© 2015 by Sofia Leon. All rights reserved.

ADAPTIVE FINITE ELEMENT SIMULATION OF FRACTURE: FROM PLASTIC DEFORMATION TO CRACK PROPAGATION

BY

SOFIA LEON

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Civil Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Doctoral Committee:

Professor Glaucio H. Paulino, Chair
Professor Waldemar Celes, Pontifical Catholic University of Rio de Janeiro
Professor Ahmed Elbanna
James Foulk, III., Sandia National Laboratories
Professor Iwona Jasiuk

Abstract

As engineers and scientists, we have a host of reasons to understand how structural systems fail. We may be able to improve the safety of buildings during natural disaster by designing more fracture resistant connectors, to lengthen the life span on industrial machinery by designing it to sustain very large deformation at high temperatures, or prepare evacuation procedures for populated areas in high seismic zones in the event of rupture in the earth's crust. In order to achieve a better understanding of how any of these structures fail, experimental, theoretical, and computational advances must be made. In this dissertation we will focus on computational simulation by means of the finite element method and will investigate topological and physical aspects of adaptive remeshing for two types of structural systems: quasi-brittle and ductile.

For ductile systems, we are interested in modeling the large deformations that occur before rupture of the material. The deformations can be so large that element distortion can cause lack of numerical convergence. Thus, we present a remeshing and internal state variable mapping technique to enable large deformation modeling and alleviate mesh distortion. We perform detailed studies on the Lie-group interpolation and variational recovery scheme and conclude that the approach results in very limited numerical diffusions and is applicable for modeling systems with significant ductile distortion.

For quasi brittle systems mesh adaptivity is the central theme as it is for the work on ductile systems. We investigate two- and three-dimensional problems on CPU and GPU systems with the main goals of either improving computational efficiency or fidelity of the final solution. We investigate quasi-brittle fracture by means of the inter-element extrinsic cohesive zone model approach in which interface elements capable of separating are adaptively inserted at bulk element facets when and where they are needed throughout the numerical simulation.

The inter-element cohesive zone model approach is known to suffer from mesh bias. Thus, we utilize polygonal element meshes with adaptive splitting to improve the capability of the mesh to represent experimentally obtained fracture patterns. The fact that we utilize the efficient linear polygonal elements and only apply the adaptive element splitting where needed means that we also achieve improved computational efficiency with this approach.

In the last half of the dissertation, we depart from the use of unstructured meshes and focus on the development of hierarchical mesh refinement and coarsening schemes on the structured 4k mesh in two and three dimensions. In three-dimensions, the size of the problem increases so rapidly that mesh adaptivity is critical to enable the simulation of large-scale systems. Thus, we develop the topological and physical aspects of the mesh refinement and coarsening scheme. The scheme is rigorously tested on two benchmark

problems; both of which shows significant speed up over a uniform mesh implementation and demonstrate physically meaningful results.

To achieve greater speed up, the adaptive mesh refinement and coarsening scheme on the 2D 4k mesh is mapped to a GPU architecture. Considerations for the numerical implementation on the massively parallel system are detailed. Further, a study on the impact of the parallelization of the dynamic fracture code is performed on a benchmark problem, and a statistical investigation reveals the validity of the approach. Finally, the benchmark example is extended to such that the speicmen dimensions matches that of the original experimental system. The speedup provided by the GPU allows us to model this large system in a pratical amount of time and ultimately allows us to investigate differences between the commonly used reduced-scale model and the actual experimental scale.

This dissertation concludes with a summary of contribution and comments on potential future research directions. Appendices featuring scripts and codes are also included for the interested reader.

For Wendy

Acknowledgments

I would like to express my gratitude to several individuals who have had a significant impact on my graduate career and on this final dissertation. First and foremost, I would like to thank my academic advisor, Professor Glaucio H. Paulino. Without him, this dissertation, related publications, and fruitful collaborations would not have been possible. I am so grateful for the numerous professional and academic opportunities he has afforded me over the past six years; I would not be where I am today without them. Lastly, I would like to thank him for his support and unwavering belief in me. The lessons I've learned as a result of being his student extend well beyond the academic aspects of this dissertation and will remain with me for many years to come.

I am grateful for the support of the National Science Foundation Graduate Research Fellowship, the Philanthropic Education Organization Scholars Award, the University of Illinois Structural Engineering Fellowship, the University of Illinois College of Engineering Mavis Future Faculty Fellowship, the University of Illinois Graduate College Travel Grant, and the Society of Women Engineers Susan E. Stutz-McDonald scholarship. The academic freedom granted by these programs and agencies resulted in a graduate school research experience that covered a broad range of topics.

Next, I would like to acknowledge the support of my PhD committee. Thank you to Professor Petros Sofronis, from whom I learned the fundamentals in solid and fracture mechanics, and like Professor Paulino, he has taught me lessons that extend well beyond the content of this dissertation. Thank you to Jay Foulk, my mentor at Sandia National Laboratory. I am grateful for our numerous fruitful conversations on topics ranging from the detailed debugging of the Sierra parallel solver to career and life decisions. Thank you to Professor Waldemar Celes with whom I have collaborated throughout all of my graduate studies. I have learned so much from him, not only about computer programming, but also about how to participate in a lasting collaborations. Next, thank you to Professor Ahmed Elbanna. His energy, fresh ideas, and encouragement have been invaluable during my PhD. Last, but certainly not least, I am very grateful to Professor Iwona Jasiuk, who kindly stepped in to participate in the final examination committee. Her comments and input have been very valuable in the final version of this dissertation.

I am fortunate to have participated in several collaborations throughout my graduate studies. Not only have they led to numerous research publications, but also long lasting friendships. Thank you to Ivan Menezes for his academic guidance and his incredible hospitality and generosity during my visits to Rio de Janeiro, by the end I felt like family. I would like to extend my thanks to Anderson Peirera who also made Brazil feel like a second home. I am grateful to Eduardo Nobre Lages who is an excellent mentor and gracious host; I am honored to have worked with him throughout the years. Thank you to Adeildo, Viviane, and the entire Ramos family for their support and generosity in both Brazil and the US. Thank you to Andrei Aldeff

and Rodrigo Espihna whom I worked very closely with on several aspects of this dissertation. Without their expertise and tireless efforts this would certainly have not been possible.

I am grateful for the support and friendship offered by fellow research group members, including Marco Alfano, Luis Arnaldo, Lauren Beghini, Daiane Brisotto, Heng Chi, Junho Chun, Will Colletti, Emily Daniels, Maryam Eidini, Evgueni Filipov, Arun Gain, Ludimar Lima de Aguiar, Chris Liu, Tam Nguyen, Ying Yu, Tomas Zegard, Shelly Zhang, and Tuo Zhao. I would especially like to thank Kyoungsoo Park, Eshan Dave and Daniel Spring with whom I worked closely on aspects of this dissertation as well as other works not shown here.

While not directly reflected in this dissertation, I am very grateful for the opportunity to be involved in extracurricular activities at the University of Illinois; they enriched my graduate student experience and provided invaluable support throughout my studies. I would especially like to thank the fellow graduate women with whom I launched GradSWE at Illinois: Samantha Knoll, Elizabeth Horstman, Danielle Joaquin, Ritu Raman, Jin Kim, Angeli Gamez, Neera Jain and Ashley Gupta. I am honored to have been part of such an amazing support community that advocated for the success of women in STEM. I know that I cannot adequately express my gratitude for the Graduate Women's Group I belonged to for five years. The women in the group helped me realize my abilities, limitations and value. To Marybeth, Meredith, Jackie and Jamie I will be always grateful.

I am lucky to have the support of my friends Evan George, Ashley Danchuk, Danny Huckenmaier, Zach Lloyd, Kelley Chaffin, and Gabbie DeFrancisi. Their friendship and positivity were so motivating to me, especially in times of difficulty during my graduate studies. I would especially like to thank Cameron Talischi. I find myself at a loss for words to describe just how grateful I am for his mentorship and friendship over the years, but maybe I'll be able to express it someday in that screenplay.

Next, I am forever grateful to my family, nothing can ever replace their continual love and support. Thank you to Dad, Carly, Jake, Gramma, Mary Jane, and Milo for making me laugh and reminding how far I've come. I always know that to you it doesn't matter if I study ductile or brittle fracture because your love is totally unconditional. Thank you to my mother, Wendy, who is one of the strongest people I know. No matter what time of day or what the issue I always know that she will be there. I am so proud to be her daughter and it is with great pleasure that I dedicate this dissertation to her.

Finally, I would like to thank my husband, Wylie Ahmed, for his unconditional love, dedication, and belief in me. Graduate school brought us together, he encouraged me to continue to the PhD, and he has been a constant source of motivation since. I cannot imagine my life without him and look forward to any obstacles and opportunities that come our way. We've got this, because we're together!

Contents

List of Tables x

List of Figures xi

Chapter 1 Introduction 1
- 1.1 Motivation to study fracture and failure 1
- 1.2 Modeling approaches 2
  - 1.2.1 Extended/generalized finite element methods - Nodal enrichment 6
  - 1.2.2 Embedded element discontinuities - Element enrichments 6
  - 1.2.3 Peridynamics 7
  - 1.2.4 Discontinuous Galerkin 7
  - 1.2.5 Molecular dynamics 7
  - 1.2.6 Element deletion 8
  - 1.2.7 Mesh free methods 8
  - 1.2.8 Virtual internal bond models 8
  - 1.2.9 Cohesive zone models: Present approach 8
- 1.3 Contributions 11
- 1.4 Document outline 13

Chapter 2 Mapping internal state variables in numerical simulation of ductile failure . . 15
- 2.1 Review of large deformation modeling 15
  - 2.1.1 Large deformation formulations 16
  - 2.1.2 Mapping internal state variables 17
- 2.2 Lie group interpolation and variational recovery 18
  - 2.2.1 Projection scheme 18
  - 2.2.2 Lie group interpolation 20
- 2.3 Computational framework for analysis of Lie-ground interposition scheme 22
- 2.4 Analysis of the Lie-Group Interpolation Scheme 24
  - 2.4.1 Model problem 24
  - 2.4.2 Mapping without remeshing 25
  - 2.4.3 Remeshing and mapping 29

Chapter 3 Towards reduction in mesh bias using adaptive splitting of polygonal finite elements 33
- 3.1 Inter-element cohesive fracture 33
  - 3.1.1 Mathematical formulation of dynamic cohesive fracture 33
  - 3.1.2 Extrinsic cohesive model 35
  - 3.1.3 PPR cohesive constitutive models 36
  - 3.1.4 Implementation of inter-element cohesive fracture 37
  - 3.1.5 Mesh bias in inter-element cohesive fracture 38
- 3.2 Proposed method to reduce mesh bias in inter-element cohesive fracture 39
  - 3.2.1 Polygonal element mesh generation 39
  - 3.2.2 Shape functions for polygonal elements 42

