{T}_l^2 &= \mathbf{C} \cdot \mathbf{Q}_l
\end{aligned} \tag{E.6}$$

where l denotes the set of three edges, Z_l, between nodes i and j. Similarly, the vectorized format $\mathbf{Q}^e_{ij}$ is $\mathbf{Q}_l$. Following the same arguments as before the final expression can be written as

$$\begin{aligned}
\mathbf{C} &= \underset{\hat{\mathbf{C}}}{\operatorname{argmin}} \sum_{l=1}^{3} ||\hat{\mathbf{C}} \cdot \mathbf{Q}_l - \mathscr{T}_l^2||_2^2 \\
\hat{\mathbf{C}} &= \sum_{l=1}^{3} \mathscr{T}_l^2 \left[\sum_{l=1}^{3} \mathbf{Q}_l \otimes \mathbf{Q}_l\right]^{\dagger} \mathbf{Q}_l
\end{aligned} \tag{E.7}$$

In Equation E.7 the quantity $\sum_{l=1}^{3} \mathbf{Q}_l \otimes \mathbf{Q}_l$ is invertible only when the area/volume of a 2D/3D element is non-zero i.e. when it does not contain any degeneracy. In that case Equation E.7 reduce to $\hat{\mathbf{C}} = \sum_{l=1}^{3} \mathscr{T}_l^2 \mathbf{Q}_l^{-1}$ which is exactly equal to Equation E.3 in matrix form. Thus, in such a scenario, the residual of Equation E.5 goes to zero. On the other hand, when the form is not invertible for degenerate elements, $\hat{\mathbf{C}}$ becomes a close approximation of $\mathbf{C}$. However, in both cases, the final expression depends only on the edge lengths of the element.

E.2 Edge Length Dependence of Anisotropic Strain Energy Density

Theorem E.2. *The anisotropic invariant I_4 can be expressed in closed form using only the length of the edges of a mesh used in FEM.*

Proof. Let us define stretch ratio of an edge, $\tilde{\lambda}_{ij}$, formed by node m and n of an element Δ_e of FEM mesh as

$$\tilde{\lambda}_{ij} = \frac{|d^e_{ij}|}{|D^e_{ij}|} \tag{E.8}$$

where d^e_{ij} and D^e_{ij} denote current and initial length of the edge respectively.

Using similar arguments as presented in Theorem E.1, we can now write

$$\tilde{\lambda}_{ij} = \mathbf{S} \cdot \mathbf{O}^e_{ij} = \mathbf{S} \cdot \left(\mathbf{Q}^e_{ij}\right)^{\circ \frac{1}{2}} \tag{E.9}$$

where $\{.\}^{\circ \frac{1}{2}}$ denotes element-wise square root (Hadamard product) matrix $\mathbf{Q}^e_{ij}$.

Following the same steps as shown in Theorem E.1, it is easy to derive the final answer as

$$\mathscr{L} = \underset{\hat{\mathbf{S}}}{\operatorname{argmin}} \ ||\hat{\mathbf{S}} \cdot \boldsymbol{O} - \tilde{\boldsymbol{\Lambda}}||^2 \tag{E.10}$$

$$\hat{\mathbf{S}} = \tilde{\boldsymbol{\Lambda}} \cdot [\boldsymbol{O} \otimes \boldsymbol{O}]^{\dagger} \cdot \boldsymbol{Y} \tag{E.11}$$

As I_4 is a function of $\mathbf{S}$, it can be concluded that I_4 can also be expressed in closed form using only the length of the edges. This concludes the proof. □

Thus it is also proved that anisotropic strain energy, $\boldsymbol{\Psi}^e_{AA}$, which consists of a $\mathbf{S}$-based invariant, I_4, and a $\mathbf{C}$-based invariant, I_5, depends only on the length of the edges of the FEM mesh.

E.3 Geometric Interpretation of Edge-based Damage

The theorems proved in Sections E.1 and E.2 together give a geometric interpretation of any hyper-elastic strain energy in terms of the edge length for an undamaged mesh. However, they

do not adequately explore the geometric interpretation of edge-based fracture that occurs in graph-based FEM.

To this end, we present a novel theoretical approach to represent any kind of material damage with an appropriate metric embedded in a Riemannian manifold.

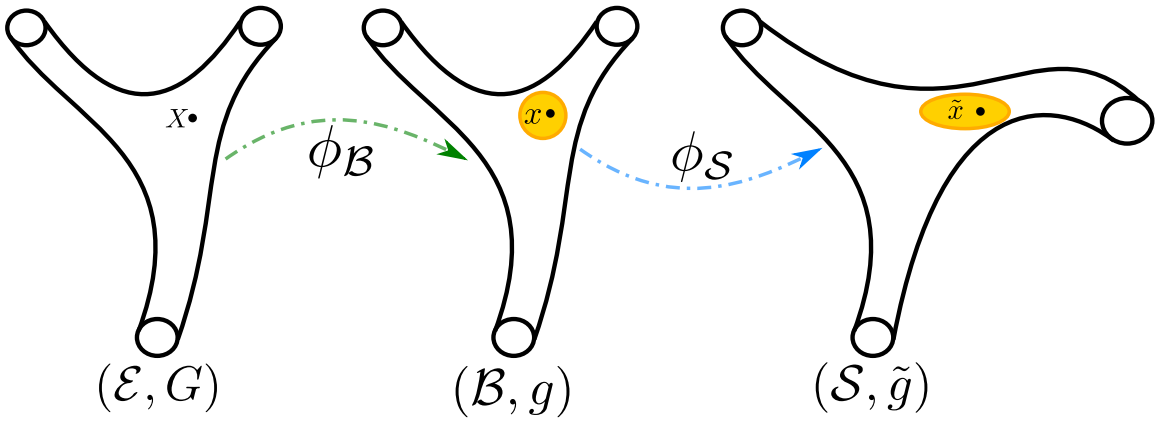


Figure E.1: Geometric description of the damage (shown in yellow) and deformation as two smooth mappings from Euclidean to Riemannian manifold.