3.2.3 Adaptive splitting through polygonal elements . . . 42
3.3 Geometric studies on crack representation . . . 45
3.3.1 Crack length studies . . . 47
3.3.2 Crack angle studies . . . 50
3.3.3 Hausdorff distance studies . . . 51
3.3.4 Summary of geometric studies . . . 53
3.4 Numerical Investigations . . . 53
3.4.1 Compact Compression Specimen . . . 56
3.4.2 Simulation results . . . 60

Chapter 4 Adaptive refinement and coarsening on structured 3D meshes . . . 65
4.1 4k structured meshing in three-dimensions . . . 65
4.2 Implementation of adaptive cohesive fracture with the TopS data structure . . . 66
4.2.1 Finite element mesh representation using TopS . . . 66
4.2.2 Automatic crack tip tracking . . . 70
4.2.3 Numerical implementation . . . 73
4.3 Adaptive insertion of cohesive elements in 3D . . . 73
4.4 Adaptive mesh refinement . . . 76
4.4.1 Geometric aspects of adaptive mesh refinement . . . 76
4.4.2 Implementation of adaptive mesh refinement with the TopS data structure . . . 77
4.4.3 Physical aspects of adaptive mesh refinement . . . 78
4.5 Adaptive mesh coarsening . . . 80
4.5.1 Geometric aspects of adaptive mesh coarsening . . . 80
4.5.2 Implementation of adaptive mesh coarsening with the TopS data structure . . . 83
4.5.3 Physical aspects of adaptive mesh coarsening . . . 83
4.6 Numerical simulations . . . 86
4.6.1 Post-processing and visualization . . . 86
4.6.2 Confined crack . . . 86
4.6.3 Three point bend notched beam . . . 97

Chapter 5 Massively parallel adaptive mesh refinement and coarsening for dynamic fracture simulations . . . 101
5.1 GPU architecture . . . 102
5.2 Review of related work . . . 103
5.3 Adaptive mesh modification on Graphical Processing Units . . . 104
5.3.1 Data structure for 4k adaptive finite element mesh representation . . . 104
5.3.2 Node and element calculations on GPU . . . 107
5.3.3 Adaptive insertion of cohesive elements . . . 108
5.3.4 Adaptive mesh refinement and coarsening . . . 108
5.4 Numerical investigations . . . 110
5.4.1 Reduced-scale micro-branching specimen . . . 112
5.4.2 Full scale micro-branching specimen . . . 118

Chapter 6 Conclusion . . . 122
6.1 Summary of contributions . . . 122
6.2 Future research directions . . . 123
6.2.1 Future investigations and applications of Lie-group interpolation and variational recovery scheme . . . 123
6.2.2 Investigation of activation criteria for extrinsic cohesive elements . . . 124
6.2.3 Automatic remeshing around crack tips . . . 126
6.2.4 Polyhedra finite elements for dynamic fracture simulation . . . 129
6.2.5 Adaptive time stepping for explicit dynamic fracture . . . 129
6.2.6 Three dimensional adaptive mesh refinement and coarsening on graphical processing units . . . 130

Chapter 7 Bibliography **132**

Appendix A Scripted procedures for remeshing and mapping internal state variables . **146**
A.1 Python script for mapping internal state variables without remeshing 146
A.2 Python script for mapping internal state variables with remeshing 153
A.3 Python script to diff two Exodus II files 160

Appendix B Callback functions for adaptive mesh refinement and coarsening **162**
B.1 Callback functions for adaptive refinement of 3D 4k mesh 162
B.2 Callback functions for adaptive coarsening of 3D 4k mesh 165
B.2.1 CallBack_CanCollapseNode 165
B.2.2 Callback_MergeElements 166
B.2.3 Callback_RemoveNode 166
B.2.4 getPatchStrainError 166

List of Tables

3.1 Cases for geometric study comparison . . . 45
3.2 Summary of geometric studies meshes with $\lambda \approx 1/80$ . . . 55
3.3 Summary of mesh statistics for the CCS problem . . . 59

4.1 Summary of computational resource usage in confined crack problem . . . 92

5.1 Comparison of final quantities between Uniform, AMR and AMR+C simulations . . . 114
5.2 Variation in crack tip velocity, energy released, and occurrence of branching for 20 simulations of each the AMR and AMR+C enabled meshes . . . 116
5.3 Comparison of wall time of the reduced scale micro-branching problem on different platforms (The speed up factor is shown with respect to the no adaptivity case on the serial CPU) . . . 117

List of Figures

1.1 Fracture and failure at different scales (a) Earthquake fault in Landers, California and 3D numerical model (from [1]) (b-top) I35-W Mississippi Bridge failure in 2007 (photo available at http://www.pbs.org/wgbh/nova/sciencenow/0304/04-whyt-09.html) (b-bottom) bridge failure was due to fracture of gusset plates at the bolt holes (from National Transportation Safety Board) (c) x-ray of fractured bone and topographic imaging failure assessment of the human spine under compressive strain (from [2]) 3
1.2 SEM micrographs showing flaws present in glass rods (from University of Cambridge Dissemination of IT for the Promotion of Materials Science (DoITPoMS)) 4
1.3 Cavity formation at particles of amorphous silica in matrix of Cu-Si alloy with 20% elongation (from [3]) 5
1.4 SEM fractograph of cleavage fracture in chromium hard plated steel (from the University of Plymouth Interactive Failure and Fracture Mechanics Resources) 5
1.5 Schematic of the cohesive zone model approach: the cohesive zone ahead of the macro crack tip includes voids and micro-cracks which is idealized by a traction separation relationship. Ahead of the macro crack tip the traction increases and complete separation is present at the macro crack tip then decreases through length of the cohesive zone. Experimental fracture specimen image from http://makine.dogus.edu.tr/en/?page_id=20 and SEM image of fracture surface from University of Cambridge Dissemination of IT for the Promotion of Materials Science (DoITPoMS) available at http://www.doitpoms.ac.uk/tlplib/mechanical-testing/results1.php 10

2.1 Location of known rotations, $\mathbf{R}(-1)$ and $\mathbf{R}(1)$, and those to be interpolated/extrapolated $\mathbf{R}(0)$ and $\mathbf{R}(2)$ 20
2.2 Flowchart of Python script for analysis of large deformation process with mapping internal state variables 23
2.3 (a) coarse and (b) fine meshes of model problem for numerical studies 25
2.4 Maximum equivalent plastic strain after each mapping procedure (a) without an equilibrium step after the mapping procedure and (b) with an equilibrium step after the mapping procedure 27
2.5 (a) Residual after equilibrium and mapping steps (b) Magnitude of residual plotted on deformed shape of model problem after first mapping procedure, the green locations indicate areas of higher residuals, blue are lowest 27
2.6 Dissipation in internal variable (equivalent plastic strain) with consecutive remaps indicates a problem in one of the analysis modules 28
2.7 Equivalent plastic strain at one integration point per element at the end of the analysis, $t = 0.25$, for (a) single interval and (b) 100 intervals 29
2.8 Load displacement curve for (a) coarse mesh and (b) fine mesh with no remeshing after each interval 30
2.9 Construction of new mesh on model problem (a) Undeformed configuration is meshed with 10 elements across the bottom ligament (b) Deformed configuration to be remeshed with 10 elements across the bottom ligament (c) Zoom in of deformed source mesh (d) Deformed domain is remeshed 31
2.10 Load displacement curve for (a) coarse mesh and (b) fine mesh with remeshing after each interval 31

3.1 Arbitrary domain with applied boundary conditions . . . 34
3.2 Schematic of insertion of extrinsic cohesive elements using a stress-based approach (a) Nodes with principle stress greater than 90% of the cohesive strength are flagged - shown in red (b) Normal and tangential tractions are computed at the facets adjacent to the flagged nodes - shown in dashed blue. If the averaged normal or tangential stress is greater than the respective cohesive strength, then a cohesive element is inserted along the facet . . . 35
3.3 Schematic of a cohesive element being inserted at the new facets of split polygonal element. (a) The element ahead of the crack tip meets the criteria to be split, indicated by the dashed line. (b) The polygonal element is split and a cohesive element is inserted at the new facets; the zoomed in region shows a schematic of the cohesive tractions provided by the PPR model 36
3.4 Traction-separation relations for (a) extrinsic model, normal opening ($\phi_n = 100N/m$, $\sigma_{max} = 40MPa$, $\alpha = 5.0$), (b) extrinsic model, tangential opening ($\phi_t = 200N/m$, $\tau_{max} = 30MPa$, $\beta = 1.5$) . . . 37
3.5 Traction-separation relations with linear unloading at 20% for (a) extrinsic model, normal opening ($\phi_n = 100N/m$, $\sigma_{max} = 40MPa$, $\alpha = 3.0$), (b) extrinsic model, tangential opening ($\phi_t = 200N/m$, $\tau_{max} = 30MPa$, $\beta = 5.0$) . . . 38
3.6 Flowchart of extrinsic, cohesive, dynamic fracture code . . . 39
3.7 4k mesh (a) without and (b) nodal perturbation . . . 40
3.8 Potential crack path directions on a 4k mesh where the solid nodes have eight potential crack directions and the white nodes have only four directions. (a) Bold edges can be swapped; (b) After the edge-swap operator is applied the dashed edges become available crack directions . 40
3.9 Triangular areas used to define α_i . . . 42
3.10 Schematic of a crack in a polygonal mesh, where crack faces are illustrated with solid black lines, the crack tip is indicated by a black circle, and potential crack paths are shown as dashed lines. (a) Potential crack directions on plain CVT mesh (b) Potential crack directions on CVT mesh with adaptive splitting employed (c) Potential crack directions on plain near-MPS mesh (d) Potential crack directions on near-MPS mesh with adaptive splitting employed . . . 43
3.11 Schematic of potential new elements that would result from splitting element between node 1 and (a) node 3, (b) node 4 and (c) node 5. The configuration shown in (b), where the element is split with node 4, minimizes the difference between the areas, A1 and A2, of the resulting new elements; once an element is split, the resulting two elements cannot be split again . . . 44
3.12 4k mesh (a) without nodal perturbation and (b) with nodal perturbation . . . 46
3.13 Potential crack path directions on a 4k mesh where the solid nodes have eight potential crack directions and the white nodes have only four directions. (a) Bold edges can be swapped. (b) After the edge-swap operator is applied the dashed edges become available crack directions . 46
3.14 The shortest distance between the start point (0, 0) and end point ($\cos(71.6°)$, $\sin(71.6°)$), measured along the finite element edges vs. the euclidean distance for a polygonal mesh with and without restricted element splitting. (a) Full domain, (b) zoom in of region inside the dotted, grey box in (a). The circular domain shown here is for illustrative purposes only, the studies were conducted on rectangular domains with an inscribed circular boundary, as described in the text . . . 48
3.15 Comparison of mesh deviation, η, for meshes with $\lambda \approx 1/80$. Deviations of (a) 4k meshes with and without edge swap, (b) 4k meshes nodal perturbation factors of 0 and 0.3 with and without edge swap, (c) CVT meshes without element splitting, with restricted element splitting and with unrestricted element splitting, (d) random polygonal meshes without element splitting, with restricted element splitting and with unrestricted element splitting . . . 49
3.16 Comparison of an arbitrary angle and geometric angle, $\theta = 34°$, for (a) unperturbed 4k mesh with and without edge swapping, (b) perturbed 4k meshes with and without edge swapping 50
3.17 Absolute value of difference between geometric angle and target angle for radial paths from from 0° to 359° at 1° increments for (a) 4k without nodal perturbation, (b) 4k with nodal perturbation; results for the perturbed mesh are averaged from from 100 meshes . . . 51
3.18 Absolute value of difference between geometric angle and target angle for radial paths from from 0° to 359° at 1° increments for (a) CVT polygonal mesh, (b) Random polygonal mesh; results are averaged from from 100 meshes . . . 52

3.19 Histograms of Hausdorff distances for radial paths from 0° to 359° at 1° increments for (a) unperturbed 4k mesh with and without edge swapping, (b) perturbed 4k meshes with and without edge swapping and (c) CVT polygonal mesh with and without element splitting, (d) random polygonal mesh with and without element splitting; results for the 4k discretization are obtained from one mesh, while the results from 100 meshes are averaged for each the perturbed 4k and polygonal discretizations . . . 54
3.20 Schematic of Compact Compression Specimen (CCS) problem, the expected crack pattern from [4] is shown in blue . . . 56
3.21 Random polygonal element mesh of CCS domain (a) full domain, (b) zoom in of red region indicated in subfigure (a), (c) zoom in of blue region indicated in subfigure (b), (d) zoom in of green region indicated in subfigure (c) . . . 58
3.22 CVT polygonal element mesh of CCS domain (a) full domain, (b) zoom in of red region indicated in subfigure (a) . . . 59
3.23 Crack patterns of CCS problem with random polygonal element mesh at (a) 64.02 μsec and (b) (a) 65.31 μsec . . . 61
3.24 Fracture pattern on CCS mesh (zoomed in near the notch tip) with the CVT polygonal element mesh without element splitting . . . 62
3.25 Fracture pattern on CCS mesh (zoomed in near the notch tip) with the CVT polygonal element mesh with restricted element splitting . . . 63
3.26 Fracture pattern on CCS mesh (zoomed in near the notch tip) with the CVT polygonal element mesh with unrestricted element splitting . . . 64

4.1 4k mesh constructed by subdividing a hexahedron into 24 tetrahedra (a) each of the 6 faces of the hexahedron is divided into 4 tetrahedra (b) exploded view of each face of the hexahedron (c) exploded view of one face of the hexahedron contains 4 tetrahedra . . . 66
4.2 Levels of refinement on a unit cube, each increase in level doubles the number of linear tetrahedral elements from the previous level. (a) Level 0 - 6 elements, 8 nodes, (b) Level 1 - 12 elements, 9 nodes, (c) Level 2 - 24 elements, 15 nodes, (d) Level 3 - 48 elements, 27 nodes, (e) Level 4 - 96 elements 35 nodes, (f) Level 5 - 192 elements 71 nodes (Note that on the exterior level 0 looks identical to level 1, and level 3 looks identical to level 4 because the splitting occurs on the interior edges as illustrated in Figure 4.3) . . . 67
4.3 Subdivision of hexahedron from 6 to 12 tetrahedra (a) the six element hexahedron is subdivided along the interior edge, show as a dashed line (b) a single tetrahedron is isolated and the edge upon which it will be split is shown by the dashed line (c) new tetrahedra resulting from splitting . . . 68
4.4 Rate of increasing in mesh size for a 2D vs. 3D mesh when all elements in the mesh are refined 68
4.5 Schematic of the client-server approach between the TopS API and the application . . . 69
4.6 3D view of crack front, cohesive elements are shown in blue and red, where the red elements are crack tip elements; the crack tip nodes, un-duplicated nodes on cohesive elements are indicated in red . . . 70
4.7 Active and inactive crack tip regions identified by the ToPS refinement manager (a) All crack tip nodes have an active spherical region of refinement associated with them and any region outside the refinement regions is inactive and coarsened as much as possible. Regions around cohesive elements will never be fully coarsened because cohesive elements stay at the most refined level and bulk elements around them must transition from fine to coarse. (b) Resulting mesh with refinement regions only at the crack tips . . . 71
4.8 Nodes are flagged (shown in red) when the principle stress exceeds 90% of the cohesive strength of the material, then adjacent edges (shown in dashed blue) are visited and stresses along them are computed to determine if a cohesive element should be inserted . . . 75
4.9 Insertion of a cohesive element in a 3D mesh (a) cohesive element is inserted at the purple facet between the blue and red elements, (b) mesh is blow up and entities separated . . . 76
4.10 Splitting along longest edge in a patch . . . 77

4.11 Mesh refined only in a select region to level 11. The elements that a fully refined to level 11 are shown in red, the coarse elements at level 0 are shown in grey, and the transition region of partially refined elements between levels 0 and 11 are shown in blue. (a) All levels (b) Transition region removed 78
4.12 Flowchart of procedures to perform adaptive mesh refinement 79
4.13 Schematic of refinement showing process, (a) two grey elements are refined by (b) insertion of new node, (c) grey (parent) elements are deleted new (child) elements are inserted 80
4.14 Case 1 for mesh coarsening: a patch of elements on a boundary is coarsened from (a) 4 elements to (b) 2 elements by removing the grey node. 2 pairs of red and blue elements are coarsened to 2 green elements 81
4.15 Case 2 for mesh coarsening: a diamond shaped patch of (a) 8 elements is coarsened to (b) 4 elements by removing the internal grey node. 4 pairs of red and blue elements are coarsened to 4 green elements 81
4.16 Case 3 for mesh coarsening: a hexahedron of (a) 12 elements is coarsened to (b) 6 elements by removing the internal grey node. 6 pairs of red and blue elements are coarsened to 6 green elements 82
4.17 Case 4 for mesh coarsening: a diamond shaped patch of (a) 16 elements is coarsened to (b) 8 elements by removing the internal grey node. 8 pairs of red and blue elements are coarsened to 8 green elements 82
4.18 Flowchart of procedures to perform adaptive mesh coarsening 84
4.19 Mesh coarsening results in removal of nodes and replacement of elements. (a) Original, refined patch of 4 elements, e1, e2, e3, e4 contains 6 nodes (b) Coarsened patch contains 2 elements, E1 and E2, and 5 nodes 85
4.20 3D printed tetrahedra: red - level 2, blue - level 3, orange - level 4, green - level 5, yellow - level 6. The black hexahedron is equivalent to 24 red, 48 blue, 96 orange, 192 green, or 384 yellow tetrahedra. (a) The black hexahedron with top face removed, the white dashed line indicates the space where one red tetrahedra would fit. The hexahedron is shown with 2 blue tetrahedra, 4 orange tetrahedra, and 4 green tetrahedra. (b) A red tetrahedra (level 2) next to pairs of yellow tetrahedra (level 6). (c) The yellow tetrahedra are oriented, and the left "sides" are pictured; the blue tetrahedra are also oriented, as evident from their position in (a) 87
4.21 Schematic of confined notched beam subjected to strain loading 88
4.22 Initial meshes for the 0.1mm thick notched beam specimen. (a) Coarse grid which is the basis for each case investigated (b) Fully refined case where all elements of the coarse grid are refined to Level 11 (c) Zoom in around crack tip of fully refined case (d) AMR and AMR+C case where elements in the region of the notch tip are refined to Level 11 (e) Zoom in around crack tip of AMR and AMR+C case 90
4.23 Finite element mesh of the 0.1 mm thick specimen at 210 nanoseconds (7000 steps) on the (a) AMR with coarse initial mesh (b) AMR+C with initial mesh of resolution 50 μm and coarsening tolerance of 0.01 91
4.24 Comparison of velocity on 0.1mm thick specimen for various levels of coarse level refinement . 91
4.25 Crack tip position versus simulation time for the 0.1 mm thick specimen 93
4.26 Energy evolution for the confined crack problem on the 0.1 mm thick specimen with an initial coarse mesh resolution of 12.5 μm. The coarsening tolerance for the AMR+C case is 0.001. (Add figure of finer initial mesh with AMR and AMR+C) 93
4.27 Stress σ_{yy} at 15 nanoseconds (500 steps) on the 0.1 mm thick specimen with a (a) uniform mesh (b) AMR with fine mesh and (c) AMR+C with initial mesh of resolution 12.5 μm and a coarsening tolerance of 0.01, shown on the x-y face of 3D meshes 94
4.28 Crack tip position versus wall time for the 0.1 mm thick specimen with AMR and AMR+C with the loosest coarsening tolerance of 0.01. (Just for demonstrative purposes) 95
4.29 Model size versus time for the 0.1 mm thick specimen with (a) uniform refinement (b) AMR (c) AMR+C with coarsening tolerance of 0.01 (d) AMR+C with coarsening tolerance of 0.0001 96
4.30 Three-point-bend beam (TPB) specimen schematic with expected fracture pattern shown in blue 98

4.31 Displacement field at 0.0072 sec on coarse initial mesh with (a) AMR (b) AMR+C and at 0.0071 sec on fine initial mesh with (c) AMR (d) AMR+C 98
4.32 Damage of cohesive elements for the TPB specimen (a) fully open elements and (c) all cohesive elements of coarse initial mesh and (c) fully open elements and (d) all cohesive elements of fine initial mesh 99
4.33 Insertion of cohesive elements versus time for the AMR and AMR+C of the TPB specimen with the coarse far field mesh and fine far field mesh 100