Let us assume that the initial undeformed configuration of a tetrahedral mesh is embedded in a Euclidean manifold $\mathscr{E}$. As shown in Figure E.1, the fracture/damage of a mesh without deformation can be interpreted as a smooth map, $\phi_{\mathscr{B}}$, from Euclidean manifold $\mathscr{E}$ to Riemannian manifold $\mathscr{B}$ [Rastiello et al., 2018] [Das et al., 2021] with corresponding Euclidean metric $\boldsymbol{G}$ and Riemannian metric $\boldsymbol{g}$ [Yavari and Goriely, 2012] [Yavari and Marsden, 2012]. The deformation of the fractured/damaged mesh can then be interpreted as another smooth map, $\phi_{\mathscr{S}}$, from Riemannian manifold $\mathscr{B}$ to another Riemannian manifold $\mathscr{S}$ with an associated metric $\tilde{\boldsymbol{g}}$ for later.

$$\begin{aligned} \phi_{\mathscr{B}} &: \mathscr{E} \longrightarrow \mathscr{B} \\ \phi_{\mathscr{S}} &: \mathscr{B} \longrightarrow \mathscr{S} \end{aligned} \tag{E.12}$$

The corresponding deformation gradients are the tangent map of $\phi_{\mathscr{B}}$ & $\phi_{\mathscr{S}}$, and are denoted by a linear map as

$$\begin{aligned} \mathbf{F}_{\mathscr{B}} &: T_{\boldsymbol{X}}\mathscr{E} \longrightarrow T_{\boldsymbol{x}}\mathscr{B} \quad \forall \boldsymbol{X} \in \mathscr{E},\ \forall \boldsymbol{x} \in \mathscr{B} \\ \mathbf{F}_{\mathscr{S}} &: T_{\boldsymbol{x}}\mathscr{B} \longrightarrow T_{\tilde{\boldsymbol{x}}}\mathscr{S} \quad \forall \boldsymbol{x} \in \mathscr{B},\ \forall \tilde{\boldsymbol{x}} \in \mathscr{S} \end{aligned} \tag{E.13}$$

Assuming $\phi_{\mathscr{B}}$ is an identity map, the final deformation gradient from undeformed and undamaged Euclidean manifold to deformed and damaged Riemannian manifold is given by

$$\mathbf{F} = \mathbf{F}_{\mathscr{S}}\mathbf{F}_{\mathscr{B}} = \mathbf{F}_{\mathscr{S}} \tag{E.14}$$

In this formulation, dislocations are represented by the evolving geometry of the material manifold which is captured by the metric $\tilde{\boldsymbol{g}}$. The transpose of the deformation gradient is defined as

$$\mathbf{F}^{T}_{\mathscr{S}} : T_{\tilde{\boldsymbol{x}}}\mathscr{S} \longrightarrow T_{\boldsymbol{x}}\mathscr{B} \quad \forall \boldsymbol{x} \in \mathscr{B},\ \forall \tilde{\boldsymbol{x}} \in \mathscr{S} \tag{E.15}$$

Finally, pulled back to the undeformed and undamaged configuration in the Euclidean manifold, the right Cauchy-Green deformation tensor for the deformed and damaged configuration in the Riemannian manifold can be represented as

$$\begin{aligned} &\mathbf{C} : T_{\boldsymbol{X}}\mathscr{E} \longrightarrow T_{\boldsymbol{X}}\mathscr{E} \\ &\mathbf{C} = \mathbf{F}^{T}_{\mathscr{S}}\tilde{\boldsymbol{g}}\mathbf{F}_{\mathscr{S}} = \mathbf{F}^{T}\tilde{\boldsymbol{g}}\mathbf{F} \end{aligned} \tag{E.16}$$

Interested readers may check [Yavari and Goriely, 2012] [Yavari and Marsden, 2012] for a more detailed and rigorous analysis of the geometric representation of deformation and damage.

Analytically defining the Riemann metric $\tilde{\boldsymbol{g}}$ for complex fracture manifold is extremely difficult. However, in our case we can incorporate the effect of the Riemann metric $\tilde{\boldsymbol{g}}$ in Theorem E.1 and Theorem E.2 by redefining the stretch ratio (Equation E.1 and Equation E.8) of an edge as

$$\begin{aligned} \lambda^{2}_{ij} &= I_{ij}\left[\frac{d^{e}_{ij} - d^{ep}_{ij}}{D^{e}_{ij}}\right]^{2} + \kappa_{e}\left(1 - I_{ij}\right)\left[\frac{d^{e}_{ij}}{D^{e}_{ij}}\right]^{2} \\ \tilde{\lambda}_{ij} &= I_{ij}\frac{|d^{e}_{ij} - d^{ep}_{ij}|}{|D^{e}_{ij}|} + \kappa_{e}\left(1 - I_{ij}\right)\frac{|d^{e}_{ij}|}{|D^{e}_{ij}|} \\ I_{ij} &= \begin{cases} 1 & \text{damaged edge} \\ 0 & \text{undamaged edge} \end{cases} \end{aligned} \tag{E.17}$$

where d^{ep}_{ij} denote the length of crack opening on the edge between nodes i and j due to fracture. The parameter κ_e is similar to $f(\Phi_l)$ from Equation E.18 which denotes the percentage of damage in Δ_e. hyper

$$\begin{aligned} \boldsymbol{\Psi}^{e}_{frac} &= f(\Phi_l)\boldsymbol{\Psi}^{e}_{ori} \\ &= \left[\frac{|\sigma^{e'}_{12}| + |\sigma^{e'}_{13}| + |\sigma^{e'}_{14}| + |\sigma^{e'}_{23}| + |\sigma^{e'}_{24}| + |\sigma^{e'}_{34}|}{|\sigma^{e}_{12}| + |\sigma^{e}_{13}| + |\sigma^{e}_{14}| + |\sigma^{e}_{23}| + |\sigma^{e}_{24}| + |\sigma^{e}_{34}|}\right]\boldsymbol{\Psi}^{e}_{ori} \end{aligned} \tag{E.18}$$

The parameter $f(\Phi_l)$ denotes the ratio of total edge stress after and before fracture. Parameters $\boldsymbol{\Psi}^{e}_{ori}$ and $\boldsymbol{\Psi}^{e}_{frac}$ denote hyper-elastic strain energy density before and after fracture. Here $\{\sigma^{e}_{12} \ldots \sigma^{e}_{34}\}$ and $\{\sigma^{e'}_{12} \ldots \sigma^{e'}_{34}\}$ are the components of normal stress tensor along the direction of the edges before and after fracture respectively.