5.1 Schematic of grid with 2×2 blocks of $3 \times 2 \times 2$ threads each 103
5.2 Schematic of GPU data structure for adaptive 4k mesh (a) progression of mesh refinement and element labeling, (b) node numbering on refined mesh, (c) facets labels indicating order of refinement on refined mesh, (d) node table showing node ids, coordinates, and adjacent element, (e) element table showing element id, nodal connectivity, adjacent elements, reference level, level of refinement, and facet labels 106
5.3 (a) Refinement of elements 1, 2 and 3 from level 0 (white) to level 2 (dark grey), (b) Binary tree representation of refinement 107
5.4 Marked opposite elements (a) elements with at least one node inside the refinement region are marked, (b) elements adjacent a marked element's hypotenuse are also marked 109
5.5 4k refinement scheme (a) Mesh is initially refined around the notch tip. (b) Cohesive elements are inserted along facets of fully refined elements, new crack tips are identified and new refinement regions associated with each crack tip are created. Elements to be refined for all crack tips are collected simultaneously, as opposed to one crack tip at a time (c) Elements within the refinement region are marked (black 'x') and elements adjacent to the hypotenuse of a marked element are marked (grey 'x') (d) Marked elements are refined to the full level and transition region refined to ensure element compatibility 111
5.6 Reduced scale micro-branching problem geometry, loading conditions, and material properties 112
5.7 Final crack pattern for the reduced scale micro-branching problem for (a) uniform mesh (b) AMR enabled mesh (c) AMR+C enabled mesh. Cohesive elements opened greater than 10% of the normal or tangential critical opening distance are shown in blue, other cohesive elements are shown in red 113
5.8 Details of crack branching including kink in the main crack, crack branches, and secondary branches 115
5.9 A coarse mesh patch of four grey elements is refined to a patch of 8 colored elements; the order in which the color elements contributions are added to the node will vary from one simulation to the the next 115
5.10 Histogram of branch lengths over 20 simulations for the (a) AMR enabled meshes with an open crack tolerance of 75% of critical normal opening, (b) AMR+C enabled meshes with an open crack tolerance of 75% of critical normal opening, (c) AMR enabled meshes with an open crack tolerance of 10% of critical normal opening and (d) AMR+C enabled meshes with an open crack tolerance of 10% of critical normal opening 117
5.11 Geometry of full scale micro-branching problem 118
5.12 Final fracture patterns for full scale micro-branching problem with an externally applied strain of (a) 0.003, (b) 0.004, and (c) 0.005 120
5.13 Detailed view of fracture pattern for the full scale micro-branching problem with an externally applied strain of 0.003 121

6.1 Stress field after projection from integration points to nodes of Tet10 elements using (a) quadratic projection (b) linear projection (c) composite linear projection 124
6.2 Experimental load versus displacement curve of 304L butt weld 125
6.3 Experimental load versus displacement curve of 304L butt weld with simulated results overlaid. The voids in the material impact the overall behavior, however, current modeling attempts have been unsuccessful due to large distortions in the finite elements 125

6.4 Polygonal remeshing around the constraint of an arbitrary crack pattern shown in red (a) 4k based triangular mesh (b) Delaunay based triangular mesh (c) 4k based quadrilateral mesh (d) Delaunay based quadrilateral mesh 128
6.5 Temporal adaptivity illustrated on a one-dimensional spatial mesh (a) a uniform time step is used throughout the domain (b) a smaller time step is used in areas of mesh refinement (Δt_B) and a larger time step is used in the other areas (Δt_A). Figure adapted from [5] 130

Chapter 1

Introduction

As engineers and scientists, we have a host of reasons to understand how and why structural systems fail. The mechanism by which failure occurs can give us greater insight into the overall behavior of the material. Furthermore, if we can better understand how structures fail, then we may be able to design a safer and stronger solution.

1.1 Motivation to study fracture and failure

In order to achieve a better understanding of how materials and systems fail, experimental, theoretical, and computational advances must be made. It is the aim of this work to contribute to the understanding of fracture and failure through computational simulation.

Motivation for the numerical simulation of fracture and failure is present many fields. Figure 1.1 demonstrates failure at different scales, from tens of kilometers in seismic fault rupture to micron scales in trabecular bone subjected to strain loading. From a traditional structural engineering perspective, numerical simulation of fracture can lead a better understanding of the failure mechanism and thus to improved design protocols. For example, numerous studies on collapse of the I-35W Mississippi River Bridge in 2007 (Figure 1.1(a)) revealed that the catastrophic collapse was due to fracture of the gusset plates connecting the floor members to the main truss frame [6, 7]. Study of progressive collapse, such as those of the World Trade Center, characterize the energy absorption of high rise structures such that collapse under extreme loading could be interfered [8]. Resistance of high rise buildings to low velocity impact is also of interest; for example, numerical and material models have been developed to numerically investigate the resistance of architectural glass to impact from debris during a severe wind storm [9]. Structural collapse has impacts well beyond the structural system itself; in the case of the bridge collapse for example, traffic flow patterns in Minneapolis were disrupted due to the abrupt rerouting of 140,000 daily vehicle trips [10].

Related to structural engineering is the study of rupture of the earth's crust during seismic events (Figure 1.1(b)). The scale of the problem may be much to large for experimental simulation, thus numerical models provide a means by which an investigation into the failure mechanism is feasible [1, 11–14]. Of course, the structural response to earthquake events is critical, and models to simulate failure of buildings or fracture of components have been developed [15, 16]. Furthermore, fracture of roadways and networks during earthquakes or environmental effects can have a large negative impact on travelers, thus studies of environmentally friendly and fracture resistant pavement materials have been underway be several researchers. Experimental procedures to classify the fracture resistance of asphalt pavements [17–19] have also been investigated numerically [20–22], with new models being developed to capture the complexities associated with the viscoelastic materials.

Much of our understanding of fatigue failure has come from studies on aerospace structures, such as that

of Southwest Flight 812 in 2011 [23]. A section of the fuselage skin fractured at the lap joint causing the vessel to open during flight. Clearly, investigations to better understand the behavior of composite materials that comprise a plane's outer shell are warranted. Several researchers have worked to improve the design of such thin walled structures under ultimate loading conditions both numerically and experimentally [24–28].

From a mechanical perspective, understanding the response of materials under extreme loading conditions, such as high stress [29] or velocity [30] may enable us to lengthen the life span of industrial machinery. New experimental techniques, such as the one proposed in [31] give us greater insight into the micro-mechanics of the failure process of common materials and composites. When classical materials become insufficient for a certain purpose, design of new materials becomes relevant. When the goal is to design a new material that will be less likely to fail, it is especially important to understand the mechanisms by which it could break. The need for structural materials with both high toughness and high strength has been an big area of interest even though these two properties are often naturally mutually exclusive [32]. A new class of engineering materials called bulk metallic glasses [33] have been designed to achieve high strength and ductility, making it one of the toughest known materials [34].

For quite some time, there has been interest in designing bio-inspired materials or even artificial biological materials. Over 30 years ago, researchers investigated the possibility to design artificial skin in which the fracture properties are of utmost importance [35]. More recently, bio-inspired adhesive surfaces were proposed; the material achieves it enhanced adhesion due its ability to trap cracks in the contact regimes [36]. Bio materials that spontaneously bond to living bone can be used as bone substitute, however they typically have low fracture toughness compared to human bone, thus researches work to developed composite material with bone-like microstructure through chemical treatment of metals or through polymerization of bioactive silica [37]. As bone is a quite complex material, many studies have focused on quantifying its mechanical properties for use in new material design or numerical models [2, 38–41] (Figure 1.1(c)). At a larger scale, researchers have also numerically examined the fracture pattern and energy evolution of bone subjected to extreme loading of a firearm [42]. While the connection to fracture simulation may not be immediately obvious, a medical doctor may need to understand how a blood clot separates in terms of mechanical properties to prevent a severe embolism. Any change of mechanical properties to the vascular wall, for example due to calcification, are shown to have an impact on the fracture and failure of coronary stents [43].

1.2 Modeling approaches

This work develops enabling technologies for the predictive simulation of structural failure, and focuses mainly on dynamic fracture of brittle systems. Before reviewing a number of approaches to model such behavior, we will first classify the main types of material failure. It is useful to consider three categories of materials: brittle, semi-brittle and ductile.

In a brittle solid (e.g. silicate glass) the relatively immobile flaws are not particularly large, they occur primarily on the surface, and vary widely in size, shape and location, as shown in Figure 1.2. Alternatively, flaws in brittle solids may arise during the preparation of the specimen or due to exposure to thermal changes [44].

Unlike highly brittle solids, semi-brittle solids feature limited plastic flow during nucleation before the crack, which resembles that of a brittle material, propagates. The strength of these materials is governed more so by the yield properties than by the initial flaw distribution. Inelastic material deformation, such as plasticity in metals, relaxes the stress at the crack tip [44].

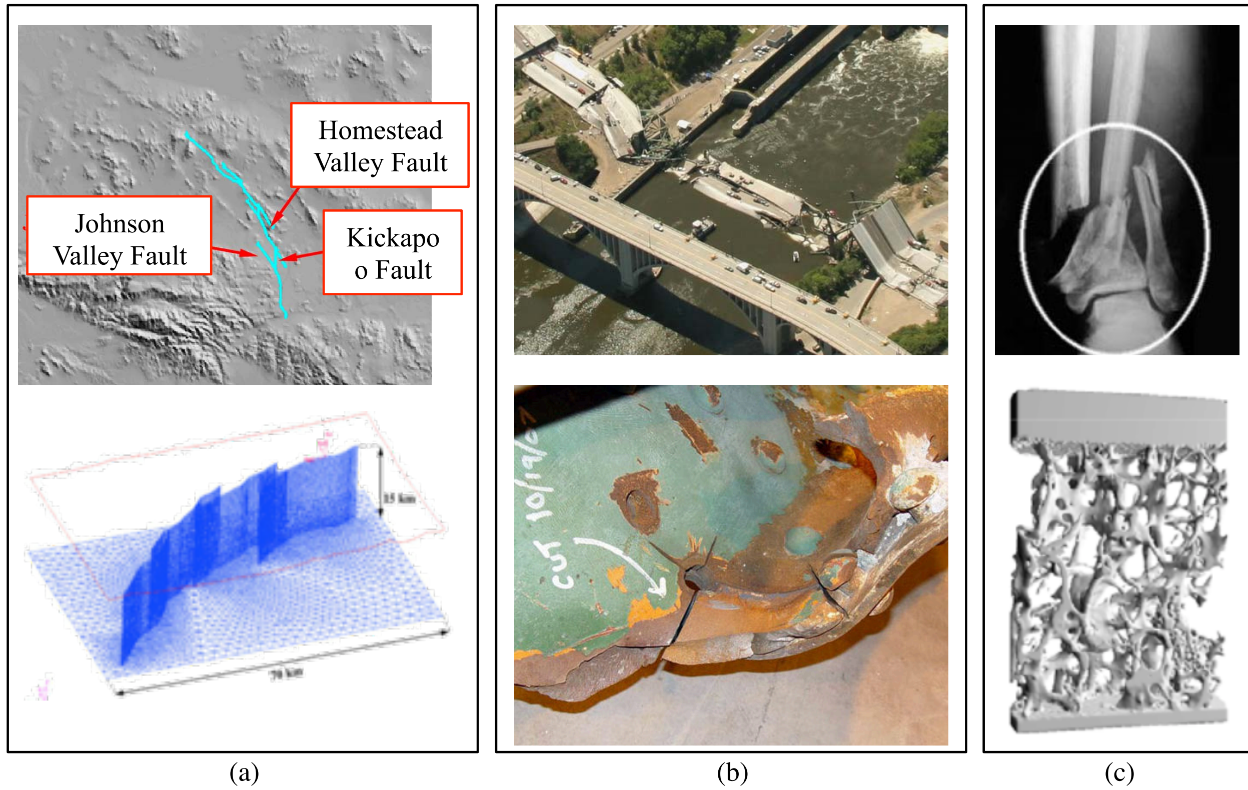


Figure 1.1: Fracture and failure at different scales (a) Earthquake fault in Landers, California and 3D numerical model (from [1]) (b-top) I35-W Mississippi Bridge failure in 2007 (photo available at http://www.pbs.org/wgbh/nova/sciencenow/0304/04-whyt-09.html) (b-bottom) bridge failure was due to fracture of gusset plates at the bolt holes (from National Transportation Safety Board) (c) x-ray of fractured bone and topographic imaging failure assessment of the human spine under compressive strain (from [2])

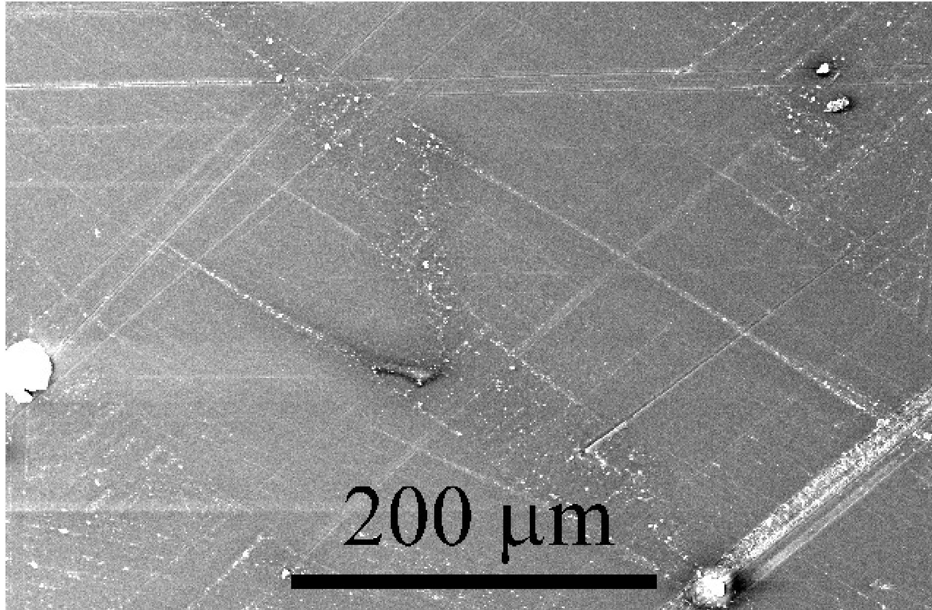


Figure 1.2: SEM micrographs showing flaws present in glass rods (from University of Cambridge Dissemination of IT for the Promotion of Materials Science (DoITPoMS) available at http://www.doitpoms.ac.uk/tlplib/BD5/results.php)

In ductile solids, the role of plasticity is the dominating factor in crack nucleation. Unlike semi-brittle solids, dislocations are flexible and can move on and between crystallographic planes. On a tensile specimen this plastic instability would be realized as a region of reduced cross-section, i.e. neck. In a pure crystal, no crack nuclei are formed and instead the atoms slide apart. This does not actually occur in real materials because of the presence of micro-structural defects, which cause cavities to nucleate [3] as shown in Figure 1.3. The initial stage of void nucleation is often the critical step. In order for this to occur, sufficient stress must be applied to break the interfacial bonds between the particle and the matrix. Once voids are nucleated, plastic strain and hydrostatic stress cause the voids to grow then fracture propagates by coalescence with adjacent voids. Another type of ductile failure is cleavage fracture in which a crack rapidly propagates along a crystallographic plane. The propagation of the crack may behave as though it were brittle, but it is preceded by ductile crack growth. Cleavage fracture occurs along planes with the lowest packing density since fewer bonds need to be broken. The crack changes direction each time it crosses a grain boundary as it seeks the most favorably oriented plane in each grain. Figure 1.4 shows an SEM fractograph of cleavage fracture in chromium hard plated steel where each facet corresponds to a single grain.

The characterization of a material as being brittle or ductile may be a function of several variables, e.g. temperature, loading rate, and specimen geometry. It is well known that at low temperatures certain materials will fail in a brittle way, at high temperatures the failure is ductile, and in between is the ductile to brittle transition region. The abruptness and location of this transition varies with different material microstructures. In some steels, for example, the transition temperature is around $0°C$ making catastrophic brittle failure more feasible in colder climates. The specimen geometry can also affect the crack initiation and propagation mechanism. A plane strain type constraint with hydrostatic stress conditions can transition to a less confined plane stress configuration [45]. The apparent plane stress conditions will give an artificially low transition temperature or artificially high fracture toughness. The effect of geometry is also evident through the phenomenon of crack tunneling. This was observed in a experimentally and in a computational

Figure 1.3: Cavity formation at particles of amorphous silica in matrix of Cu-Si alloy with 20% elongation (from [3])

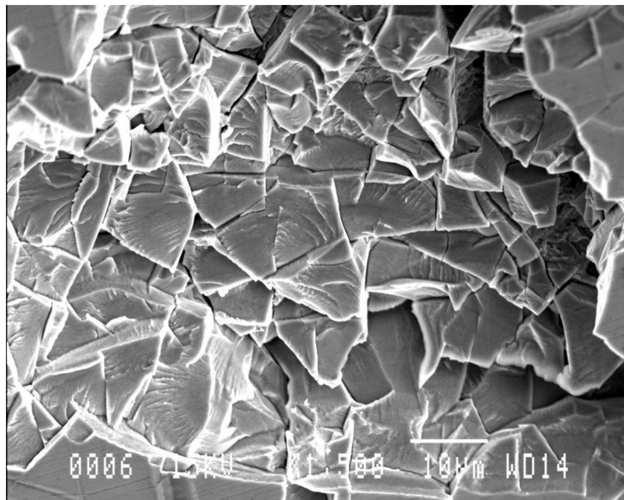


Figure 1.4: SEM fractograph of cleavage fracture in chromium hard plated steel (from the University of Plymouth Interactive Failure and Fracture Mechanics Resources available at http://www.tech.plymouth.ac.uk/sme/Interactive_Resources/tutorials/FailureAnalysis/Fractography)

model of a three pint bend specimen with a sharp initial notch [46]. Substantial plastic deformation at crack initiation resulted in large gradients of the opening displacement along the initial crack front; the greatest opening being at the center and diminishing outwards. In this specimen type, a steady state crack profile is expected after some crack propagation, but this is a function of specimen geometry.

Next we will focus back on brittle failure behavior and discuss some modeling technologies for simulation of dynamic fracture. Several methods exist to model crack propagation [47] and we will review the more common approaches here.

1.2.1 Extended/generalized finite element methods - Nodal enrichment

The eXtended finite element method (XFEM) or generalized finite element (GFEM) is utilized for crack propagation problems to avoid the issues of remeshing necessary with standard cohesive zone approaches [48–59]. The key idea of the method is that displacement field incorporates a discontinuity as an additional term in the finite element displacement approximation via discontinuous partitions of unity. The enrichment is inserted when some criteria is met, loss of hyperbolicity for example [48]. A number of variations on the types of discontinuities inserted are present in the literature. In [60], for example, a step function enrichment was utilized in elements that are completely cracked. Later the method was generalized such the the discontinuity could be arbitrary and could include discontinuities in the derivatives of the displacement [48]. These methods can have weaknesses in representing many cracks, especially micro-branching, and interaction between cracks. In a comparative paper with embedded element discontinuities, see Section 1.2.2, the authors concluded that while both methods have similar levels of accuracy, extended finite element approaches were 1.1 to 2.5 more expensive for a single crack and cost that increases linearly as more cracks appear in the domain. This additional const is due to the non-condensable degrees of freedom that are added to the system.

1.2.2 Embedded element discontinuities - Element enrichments

Embedded discontinuity methods have been widely used to model strain localization and crack propagation. In the case of strong discontinuities, the displacement field is embedded with a discontinuity, whereas in the weak discontinuity case, the discontinuity is embedded in the strain field. The finite element discretization of the three field variational problem contains added terms in either the displacement approximation, or strain approximation, or both. The enhancements to the fields are contained at the element (local) level and can be condensed out such that the global system of equations only contains the standard degrees of freedom.

Essentially the method is a combination of a discrete approach, where the deformation due to cracking is contained into a displacement discontinuity, and a smeared approach in which the deformation due to cracking is distributed over some material volume [61]. The method was developed to address stress locking issues that arise from a fully smeared approach by improving the kinematics of the highly localized strains. Weak discontinuities were first introduced in finite elements to represent shear bands through elements [62–64]. Discrete cracks have been represented by strong discontinuities in which a term is added to the principle of virtual work to represent the cohesive traction over the displacement discontinuity [65, 66]. In a comparison paper, [61], the authors classified the different approaches into three broad categories. The statically optimal symmetric methods cannot represent the kinematics of an open crack, but lead to traction continuity between the bulk elements and the tractions across the discontinuity. The kinematically optimal symmetric can represent the kinematics consistently, but do not give traction continuity. And finally the

statically and kinematically optimal nonsymmetric methods can manage both the consistent kinematics and traction continuity, but as the name suggests, the tangent stiffness matrices are non symmetric.