The rest of the derivations for Theorem E.1 and Theorem E.2 remain the same. The Riemann metric $\tilde{\boldsymbol{g}}$ due to fracture can then be obtained by solving $\hat{\mathbf{C}}$ from Equation E.5 and $\hat{\mathbf{S}}$ from Equation E.11 together.

This page was intentionally left blank.

References

P.M. Adler. 1992. *Porous Media: Geometry and Transports*. Butterworth-Heinemann.

Hanen Amor, Jean-Jacques Marigo, and Corrado Maurini. 2009. Regularized formulation of the variational brittle fracture with unilateral contact: Numerical experiments. *Journal of the Mechanics and Physics of Solids* 57, 8 (2009), 1209–1229. https://doi.org/10.1016/j.jmps.2009.04.011

K. Aoki, N.H. Dong, T. Kaneko, and S. Kuriyama. 2004. Physically based simulation of cracks on drying 3D solids. In *Proceedings Computer Graphics International*. ACM, 357–364. https://doi.org/10.1109/CGI.2004.1309233

Pedro M. A. Areias and Ted Belytschko. 2005. Analysis of three-dimensional crack initiation and propagation using the extended finite element method. *Internat. J. Numer. Methods Engrg.* 63, 5 (2005), 760–788. https://doi.org/10.1002/nme.1305

David Arthur and Sergei Vassilvitskii. 2007. K-Means++: The Advantages of Careful Seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (New Orleans, Louisiana) *(SODA '07)*. Society for Industrial and Applied Mathematics, USA, 1027–1035.

Zhaosheng Bao, Jeong-mo Hong, Joseph Teran, and Ronald Fedkiw. 2007. Fracturing Rigid Materials. *IEEE Transactions on Visualization and Computer Graphics* 13, 2 (2007), 370–378. https://doi.org/10.1109/TVCG.2007.39

Albert-László Barabási and Réka Albert. 1999. Emergence of Scaling in Random Networks. *Science* 286, 5439 (1999), 509–512.

David Baraff, Andrew Witkin, and Michael Kass. 2003. Untangling Cloth. In *ACM SIGGRAPH 2003 Papers* (San Diego, California) *(SIGGRAPH '03)*. ACM, New York, NY, USA, 862–870. https://doi.org/10.1145/1201775.882357

J. Barbič and D. L. James. 2009. Six-DoF haptic rendering of contact between geometrically complex reduced deformable models: Haptic demo. In *World Haptics 2009 - Third Joint*

EuroHaptics conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. Institute of Electrical and Electronics Engineers, 393–394. https://doi.org/10.1109/WHC.2009.4810910

Adam W. Bargteil, Chris Wojtan, Jessica K. Hodgins, and Greg Turk. 2007. A Finite Element Method for Animating Large Viscoplastic Flow. *ACM Transactions on Graphics* 26, 3 (jul 2007), 16–es. https://doi.org/10.1145/1276377.1276397

J. Bear. 1988. *Dynamics of Fluids in Porous Media*. Dover.

T. Belytschko and T. Black. 1999. Elastic crack growth in finite elements with minimal remeshing. *Internat. J. Numer. Methods Engrg.* 45, 5 (1999), 601–620. https://doi.org/10.1002/(SICI)1097-0207(19990620)45:5<601::AID-NME598>3.0.CO;2-S

F. Bergaya and G. Lagaly. 2013. *Handbook of Clay Science*. Elsevier Science.

Julian Besag. 1974. Spatial Interaction and the Statistical Analysis of Lattice Systems. *Journal of the Royal Statistical Society: Series B (Methodological)* 36, 2 (1974), 192–225.

Blender. 2023. Blender. Retrieved Feb 10, 2023 from https://www.blender.org/

Christopher Brandt, Elmar Eisemann, and Klaus Hildebrandt. 2018. Hyper-Reduced Projective Dynamics. *ACM Transactions on Graphics* 37, 4, Article 80 (jul 2018), 13 pages. https://doi.org/10.1145/3197517.3201387

CallOfDuty. 2023. Call of Duty Game. Retrieved January 21, 2023 from https://www.callofduty.com/

Wei Chen, Fei Zhu, Jing Zhao, Sheng Li, and Guoping Wang. 2018. Peridynamics-Based Fracture Animation for Elastoplastic Solids. *Computer Graphics Forum* 37, 1 (2018), 112–124. https://doi.org/10.1111/cgf.13236

Zhili Chen, Miaojun Yao, Renguo Feng, and Huamin Wang. 2014. Physics-Inspired Adaptive Fracture Refinement. *ACM Transactions on Graphics* 33, 4, Article 113 (jul 2014), 7 pages. https://doi.org/10.1145/2601097.2601115

Floyd M. Chitalu. 2020. MCUT. Retrieved August 04, 2022 from https://github.com/cutdigital/mcut

Floyd M. Chitalu, Qinghai Miao, Kartic Subr, and Taku Komura. 2020. Displacement-Correlated XFEM for Simulating Brittle Fracture. *Computer Graphics Forum* 39, 2 (2020), 569–583. https://doi.org/10.1111/cgf.13953

CounterStrike. 2023. Counter Strike Game. Retrieved January 21, 2023 from https://blog.counter-strike.net/

Frank Dachille, Hong Qin, Arie Kaufman, and Jihad El-Sana. 1999. Haptic Sculpting of Dynamic Surfaces. In *Proceedings of the 1999 Symposium on Interactive 3D Graphics* (Atlanta, Georgia, USA) *(I3D '99)*. ACM, New York, NY, USA, 103–110. https://doi.org/10.1145/300523.300535

Sanhita Das, Shubham Sharma, Ananth Ramaswamy, Debasish Roy, and J. N. Reddy. 2021. A Geometrically Inspired Model for Brittle Damage in Compressible Elastomers. *Journal of Applied Mechanics* 88, 8 (04 2021). https://doi.org/10.1115/1.4050620

Jessica T. Davis, Matteo Chinazzi, Nicola Perra, Kunpeng Mu, Ana Pastore y Piontti, Marco Ajelli, Natalie E. Dean, Corrado Gioannini, Maria Litvinova, Stefano Merler, Luca Rossi, Kaiyuan Sun, Xinyue Xiong, Ira M. Longini, M. Elizabeth Halloran, Cécile Viboud, and Alessandro Vespignani. 2021. Cryptic transmission of SARS-CoV-2 and the first COVID-19 wave. *Nature* 600, 7887 (2021), 127–132.