1.2.3 Peridynamics

Another class of methods is based on peridynamics models, in which the theory of continuum mechanics is expressed as an integral equation as opposed to partial differential equations. In the peridynamic theory a continuum view of material taken in which the material points of the body are continuous and interact with all other points in its range. Material points interact with each other through a response function. The nonlinear response function is nonlinear so numerical integration over discretized subdomains is conducted. It has been used to model dynamic fracture and has shown agreement with experimental results in representing crack direction and branching [47,67–69]. In [67], the authors show that round off error in the computational peridynamic model generates asymmetries in crack patterns when the initial model is perfectly symmetric.

1.2.4 Discontinuous Galerkin

The discontinuous Galerkin approach for dynamic fracture problems was developed as an alternative to the standard cohesive zone approach (see Section 1.2.9 below). One of the main motivations for this approach is due to the apparent lack of scalability of the standard cohesive zone approach and difficulty in modeling 3D problems [70]. In this thesis, however, we show that the method is in fact scalable to massively parallel systems and with use of adaptivity 3D problems are successfully simulated.

In the discontinuous Galerkin method the weak form of the governing partial differential equations are with a continuous polynomial approximation of the field variables is ensure only inside the elements, meaning that they are discontinuous across element interfaces and internal boundaries. Numerical flux terms are introduced to address the inter-element discontinuity and a stabilization term is added for nonlinear mechanics problems [70]. Using this formulation, the cohesive approach can also be incorporated. When an external criterion is met to activate the traction-separation relation, the flux terms to control inter-element discontinuity are replaced by the integral form of the traction separation relation. A parallel implementation of the method was used to model 3D dynamic fracture and fragmentation [71]. Further advances with the discontinuous Galerkin method involve using spacetime elements in which time is treated as an extra dimension and discretized in much the same way as the spatial dimensions. The spacetime discontinuous Galerkin method with h-adaptivity has been used to model evolving discontinuities using a cohesive model [72, 73].

1.2.5 Molecular dynamics

A physically relevant approach to modeling fracture is through molecular dynamics approach. These methods are naturally computationally demanding, so they are often coupled with other methods to lessen the computational burden. Part of the body is modeled using molecular dynamics and the rest is modeled using continuum mechanics, e.g. finite elements. A main issue in this approach is the communication between the two parts of the body, the so-called “handshake region.” If not done carefully, spurious behavior in energy preservation and wave propagation lead to unphysical results. A number of approaches have been developed to give a gradual transition between the regions [74–78]. However, even with such advances, this approach is limited in that fracture can only occur in the molecular dynamics region of the domain, see reference [79], suggesting that additional adaptive features are needed.

1.2.6 Element deletion

The element deletion method is perhaps one of the simplest approaches to modeling fracture propagation. Using the standard finite element method, new crack surfaces are represented by a set of deleted elements. The elements are deleted by setting their stress to zero such that they have zero material resistance. This is accomplished through a stress-strain constitutive relation in which the stress tends to zero at high strains. This approach would be spuriously mesh dependent if adjustments to the stress-strain curve were not met because the energy released due to deleting an element is proportional to the element size. Thus, the softening curve needs to be adjusted such that the fracture energy is independent of element size [80]. While this method is widely used in commercial applications, (e.g. LS-DYNA [81]), it was shown in [80] that it does not accurately capture crack velocities and patterns even for relatively simple geometries.

1.2.7 Mesh free methods

Mesh free methods provide a representation of the unknown fields in a system based solely on nodes rather than on elements and nodes. A number of properties make such methods attractive for fracture simulation, [82] listed them as: natural adaptivity, reduced bias from discretization, improved robustness under large deformation, smoothness, and multiscale capability. It should be noted that these methods are not entirely mesh-free as a background grid is necessary to perform numerical integration of the governing equations. The displacement field at any point $\mathbf{x}$ in meshless methods is a linear combination of the contributions of displacements from a set of nodes in the vicinity of $\mathbf{x}$. A crack is represented by modifying the nodes included in the neighborhood of $\mathbf{x}$ by excluding those separated by the line of discontinuity [82]. An alternative approach to representing cracks was presented in [83], where a crack is modeled by a discrete set of cracks that lie on particles. The discrete cracks are represented by a discontinuous enrichment to the displacement field. While the model has been used to capture dynamic fracture and branching behavior, it does not have the same level of accuracy in representing the fractured surface as other methods.

1.2.8 Virtual internal bond models

The virtual internal bond model (VIB), represents the microstructure of a material by spatially distributing cohesive bonds throughout the domain according to a spatial bond density function [82,84]. The Cauchy-Born rule is used to connect the cohesive behavior at the continuum level with that of the micro-structural level, meaning that the cohesive-type law is incorporated directly into the constitutive relation of the material. The strain energy function is constructed to take into account both the elastic and fracture behavior by equating the strain energy function on the continuum level to the potential energy stored in the cohesive bonds during deformation. The model was extended to include two different fracture energies for each mode of fracture in quasi-brittle materials [85].

1.2.9 Cohesive zone models: Present approach

In the cohesive zone model approach, the fracture process zone ahead of the crack tip is approximated by a nonlinear traction-separation relation. This approach is attractive in its simplicity: the degrading and softening mechanisms where micro-cracks and voids initiate and coalesce ahead of the crack tip are not explicitly modeled, rather they are approximated by the cohesive zone [86, 87]. The concept is illustrated in Figure 1.5. The macro crack tip contains zero tractions and complete separation, then ahead of this

point the traction increases and opening decreases. In the simplest sense, the cohesive relation is only a function of displacement, thus these elements can be incorporated with any type of bulk material model, e.g hyperelastic [88, 89], viscoelastic [90], etc.

Cohesive models are either constructed from a potential or based on heuristic relation. Non potential based models are generally simple to construct (e.g. bilinear cohesive model, trapezoidal, exponential, etc.), however the main limitation is that they are not physically based and can lead to incorrect results [91]. Conversely, in potential-based models the normal and tangential tractions are determined by taking the derivative of the potential with respect to the normal opening and tangential opening, respectively. The potential function is associated with physically relevant field quantities at the atomistic or continuum level. A detailed review of different types of cohesive relations can be found in reference [91].

Cohesive elements can be inserted into the mesh before the simulation begins in the so-called intrinsic model. In this case, the model contains an initial elastic range where opening of the cohesive elements starts before the critical separation is reached and softening begins. The intrinsic approach is known to produce artificial compliance in the system [92]. In the extrinsic model, the criteria to activate the elements is external to the model. Once activated the elements are inserted into the mesh on the fly. The extrinsic model is known to suffer from time discontinuity between the bulk and cohesive elements at the time of insertion [4, 93].

Since the mesh topology is changed on the fly in the extrinsic case, additional mesh adaptive procedures (e.g., refinement and coarsening) can be introduced under the same computational framework. The mesh topology is not changed for the intrinsic scheme, so adding the computational effort to provide mesh adaptivity would be excessive. Secondly, if mesh adaptivity were to be introduced for the intrinsic case, then internal variables associated with the cohesive model would need to be mapped from the old elements to the new ones, which is not a trivial problem (see Chapter 2). In the extrinsic model, mesh adaptivity can be performed before cohesive elements are inserted, thus mapping on the cohesive elements is not necessary.

It should be noted that cohesive elements have an impact on the critical time step in the conditionally stable explicit time integration scheme. Effectively, the measure of the mesh discretization, l_e, is governed by the smaller of the distance between nodes and the domain of dependence of the cohesive zone. When an intrinsic cohesive model is used the time step needs to be even smaller than for an extrinsic model because the initial elastic region needs to be resolved.

The impact on the numerical results between the two models has been studied by serval authors. In [92], the authors explain that the extrinsic model is expected to maintain its velocity if large branching does not occur. However, micro-branching off of the main crack was prohibited in the study in [92], and it is known that crack speeds are higher when micro-branching is not present. Furthermore, in [92, 94], the authors noted that macro-branching is present with intrinsic models, but not in extrinsic models for the same problem. The overall compliance of the problem is higher for the intrinsic case because of the initial elastic region, which is not present for the extrinsic case. The result of this overall lower compliance is that the stress concentrations are not evident at crack tips, especially if the initial slope is shallow [94]. This added compliance in the intrinsic model is not physical and increases with the number of elements insert *a priori*. If the crack pattern is known (for example, in fracture that occurs at material interfaces) then the compliance may be low enough to be permissible. However, for problems in which the crack pattern is unknown (e.g., branching problems as the ones investigated in this work) the extrinsic model should be employed over the intrinsic model.

When used with standard finite elements, the cohesive model approach requires a sufficiently fine mesh to capture crack initiation and propagation. Adaptive remeshing may be utilized to provide the appropriate