Kenny Erleben, Jon Sporring, Knud Henriksen, and Kenrik Dohlman. 2005a. *Physics-based Animation (Graphics Series)*. Charles River Media, Inc.

Kenny Erleben, Jon Sporring, Knud Henriksen, and Kenrik Dohlman. 2005b. *Physics-Based Animation (Graphics Series)*. Charles River Media, Inc., USA.

J. D. Eshelby. 1957. The Determination of the Elastic Field of an Ellipsoidal Inclusion, and Related Problems. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences* 241, 1226 (1957), 376–396.

ExpeditionWorkshed. 2013. Brazilian Test - Tensile Failure of Concrete in Slow Motion.

Ihor Farmaga, Petro Shmigelskyi, Piotr Spiewak, and Lukasz Ciupinski. 2011. Evaluation of computational complexity of finite element analysis. In *2011 11th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)*. 213–214.

Yun (Raymond) Fei, Christopher Batty, Eitan Grinspun, and Changxi Zheng. 2018. A Multi-Scale Model for Simulating Liquid-Fabric Interactions. *ACM Transactions on Graphics* 37, 4, Article 51 (jul 2018), 16 pages. https://doi.org/10.1145/3197517.3201392

R.A. Fisher. 1925. *Statistical methods for research workers*. Edinburgh Oliver & Boyd.

Robert G. Gallager. 2013. *Stochastic Processes: Theory for Applications*. Vol. 1. Cambridge University Press.

Zhan Gao and Ian Gibson. 2006. Haptic Sculpting of Multi-resolution B-spline Surfaces with Shaped Tools. *Computer-Aided Design* 38, 6 (2006), 661–676. https://doi.org/10.1016/j.cad.2006.02.004

Saeid Ghouli, Bahador Bahrami, Majid R. Ayatollahi, Thomas Driesner, and Morteza Nejati. 2021. Introduction of a Scaling Factor for Fracture Toughness Measurement of Rocks Using the Semi-circular Bend Test. *Rock Mechanics and Rock Engineering* 54, 8 (2021), 4041 – 4058. https://doi.org/10.1007/s00603-021-02468-1

E. N. Gilbert. 1959. Random Graphs. *The Annals of Mathematical Statistics* 30, 4 (1959), 1141 – 1144.

Loeiz Glondu, Maud Marchal, and Georges Dumont. 2013. Real-Time Simulation of Brittle Fracture Using Modal Analysis. *IEEE Transactions on Visualization and Computer Graphics* 19, 2 (2013), 201–209. https://doi.org/10.1109/TVCG.2012.121

A. A. Griffith. 1921. The Phenomena of Rupture and Flow in Solids. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character* 221 (1921), 163–198.

Chris Gunn. 2006. Collaborative Virtual Sculpting with Haptic Feedback. *Virtual Reality* 10, 2 (oct 2006), 73–83.

Philip E. Gustafson and Brian A. Hagler. 1999. Gaussian quadrature rules and numerical examples for strong extensions of mass distribution functions. *J. Comput. Appl. Math.* 105, 1 (1999), 317–326. https://doi.org/10.1016/S0377-0427(99)00023-0

David Hahn and Chris Wojtan. 2015. High-Resolution Brittle Fracture Simulation with Boundary Elements. *ACM Transactions on Graphics* 34, 4, Article 151 (jul 2015), 12 pages. https://doi.org/10.1145/2766896

David Hahn and Chris Wojtan. 2016. Fast Approximations for Boundary Element Based Brittle Fracture Simulation. *ACM Transactions on Graphics* 35, 4, Article 104 (jul 2016), 11 pages. https://doi.org/10.1145/2897824.2925902

Koichi Hirota, Yasuyuki Tanoue, and Toyohisa Kaneko. 1998. Generation of crack patterns with a physical model. *The Visual Computer* 14, 3 (1998), 126 – 137. https://doi.org/10.1007/s003710050128

Koichi Hirota, Yasuyuki Tanoue, and Toyohisa Kaneko. 2000. Simulation of three-dimensional cracks. *The Visual Computer* 16, 7 (2000), 371 – 378. https://doi.org/10.1007/s003710000069

Yuanming Hu, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Andre Pradhana, and Chenfanfu Jiang. 2018a. A Moving Least Squares Material Point Method with Displacement Discontinuity and Two-Way Rigid Body Coupling. *ACM Transactions on Graphics* 37, 4, Article 150 (jul 2018), 14 pages. https://doi.org/10.1145/3197517.3201293

Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018b. Tetrahedral Meshing in the Wild. *ACM Transactions on Graphics* 37, 4, Article 60 (jul 2018), 14 pages. https://doi.org/10.1145/3197517.3201353

G. Irving, J. Teran, and R. Fedkiw. 2004. Invertible Finite Elements for Robust Simulation of Large Deformation. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Grenoble, France) *(SCA '04)*. Eurographics Association, Goslar, DEU, 131–140. https://doi.org/10.1145/1028523.1028541

Ernst Ising. 1925. Beitrag zur Theorie des Ferromagnetismus. *Zeitschrift für Physik* 31, 1 (1925), 253–258.

Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. 2011. Bounded Biharmonic Weights for Real-Time Deformation. *ACM Transactions on Graphics* 30, 4, Article 78 (jul 2011), 8 pages. https://doi.org/10.1145/2010324.1964973

Robert Jagnow and Julie Dorsey. 2002. Virtual Sculpting with Haptic Displacement Maps. In *Proceedings of the Graphics Interface 2002 Conference* (Calgary, Alberta). Canadian Information Processing Society, 125–132.

Doug L. James and Dinesh K. Pai. 1999. ArtDefo: Accurate Real Time Deformable Objects. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., USA, 65–72.

Svante Janson and Lutz Warnke. 2021. Preferential attachment without vertex growth: Emergence of the giant component. *The Annals of Applied Probability* 31, 4 (2021), 1523 – 1547.

Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. 2016. The Material Point Method for Simulating Continuum Materials. In *ACM SIGGRAPH 2016 Courses* (Anaheim, California) *(SIGGRAPH '16)*. ACM, New York, NY, USA, Article 24, 52 pages. https://doi.org/10.1145/2897826.2927348

Tao Ju, Scott Schaefer, and Joe Warren. 2005. Mean Value Coordinates for Closed Triangular Meshes. *ACM Transactions on Graphics* 24, 3 (jul 2005), 561–566. https://doi.org/10.1145/1073204.1073229

Peter I. Kattan. 2008. *The Linear Tetrahedral (Solid) Element*. Springer Berlin Heidelberg, Berlin, Heidelberg, 337–365.