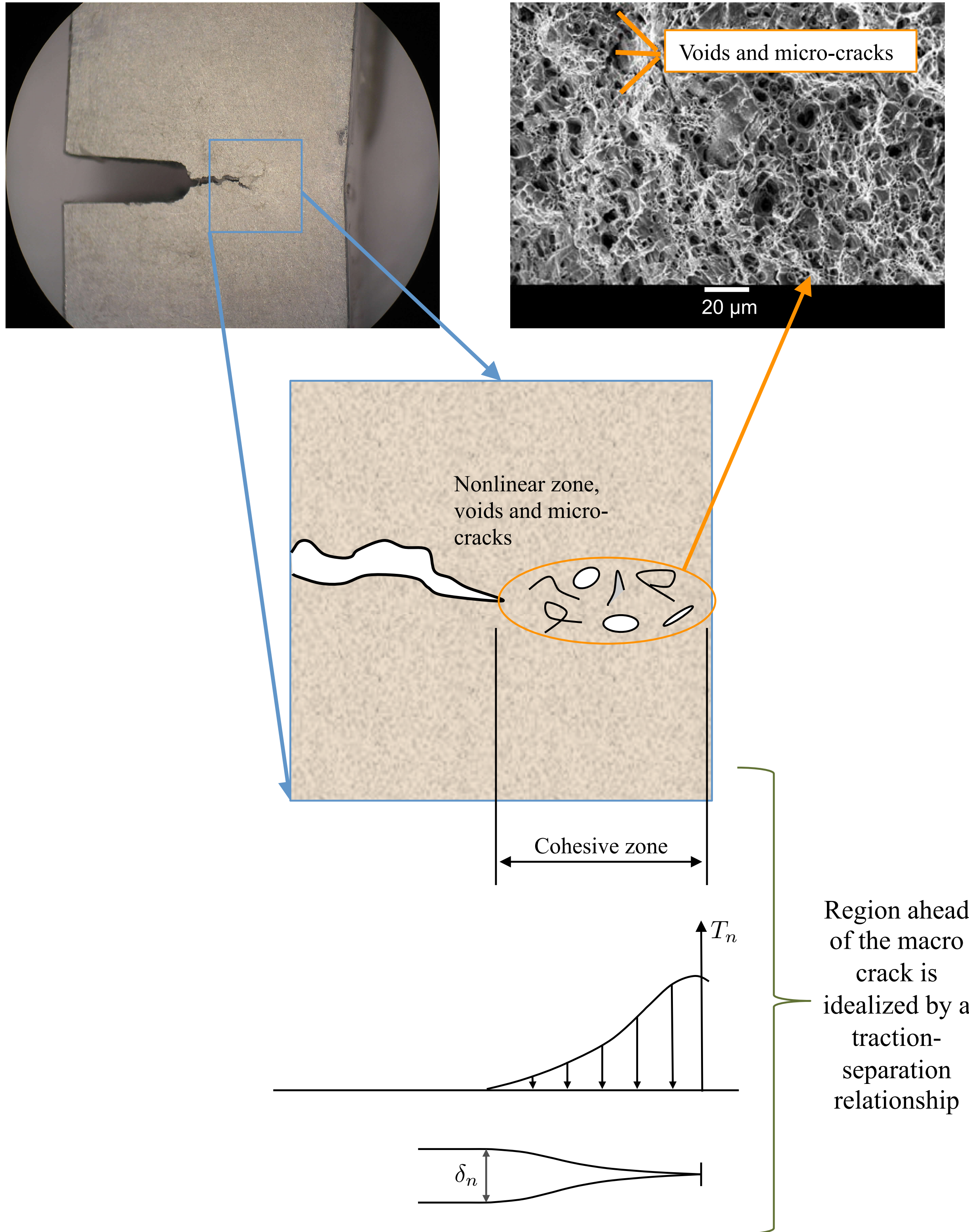


Figure 1.5: Schematic of the cohesive zone model approach: the cohesive zone ahead of the macro crack tip includes voids and micro-cracks which is idealized by a traction separation relationship. Ahead of the macro crack tip the traction increases and complete separation is present at the macro crack tip then decreases through length of the cohesive zone. Experimental fracture specimen image from http://makine.dogus.edu.tr/en/?page_id=20 and SEM image of fracture surface from University of Cambridge Dissemination of IT for the Promotion of Materials Science (DoITPoMS) available at http://www.doitpoms.ac.uk/tlplib/mechanical-testing/results1.php

level of discretization where needed without making the problem computationally untenable. Even for problems with extensive fragmentation, remeshing may become too extensive throughout the domain and results may not be obtainable [80]. In this case, the applicability of the cohesive zone approach is limited.

The range of applicability of the cohesive zone approach is ultimately based on the power of the computational implementation and on the availability of material properties [95]. In the standard cohesive zone model approach, the bulk elements and the cohesive elements are disjointed. While this gives the approach some flexibility in the types of material materials that can be modeled, it is also fair to point out that the disconnect is inconsistent. The standard cohesive zone approach could be seen as a macroscale model suitable for brittle and quasi brittle fracture [96]. To model very small cracks (sub micron), the constitutive relations inside the cohesive zone may become inaccurate because the size dependence of irreversible plasticity is not reflected in this approach. Thus, researchers have developed multiscale cohesive zone model approaches. In [96], the constitute relation for the bulk and cohesive are modeled together with the Cauchy-Born rule to yield an atomistically based constitute relation. A finite width compliant cohesive zone is added to model non uniform, local deformation and the cohesive zone is properly connected to the kinematics of the bulk material.

Despite some limitations, the standard cohesive zone approach is quite powerful, hence it is the approach used throughout this thesis. Specifically, we use a potential based, extrinsic cohesive zone model (see Section 3.1.3 for details on the constitutive model). We use adaptive remeshing to provide additional directions for crack propagations (Chapter 3) and to achieve a sufficiently fine mesh when and where needed in the simulation (Chapters 4 and 5) . We also utilizes the central difference time integration scheme in our approach. A small time step is necessary to capture the high frequency behavior associated with brittle fracture, so an explicit scheme is natural to consider. By enforcing a lumped mass matrix, we arrive at a fully explicit scheme in which a linear system does not need to be solved at every step. This method is also advantageous in that it is easily parallelized, which is important for the developments of Chapter 5. Alternative approaches to the explicit central difference scheme could be explored, however they are outside the scope of this work. Energy conservation is not assumed in the central different scheme, but loss is minimal when a sufficiently small time step is used [97]. Energy conservation would be easier to achieve in an implicit scheme, however when adaptivity is employed the system matrices would need to be assembled frequently, adding to the computational cost. Hybrid implicit-explicit schemes have been introduced to in an attempt to take advantage of both approaches, [98, 99].

1.3 Contributions

This work is the result of a number of fruitful collaborations. I am grateful for the opportunity to have worked with so many people with such a broad range of expertise. Our combined knowledge and skills made the advancements in this document possible. For the purpose of clarity, I will indicate my specific contributions. I have organized the contribution by chapter.

In addition to the contributions described below, I have worked on a number of other topics that have resulted in peer reviewed journal publications during my PhD that are not included in this thesis. This includes: analysis of a nonlinear finite element solution scheme [100], development of an analysis and design tool considering low temperature cracking of asphalt pavements [21, 90], and geometric investigations of adaptive refinement of polygonal elements [89].

My specific contributions are as follows:

Chapter 2: Mapping internal state variables in numerical simulation of ductile failure[1]

- Automated finite element analysis and internal state variable mapping scheme by developing Python scripts
- Developed tools to post-process, compare and efficiently visualize results of finite element analyses with internal state variable mapping
- Direct development and implementation in proprietary research codes used in the Mechanics Department at Sandia National Lab
- Evaluated the internal state variable mapping scheme via extensive numerical testing to stress the algorithms and found little numerical diffusion even in the extreme conditions

Chapter 3: Towards reduction in mesh bias using adaptive splitting of polygonal finite elements [2]

- Developed and implemented adaptive splitting of CVT polygonal finite elements for dynamic fracture simulation on CCS specimen (equal contributor with D. Spring)
- Quantified geometric properties of structured and unstructured finite element meshes by developing metrics to measure accurate crack representation
- Showed geometric isotropy and accuracy of unstructured meshes is enhanced with adaptive element splitting and performs far better than structured meshes
- Developed an algorithm to mesh an arbitrary domain with random polygonal elements (applied to CCS domain)
- Performed dynamic fracture simulation on CCS specimen using polygonal elements (CVT and random) with adaptive element splitting

[101]

Chapter 4: Adaptive refinement and coarsening on structured 3D meshes [3]

- Connected the physical simulation with its numerical representation in the TopS data structure through call-back functions
- Developed finite element analysis code to perform 3D simulation with computationally efficient mesh refinement and coarsening using the topological data structure
- Developed algorithms to locally transfer field variables between 3D mesh discretizations undergoing refinement or coarsening during dynamic simulation

---

[1] A journal paper that includes my contributions on the internal state variable mapping scheme studies is currently in preparation with collaborators at Sandia National Laboratory

[2] S. E. Leon, D. W. Spring, and G. H. Paulino. "Reduction in Mesh Bias for Dynamic Fracture Using Adaptive Splitting of Polygonal Finite Elements." International Journal for Numerical Methods in Engineering, 100(8): 555–76, 2014. Note: First two authors are equally contributing.

[3] S. E. Leon, R. Espinha, W. Celes, G. H. Paulino. "Three-dimensional dynamic cohesive fracture simulation with adaptive mesh refinement and coarsening." In preparation.

- Performed 3D adaptive mesh refinement and coarsening for dynamic fracture applications including a confined crack and three-point bending specimen
- Demonstrated significant improvements to computational efficiency (with comparable numerical accuracy) through evaluation of adaptive mesh refinement and coarsening

Chapter 5: Massively parallel adaptive mesh refinement and coarsening for dynamic fracture simulations [4]

- Determined the relevant fracture mechanics for implementation of the GPU-designed data structure and algorithms to perform the 2D adaptive mesh refinement and coarsening
- Developed metrics to quantify fracture patterns featuring micro-branches
- Studied the reduced scale micro-branching problem and showed that GPU introduced randomness does not influence the physically relevant properties of fracture
- Systematically explored geometry, mesh, boundary conditions, material properties to determine input parameters for physically accurate simulations

1.4 Document outline

The next four chapters of this dissertation detail the advancements made during the course the PhD of the author. While the work is broad, the common thread linking them together is the presence of mesh adaptivity as part of the simulation process.

As we will demonstrate in this thesis, remeshing during the simulation process may be necessary for a variety of reasons, e.g. due to element distortion, moving boundaries, insufficient/excessive level of refinement, etc. Whenever elements are inserted or moved in the mesh the internal variables that describe the state of the material must be transferred from the old mesh to the new one. In Chapter 2, we study the intricate problem of consistently mapping internal state variables after adaptive remeshing. Next, we shift the focus to dynamic brittle failure and model the creation of new surfaces in Chapters 3, 4, and 5. In each chapter we focus on a different methodology, but the common theme among them all is adaptive remeshing. In Chapter 3, we propose a method to improve the crack patterns using unstructured polygonal finite elements and local adaptive remeshing technique in which elements are split to allow cracks to propagate through them. We investigate two types of polygonal discretizations and extensively study the element splitting technique through geometric exercises and physical examples. In the next two body chapters we focus on increasing computational efficiency of dynamic fracture simulation using adaptive mesh refinement and coarsening strategies. We move back to a structured mesh because of the convenience of generating a hierarchal refinement scheme. Of course we lose some of the benefits of the unstructured mesh, but we make up for the loss with the gains in computational efficiency. Specifically, in Chapter 4, we develop the mesh refinement and coarsening scheme for three dimensional models. The geometric and physical aspects of local mesh refinement and coarsening are detailed and the algorithm for executing the strategy is demonstrated. Using two numerical examples, we show the ability of the method the solve large scale problems in a fraction of the time as a uniform counterpart. Finally, in the last body chapter, we continue to the analogous

[4] S. E. Leon*, A. Alhadeff*, W. Celes, G. H. Paulino, "Massively parallel adaptive mesh refinement and coarsening for dynamic fracture simulations." To be submitted to Engineering with Computers in November, 2014. Note: First two authors are equally contributing.

two dimensional mesh refinement and coarsening schemes on a massive parallel platform. Until Chapter 5, all simulations are conducted on CPU systems, but for this work we are interested in developing the framework to conduct rapid simulations. We move back to 2D systems with adaptive mesh refinement and coarsening, as the computational resources necessary for 3D systems on the GPU are not yet available. We perform a host of studies through numerical examples to examine the impact that high performance parallel programming has on the final fracture results. Because of the speedup provided by the GPU, we have the ability to examine aspects of the simulation that would otherwise be too computationally cumbersome. This dissertation closes with a summary of the contributions and a detailed discussion of potential future research directions that build upon those contributions in Chapter 6. Appendices containing pertinent codes and scripts are included for the interested reader.