Parisa Khodabakhshi, J. N. Reddy, and Arun Srinivasa. 2016. GraFEA: a graph-based finite element approach for the study of damage and fracture in brittle materials. *Meccanica* 51, 12 (2016), 3129 – 3147. https://doi.org/10.1007/s11012-016-0560-6

Parisa Khodabakhshi, J. N. Reddy, and Arun Srinivasa. 2019. A nonlocal fracture criterion and its effect on the mesh dependency of GraFEA. *Acta Mechanica* 230, 10 (2019), 3593–3612. https://doi.org/10.1007/s00707-019-02479-8

Ryo Kikuuwe, Hiroaki Tabuchi, and Motoji Yamamoto. 2009. An Edge-Based Computationally Efficient Formulation of Saint Venant-Kirchhoff Tetrahedral Finite Elements. *ACM Transactions on Graphics* 28, 1, Article 8 (feb 2009), 13 pages. https://doi.org/10.1145/1477926.1477934

Theodore Kim, Fernando De Goes, and Hayley Iben. 2019. Anisotropic Elasticity for Inversion-Safety and Element Rehabilitation. *ACM Transactions on Graphics* 38, 4, Article 69 (jul 2019), 15 pages. https://doi.org/10.1145/3306346.3323014

Theodore Kim and David Eberle. 2022. Dynamic Deformables: Implementation and Production Practicalities (Now with Code!). In *ACM SIGGRAPH 2022 Courses* (Vancouver, British Columbia, Canada) *(SIGGRAPH '22)*. ACM, New York, NY, USA, Article 7, 259 pages. https://doi.org/10.1145/3532720.3535628

Ross Kindermann and Laurie Snell. 1980. *Markov random fields and their applications*. Vol. 1. https://doi.org/10.1090/conm/001

Dan Koschier, Jan Bender, and Nils Thuerey. 2017. Robust EXtended Finite Elements for Complex Cutting of Deformables. *ACM Transactions on Graphics* 36, 4, Article 55 (jul 2017), 13 pages. https://doi.org/10.1145/3072959.3073666

Dan Koschier, Sebastian Lipponer, and Jan Bender. 2015. Adaptive Tetrahedral Meshes for Brittle Fracture Simulation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Copenhagen, Denmark) *(SCA '14)*. Eurographics Association, Goslar, DEU, 57–66.

J. A. Levine, A. W. Bargteil, C. Corsi, J. Tessendorf, and R. Geist. 2015. A Peridynamic Perspective on Spring-Mass Fracture. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Copenhagen, Denmark) *(SCA '14)*. Eurographics Association, Goslar, DEU, 47–55.

Avirup Mandal, Parag Chaudhuri, and Subhasis Chaudhuri. 2022a. Interactive Physics-Based Virtual Sculpting with Haptic Feedback. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 5, 1, Article 9 (may 2022), 20 pages. https://doi.org/10.1145/3522611

Avirup Mandal, Parag Chaudhuri, and Subhasis Chaudhuri. 2022b. Simulating Fracture in Anisotropic Materials Containing Impurities. In *Proceedings of the 15th ACM SIGGRAPH Conference on Motion, Interaction and Games* (Guanajuato, Mexico) *(MIG '22)*. ACM, New York, NY, USA, Article 3, 10 pages. https://doi.org/10.1145/3561975.3562956

Avirup Mandal., Parag Chaudhuri., and Subhasis Chaudhuri. 2023. Real-Time Physics-Based Mesh Deformation with Haptic Feedback and Material Anisotropy. In *Proceedings of the 18th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - GRAPP*. INSTICC, SciTePress, 153–161. https://doi.org/10.5220/0011611800003417

Avirup Mandal, Parag Chaudhuri, and Subhasis Chaudhuri. 2023. Remeshing-free Graph-based Finite Element Method for Fracture Simulation. *Computer Graphics Forum* 42, 1 (2023), 117–134. https://doi.org/10.1111/cgf.14725

William A. McNeely, Kevin D. Puterbaugh, and James J. Troy. 1999. Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., USA, 401–408. https://doi.org/10.1145/311535.311600

Alexandre Mercier-Aubin, Paul G. Kry, Alexandre Winter, and David I. W. Levin. 2022a. Adaptive Rigidification of Elastic Solids. *ACM Transactions on Graphics* 41, 4, Article 71 (jul 2022), 11 pages. https://doi.org/10.1145/3528223.3530124

Alexandre Mercier-Aubin, Paul G. Kry, Alexandre Winter, and David I. W. Levin. 2022b. Adaptive Rigidification of Elastic Solids. *ACM Transactions on Graphics* 41, 4, Article 71 (jul 2022), 11 pages. https://doi.org/10.1145/3528223.3530124

Christian Miehe, Lisa-Marie Schänzel, and Heike Ulmer. 2015. Phase field modeling of fracture in multi-physics problems. Part I. Balance of crack surface and failure criteria for brittle crack propagation in thermo-elastic solids. *Computer Methods in Applied Mechanics and Engineering* 294 (2015), 449–485. https://doi.org/10.1016/j.cma.2014.11.016

Nicolas Moës, John Dolbow, and Ted Belytschko. 1999. A finite element method for crack growth without remeshing. *Internat. J. Numer. Methods Engrg.* 46, 1 (1999), 131–150. https://doi.org/10.1002/(SICI)1097-0207(19990910)46:1<131::AID-NME726>3.0.CO;2-J

Neil Molino, Zhaosheng Bao, and Ron Fedkiw. 2004. A Virtual Node Algorithm for Changing Mesh Topology during Simulation. *ACM Transactions on Graphics* 23, 3 (aug 2004), 385–392. https://doi.org/10.1145/1015706.1015734

Tomas Möller. 1997. A Fast Triangle-Triangle Intersection Test. *Journal of Graphics Tools* 2, 2 (1997), 25–30. https://doi.org/10.1080/10867651.1997.10487472

B. Müller, F. Kummer, and M. Oberlack. 2013. Highly accurate surface and volume integration on implicit domains by means of moment-fitting. *Internat. J. Numer. Methods Engrg.* 96, 8 (2013), 512–528. https://doi.org/10.1002/nme.4569

Matthias Müller and Markus Gross. 2004. Interactive Virtual Materials. In *Proceedings of Graphics Interface 2004* (London, Ontario, Canada) *(GI '04)*. Canadian Human-Computer Communications Society, Waterloo, CAN, 239–246.

Alan Norton, Greg Turk, Bob Bacon, John Gerth, and Paula Sweeney. 1991. Animation of Fracture by Physical Modeling. *The Visual Computer* 7, 4 (jul 1991), 210–219. https://doi.org/10.1007/BF01900837

OrthoInf o AAOS. 2020. *Femur Shaft Fractures (Broken Thighbone)*.

James F. O'Brien, Adam W. Bargteil, and Jessica K. Hodgins. 2002. Graphical Modeling and Animation of Ductile Fracture. *ACM Transactions on Graphics* 21, 3 (jul 2002), 291–294. https://doi.org/10.1145/566654.566579

James F. O'Brien and Jessica K. Hodgins. 1999. Graphical Modeling and Animation of Brittle Fracture. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., USA, 137–146. https://doi.org/10.1145/311535.311550

Michael Ortega, Stephane Redon, and Sabine Coquillart. 2007. A Six Degree-of-Freedom God-Object Method for Haptic Display of Rigid Bodies with Surface Properties. *IEEE Transactions on Visualization and Computer Graphics* 13, 3 (2007), 458–469. https://doi.org/10.1109/TVCG.2007.1028

M.A. Otaduy and M.C. Lin. 2005. Stable and responsive six-degree-of-freedom haptic manipulation using implicit integration. In *First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. World Haptics Conference*. Institute of Electrical and Electronics Engineers, 247–256. https://doi.org/10.1109/WHC.2005.120

Saket Patkar and Parag Chaudhuri. 2013. Wetting of Porous Solids. *IEEE Transactions on Visualization and Computer Graphics* 19, 9 (2013), 1592–1604. https://doi.org/10.1109/TVCG.2013.8

Herwig Peterlik, Paul Roschger, Klaus Klaushofer, and Peter Fratzl. 2006. From brittle to ductile fracture of bone. *Nature materials* 5, 1 (2006), 52–55.

Tobias Pfaff, Rahul Narain, Juan Miguel de Joya, and James F. O'Brien. 2014. Adaptive Tearing and Cracking of Thin Sheets. *ACM Transactions on Graphics* 33, 4, Article 110 (jul 2014), 9 pages. https://doi.org/10.1145/2601097.2601132

Boris Pittel. 2010. On a random graph evolving by degrees. *Advances in Mathematics* 223, 2 (2010), 619–671. https://doi.org/10.1016/j.aim.2009.08.015

Pixologic. 2023. ZBrush. Retrieved Feb 10, 2023 from https://pixologic.com/

Giuseppe Rastiello, Cédric Giry, Fabrice Gatuingt, and Rodrigue Desmorat. 2018. From diffuse damage to strain localization from an Eikonal Non-Local (ENL) Continuum Damage model with evolving internal length. *Computer Methods in Applied Mechanics and Engineering* 331 (2018), 650–674. https://doi.org/10.1016/j.cma.2017.12.006

J.N. Reddy and Arun Srinivasa. 2015. On the Force-Displacement Characteristics of Finite Elements for Elasticity and Related Problems. *Finite Elements in Analysis and Design* 104, C (oct 2015), 35–40. https://doi.org/10.1016/j.finel.2015.04.011

Mark W Schraad. 2014. A Theoretical Approach to the Coupled Fluid–Solid Physical Response of Porous and Cellular Materials: Dynamics. *Nonlinear Elasticity and Hysteresis: Fluid-Solid Coupling in Porous Media* (2014), 127–152.

Hang Si. 2015. TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Trans. Math. Software* 41, 2, Article 11 (feb 2015), 36 pages. https://doi.org/10.1145/2629697

Eftychios Sifakis and Jernej Barbic. 2012. FEM Simulation of 3D Deformable Solids: A Practitioner's Guide to Theory, Discretization and Model Reduction. In *ACM SIGGRAPH 2012 Courses* (Los Angeles, California) *(SIGGRAPH '12)*. ACM, New York, NY, USA, Article 20, 50 pages. https://doi.org/10.1145/2343483.2343501

Eftychios Sifakis, Kevin G. Der, and Ronald Fedkiw. 2007. Arbitrary Cutting of Deformable Tetrahedralized Objects. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (San Diego, California) *(SCA '07)*. Eurographics Association, Goslar, DEU, 73–80.

F. S. Sin, D. Schroeder, and J. Barbič. 2013. Vega: Non-Linear FEM Deformable Object Simulator. *Computer Graphics Forum* 32, 1 (2013), 36–48. https://doi.org/10.1111/j.1467-8659.2012.03230.x

Breannan Smith, Fernando De Goes, and Theodore Kim. 2018. Stable Neo-Hookean Flesh Simulation. *ACM Transactions on Graphics* 37, 2, Article 12 (mar 2018), 15 pages. https://doi.org/10.1145/3180491

Breannan Smith, Fernando De Goes, and Theodore Kim. 2019. Analytic Eigensystems for Isotropic Distortion Energies. *ACM Transactions on Graphics* 38, 1, Article 3 (feb 2019), 15 pages. https://doi.org/10.1145/3241041

A. R. Srinivasa. 2021. Discrete differential geometry and its role in computational modeling of defects and inelasticity. *Meccanica* 56, 7 (2021), 1847–1865. https://doi.org/10.1007/s11012-021-01335-1

Alexey Stomakhin, Russell Howes, Craig Schroeder, and Joseph M. Teran. 2012. Energetically Consistent Invertible Elasticity. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Lausanne, Switzerland) *(SCA '12)*. Eurographics Association, Goslar, DEU, 25–32.

Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. 2013. A Material Point Method for Snow Simulation. *ACM Transactions on Graphics* 32, 4, Article 102 (jul 2013), 10 pages. https://doi.org/10.1145/2461912.2461948

Min Tang, Dinesh Manocha, Miguel A. Otaduy, and Ruofeng Tong. 2012. Continuous Penalty Forces. *ACM Transactions on Graphics* 31, 4, Article 107 (jul 2012), 9 pages. https://doi.org/10.1145/2185520.2185603

Min Tang, Dinesh Manocha, and Ruofeng Tong. 2010a. Fast Continuous Collision Detection Using Deforming Non-Penetration Filters. In *Proceedings of the 2010 ACM SIGGRAPH*

Symposium on Interactive 3D Graphics and Games (Washington, D.C.) *(I3D '10)*. ACM, New York, NY, USA, 7–13. https://doi.org/10.1145/1730804.1730806

Min Tang, Dinesh Manocha, and Ruofeng Tong. 2010b. Fast Continuous Collision Detection Using Deforming Non-Penetration Filters. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (Washington, D.C.) *(I3D '10)*. ACM, New York, NY, USA, 7–13. https://doi.org/10.1145/1730804.1730806

Demetri Terzopoulos and Kurt Fleischer. 1988. Modeling Inelastic Deformation: Viscolelasticity, Plasticity, Fracture. *ACM SIGGRAPH Computer Graphics* 22, 4 (jun 1988), 269–278. https://doi.org/10.1145/378456.378522

J.P Tronskar, M.A Mannan, and M.O Lai. 2002. Measurement of fracture initiation toughness and crack resistance in instrumented Charpy impact testing. *Engineering Fracture Mechanics* 69, 3 (2002), 321–338. https://doi.org/10.1016/S0013-7944(01)00077-7

International Telecommunication Union. 2013. ITU-R The double-stimulus continuous quality-scale. https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.500-13-201201-I!!PDF-E.pdf. Accessed: 2021-04-17.

Bolun Wang, Zachary Ferguson, Teseo Schneider, Xin Jiang, Marco Attene, and Daniele Panozzo. 2021. A Large-Scale Benchmark and an Inclusion-Based Algorithm for Continuous Collision Detection. *ACM Transactions on Graphics* 40, 5, Article 188 (sep 2021), 16 pages. https://doi.org/10.1145/3460775

Yuting Wang, Chenfanfu Jiang, Craig Schroeder, and Joseph Teran. 2015. An Adaptive Virtual Node Algorithm with Robust Mesh Cutting. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Copenhagen, Denmark) *(SCA '14)*. Eurographics Association, Goslar, DEU, 77–85.

Jeffrey A. Weiss, Bradley N. Maker, and Sanjay Govindjee. 1996. Finite element implementation of incompressible, transversely isotropic hyperelasticity. *Computer Methods in Applied Mechanics and Engineering* 135, 1 (1996), 107–128. https://doi.org/10.1016/0045-7825(96)01035-3

Martin Wicke, Daniel Ritchie, Bryan M. Klingner, Sebastian Burke, Jonathan R. Shewchuk, and James F. O'Brien. 2010. Dynamic Local Remeshing for Elastoplastic Simulation. *ACM*

Transactions on Graphics 29, 4, Article 49 (jul 2010), 11 pages. https://doi.org/10.1145/1778765.1778786

Wikipedia. 2022. Neo-Hookean solid. Retrieved Nov 18, 2022 from https://en.wikipedia.org/wiki/Neo-Hookean_solid

Joshuah Wolper, Yunuo Chen, Minchen Li, Yu Fang, Ziyin Qu, Jiecong Lu, Meggie Cheng, and Chenfanfu Jiang. 2020. AnisoMPM: Animating Anisotropic Damage Mechanics. *ACM Transactions on Graphics* 39, 4, Article 37 (aug 2020), 16 pages. https://doi.org/10.1145/3386569.3392428

Joshuah Wolper, Yu Fang, Minchen Li, Jiecong Lu, Ming Gao, and Chenfanfu Jiang. 2019. CD-MPM: Continuum Damage Material Point Methods for Dynamic Fracture Animation. *ACM Transactions on Graphics* 38, 4, Article 119 (jul 2019), 15 pages. https://doi.org/10.1145/3306346.3322949

Jian-Ying Wu and Feng-Bo Li. 2015. An improved stable XFEM (Is-XFEM) with a novel enrichment function for the computational modeling of cohesive cracks. *Computer Methods in Applied Mechanics and Engineering* 295 (2015), 77–107. https://doi.org/10.1016/j.cma.2015.06.018

Chuhua Xian, Hongwei Lin, and Shuming Gao. 2009. Automatic generation of coarse bounding cages from dense meshes. In *2009 IEEE International Conference on Shape Modeling and Applications*. Institute of Electrical and Electronics Engineers, 21–27. https://doi.org/10.1109/SMI.2009.5170159

Zangyueyang Xian, Xin Tong, and Tiantian Liu. 2019. A Scalable Galerkin Multigrid Method for Real-Time Simulation of Deformable Objects. *ACM Transactions on Graphics* 38, 6, Article 162 (nov 2019), 13 pages. https://doi.org/10.1145/3355089.3356486

H. Xu and J. Barbič. 2017. 6-DoF Haptic Rendering Using Continuous Collision Detection between Points and Signed Distance Fields. *IEEE Transactions on Haptics* 10, 2 (2017), 151–161. https://doi.org/10.1109/TOH.2016.2613872

Hongyi Xu, Funshing Sin, Yufeng Zhu, and Jernej Barbič. 2015. Nonlinear Material Design Using Principal Stretches. *ACM Transactions on Graphics* 34, 4, Article 75 (jul 2015), 11 pages. https://doi.org/10.1145/2766917

Arash Yavari and Alain Goriely. 2012. Riemann–Cartan Geometry of Nonlinear Dislocation Mechanics. *Archive for Rational Mechanics and Analysis* 205 (2012), 59–118.

Arash Yavari and Jerrold E. Marsden. 2012. Covariantization of nonlinear elasticity. *Zeitschrift für angewandte Mathematik und Physik* 63 (2012), 921–927.

Young June Yoon and Stephen C. Cowin. 2009. The elastic moduli estimation of the solid-water mixture. *International Journal of Solids and Structures* 46, 3 (2009), 527–533. https://doi.org/10.1016/j.ijsolstr.2008.09.010

Linbo Zhang, Tao Cui, and Hui Liu. 2009. A Set of Symmetric Quadrature Rules on Triangles and Tetrahedra. *Journal of Computational Mathematics* 27, 1 (2009), 89–96.

Qi-Zhi Zhu. 2012. On enrichment functions in the extended finite element method. *Internat. J. Numer. Methods Engrg.* 91, 2 (2012), 186–217. https://doi.org/10.1002/nme.4272

Yufeng Zhu, Robert Bridson, and Chen Greif. 2015. Simulating Rigid Body Fracture with Surface Meshes. *ACM Transactions on Graphics* 34, 4, Article 150 (jul 2015), 11 pages. https://doi.org/10.1145/2766942

Goangseup Zi and Ted Belytschko. 2003. New crack-tip elements for XFEM and applications to cohesive cracks. *Internat. J. Numer. Methods Engrg.* 57, 15 (2003), 2221–2240. https://doi.org/10.1002/nme.849

C.B. Zilles and J.K. Salisbury. 1995. A constraint-based god-object method for haptic display. 3, 1 (1995), 146–151. https://doi.org/10.1109/IROS.1995.525876

List of Publications

Journals / Conferences

1. **Avirup Mandal**, Parag Chaudhuri, and Subhasis Chaudhuri. 2023.“Remeshing-Free Graph-Based Finite Element Method for Fracture Simulation ”. *Computer Graphics Forum*, Vol. 42, No. 1, 117-134.

2. **Avirup Mandal**, Parag Chaudhuri, and Subhasis Chaudhuri. 2022.“Simulating Fracture in Anisotropic Materials Containing Impurities ”. *ACM SIGGRAPH Conference on Motion, Interaction and Games (MIG)*. ACM, New York, NY, USA, Article 3, 10 pages.

3. **Avirup Mandal**, Parag Chaudhuri, and Subhasis Chaudhuri. 2022.“Interactive Physics-Based Virtual Sculpting with Haptic Feedback ”. *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)*. ACM, Virtual, USA, Article 9, 20 pages. (Journal version appeared in *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, Vol. 5, No. 1, Article 9, 20 pages).

4. **Avirup Mandal**, Parag Chaudhuri, and Subhasis Chaudhuri. 2023.“Real-time Physics-based Mesh Deformation with Haptic Feedback and Material Anisotropy ”. *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - GRAPP*. INSTICC, SciTePress, 153-161.

Posters

1. **Avirup Mandal**, Parag Chaudhuri, and Subhasis Chaudhuri. 2022. “Artist Controlled Fracture Design Using Impurity Maps ”. *ACM SIGGRAPH 2022 Posters*. ACM, New York, NY, USA, Article 6, 2 pages. [**SRC semi-finalist**]

2. **Avirup Mandal**, Parag Chaudhuri, and Subhasis Chaudhuri. 2021. "Scalable Visual Simulation of Ductile and Brittle Fracture ". *ACM SIGGRAPH 2021 Posters*. ACM, New York, NY, USA, Article 41, 2 pages.

Acknowledgments

Think of love as a state of grace; not the means to anything but the alpha and omega, an end in itself. — Gabriel García Márquez, Love in the Time of Cholera

As I ponder upon what to write in this section, my favourite author, Gabriel García Márquez, comes to mind with his memorable quote. This prompts me to acknowledge how fortunate and grateful I am for the abundance of love, blessings, and support that countless individuals have shown me. Their unwavering encouragement has made it possible for me to write this dissertation.

I express my deepest gratitude to my supervisors, Prof. Parag Chaudhuri and Prof. Subhasis Chaudhuri, for their invaluable guidance and unwavering support throughout my Ph.D. study and related research. I am immensely grateful for the opportunity to interact with both of you and learn from your expertise, which was undoubtedly the highlight of my doctoral study. Having such knowledgeable, empathetic, and humble advisors is a rare blessing, and I will cherish those moments for a lifetime. Additionally, I extend my thanks to Prof. Abhishek Gupta and Prof. Sharat Chandran for serving on my RPC and providing insightful suggestions that helped to enhance my work. I treasure their words of encouragement and wise counsel.

I am indebted beyond measure to the exceptional faculty at IIT Bombay, whose mentoring and guidance made all of this possible. I hold nothing but immense admiration in my heart for you. My sincere gratitude goes out to the teachers at my school who first kindled my interest in research and inspired me to pursue it as a career. I am immensely grateful to the staff of the EE office and the Academic section at IIT Bombay for handling all intricate administrative formalities, allowing us to invest our time in research peacefully. A special thanks to Mr. Santosh Kharat, Ms. Tanvi Shelatkar and Ms. Madhumati Shetty from the EE office. My stay at IIT Bombay would have been incomplete without my amazing friends. I especially thank Sri Prakash, Eswar, Arup, Annoy, Kaustav, Suman, Subhranil, Sartarshi, Pratik, Meet,

Sagar, Previn, Samir, Akankshya, Meera and Karen for their warm friendship. I also give a big shoutout to my colleagues at the Vision and Image Processing Lab — Avik, Amit, Suhas, Fasil, Feroz, Priyadarshini, Anusha, Sanchar, Sayan, Deepshika, Sandika, Dwaipayan, Saurabh, Omkar, Ushasi, Shivali and Neha — whose companionship shaped me into a better individual. I can never repay this debt. Above all, you made IIT Bombay feel like a second home far from my own. I will dearly miss our casual chats on worldly matters, spontaneous outings, and late-night dinners.

A significant portion of my Ph.D. journey was spent during the COVID-19 pandemic, which brought the world to a standstill. I would like to pay tribute to the countless health workers worldwide who risked their lives without hesitation to keep us healthy and worked tirelessly to eradicate the virus. Like everyone else, the past six years have been a roller-coaster ride for me. There were peaks of elation and chasms of despair. During the lowest moments, I found solace and mental strength in many magnificent pieces of world literature. In particular, the lines by Rudyard Kipling — *If you can meet with Triumph and Disaster/And treat those two impostors just the same* — have been particularly inspiring.[1] I am grateful to all the authors whose brilliant works kept me company during my loneliest hours, became a beacon of hope when everything seemed lost, and gave me the courage to stand up again when I was crestfallen.

In conclusion, it would have been impossible for me to complete this journey without the unconditional support and love of my parents, Satyanarayan Mandal and Anuva Mandal. You are the two cornerstones of my life, and I owe everything to you. Thank you for always standing by my side, through thick and thin, and encouraging me to take tough decisions in life.

Although it is a perilous task to acknowledge all the lovely people in my life in such a short space, I want to express my deep gratitude to everyone who has supported me. I apologize to anyone I may have inadvertently left out. I harbour no malice towards them. Rather blame my poor memory for the omissions.

Date: 25-07-2023

Avirup Mandal

Avirup Mandal

Roll No. 163070008

[1] Rudyard Kipling, If