Chapter 2

Mapping internal state variables in numerical simulation of ductile failure

Predictive simulation of failure of ductile structural materials, such as metals and alloys, is an open problem in the computational mechanics community. Only two years before the completion of this dissertation, Sandia National Laboratories created the so-called Sandia Challenge to bring members of the mechanics community together to predict the failure of a geometrically simple but mechanically complex structure [102]. Over 50 scientists from 20 institutions volunteered to participate in the challenge, and published their results in a special issue of the International Journal of Fracture in 2014 [103–111]. The findings revealed that the fracture problem is indeed not solved and motivated the need for development of improved understanding of ductile failure and modeling technologies.

In this chapter we address the problem of ductile failure using the finite element method with remeshing. When deformations become extremely large, the finite element mesh may become so distorted that the numerical solution cannot be obtained. We combat this severe distortion by remeshing the domain either locally in the regions of distortion or globally over the entire mesh.

Ductile materials are described at any point by a set of state variables. In contrast to brittle materials, the state of a ductile material cannot be determined from the displacement field alone. Thus, in order accurately represent the material through the remeshing process, the internal variables that describe the current state of the material must be mapped from the old, distorted mesh, to the new mesh comprised of well-formed elements.

Mapping internal state variables involves projecting them to the nodes of the old mesh, then interpolating them to the integration points of the new mesh. Extending variables requires special care to ensure error is minimized and that the element quantities remain in their original admissible spaces. In the present work, we focus our attention on the mapping of internal state variables and in particular we examine the method of Lie-group interpolation with variational recovery, proposed in [112].

The remainder of this chapter is outlined as follows. First, we review the current state of the art for mapping internal state variables in Section 2.1. Then, the Lie-group interpolation with variational recovery scheme adopted in this work is described in Section 2.2. Next, the computational framework for analysis and investigation of this scheme is detailed in Section 2.3. Finally, in Section 2.4a series of studies is performed on the mapping scheme, and the results lead to some discussion and recommendations about use of the scheme.

2.1 Review of large deformation modeling

Large deformation processes require special modeling attention; here we will briefly review common finite element approaches to account for this behavior.

2.1.1 Large deformation formulations

For solid mechanics applications, Lagrangian formulations are typically utilized to account for finite deformations. In Lagrangian formulations the computational grid (mesh) follows the continuum in its motion and the grid points are permanently connected to the material points. This is in contrast to an Eulerian formulation, typically used in fluid mechanics applications, where the computational grid stays fixed and continuum moves in relation to it. Lagrangian formulations are computationally advantageous, especially when history dependent material models are utilizes, because the finite element always contains the same materiel points and expensive convective terms are not present in the formulation. Furthermore in solid mechanics applications the domain boundaries need to be resolved accurately. This is well handled by Lagrangian formulations, but not for Eulerian formulations. The downside to Lagrangian formulations is that severe element distortion may result in loss of accuracy or inability to converge numerically. While element distortion is not an issue in Eulerian formulations because the mesh does not move, the benefits of the Lagrangian formulations for solid mechanics outweigh the negatives. In the remainder of this section, we will review the basic types of Lagrangian formulations and discuss two means of handling element distortion that may arise during the large deformation process [113–116].

The difference in Lagrangian formulations lies in the definition of the reference configuration in which the integration is performed. In the Total Lagrangian (TL) formulation the reference configuration remains fixed, while in the Updated Lagrangian (UL) formulation the reference configuration is updated, it is typically taken as the last converged equilibrium configuration. It should be noted that the reference configuration for the TL formulation is typically taken as the original undeformed configuration, however in a so-called staged analysis, the reference configuration may be updated to the initial configuration at the start of an analysis stage. Both the TL and UL formulations are suitable for finite deformations and strains, and in principle the TL and UL methods are identical. However, there is some disagreement in the literature as to which is more suitable for larger deformations [117, 118].

In either case, there comes a point when the finite element mesh may become severely distorted and result in unacceptable levels of discretization error or even cause lack of convergence of the nonlinear solution algorithm. Several remedies have been explored in the computational mechanics literature, and in fact, this still represents an open question in the field of large deformation modeling. Applications in which extremely large deformation makes the analysis impossible with a standard TL or UL formulation include metal powder forming [119], fluid-solid interaction [115], fault propagation [120], etc.

A first way to handle element distortion is to combine the Lagrangian and Eulerian formulations through the Arbitrary Eulerian Lagrangian (ALE) formulation. We will not provide the full derivation of the classic ALE method, which combines the benefits of the Eulerian and Lagrangian approaches, for that the reader is directed to [115]. The method is commonly used in solid mechanics and for fluid-solid interaction problems. However it also has applications in large deformation solid mechanics problems in which element distortion becomes excessive; this is version we will focus on here. In the ALE formulation, two phases are considered: a Lagrangian phase in which the material points are adjusted with the computational grid, and a convective phase in which the convective terms (relative velocity between the continuum and grid) are taken into account. These phases may be done simultaneously [121] as in the classic ALE formulation (referred to as the *unsplit* approach) or separately [122, 123] (referred to as the *split* or Updated ALE approach). In the split approach the convective phase is left out of the iteration loop. The Lagrangian iterations are performed inside the loop until equilibrium is reached, then the convective phase is performed outside the loop. This approach is computationally more efficient than the unsplit approach, because the convective terms only

need to be computed once per step. In the unsplit ALE approach, quadratic convergence of the nonlinear solution scheme is lost due to the to the inclusion of the convective terms. However, this comes at the cost of loss of equilibrium at the end of the step. The convective phase pulls the system out of equilibrium then extra residual forces are present on the body at the start of the next step. However, this disadvantage has been show to be insignificant through numerical experiments in [124] in which the author compared the split and unsplit versions on several benchmark problems.

Alternatively, a purely Lagrangian formulation may be utilized and remeshing performed when element distortion is excessive. Many authors have investigated remeshing in the context of both the TL and UL formulations [125–127]. In these applications, remeshing is trigged by some indicator, e.g. strain measure, volume element quantity, fixed intervals of increments, mesh penetration, etc, and the target mesh is achieved by an error estimators, such as the Super-convergent Patch Recovery (SPR) [128]. Notice that remeshing will establish a new reference configuration, for either the TL or UL approach. In the TL approach this corresponds to the staged analysis, mentioned previously. As part of the remeshing approach, node and element variables need to be transferred from the old mesh to the new mesh, which is the focus of the remainder of this Chapter. We start with a review of mapping approaches in the next section.

2.1.2 Mapping internal state variables

Internal state variables may be mapped from one mesh to another in a number of ways, and there exists a vast literature on the subject. Before reviewing the current state of the practice, we first explain when mapping of internal state variables is not necessary. In a linear elastic material for example, state variables are not needed to describe the material state, the displacement field is sufficient. In a typical finite element formulation, the displacements are nodal quantities that exits at the nodes. Nodal quantities are easily evaluated anywhere in the domain by virtue of the finite element shape functions. Thus, quantities that are computed at nodes do not need any mapping other than interpolation by the shape functions. This is in contrast to element quantities such as damage or stress that live discretely at the integration points; moving them from integration points to other points in the mesh is nontrivial, and is in fact the focus of this chapter. We should note that for hyper-elastic models, stresses can be calculated directly from the displacement, and therefore they do not need to be mapped like an internal state variable. Instead, the displacements can be interpolated to the nodes of the new mesh, then stresses can be computed from these displacements.

A common approach is to use a combination of extrapolation then interpolation [129], though this method is known to be non-conservative [130]. That is, element quantities may not belong to their original admissible spaces after the procedure is conducted. In the extrapolation/interpolation approach, the element quantities present at integration points are extrapolated to the nodes of the element using the standard finite element shape functions. The final nodal quantity can then be averaged from the contributions of its adjacent elements. Furthermore, this procedure will generate volumetric error between the old and new discretizations, which can cause numerical problems if not minimized [112]. An alternative approach is to simply assign the node the value from the integration points. Once the quantities are known at the nodes, then the finite element shape functions are used to interpolate them anywhere in the domain. The integration points in the new mesh are located with respect to the elements of the old mesh, an inverse mapping is performed to find the location of the integration point in the parent coordinates of the old mesh, and the shape functions in the parent domain are used to interpolate the elements nodal quantities to the integration point.

To overcome these and other issues is mapping internal variables (see [129, 131] for a discussion), more sophisticated mapping procedures have been introduced in the literature. The Hu-Washizu principle is used

in [132] to derive transfer operators from an extended variational principle for strain localization problems. The problem is viewed as an evolution in time, where a discontinuity will be present after mapping from the old mesh to the new mesh in [133]. Then, error estimates between these meshes are used to minimize the jump. In [131], the authors perform an L_2 minimization of the internal state variables between the old and new mesh, in which the essential computational problem lies in calculating the volume of intersections between the old and new meshes. The authors present an efficient means to approximate the regions by solving an appropriately defined minimization problem. Several schemes including L_2-projection schemes, element-oriented local transfer, and patch-oriented transfer are compared in [134]. Additionally, the authors propose integrating the material law at the nodes directly, such that no transfer is necessary from integration points that lie inside the element.

As discussed in [134], stress recovery techniques can also be viewed as a means to transfer internal state variables. For example, the super convergent patch recovery (SPR) technique is commonly used to transfer data from one mesh to another [128]. A 3D modified version of the SPR with C^0, C^1, and C^2 continuities was developed to project state variables from integration points to nodes during large plastic deformations [127].

2.2 Lie group interpolation and variational recovery

The method for transferring internal state variables from one mesh (or region within a mesh) to another studied in this work is that proposed in [112]. The recovery procedure achieves two goals: (i) the error between the source and target internal variables is minimized in the L_2 sense, and (ii) the internal variables remain in their original admissible spaces.

2.2.1 Projection scheme

The internal variables at the integration points on the source mesh are extended to the nodes of the source mesh via a global projection scheme. Once the quantities are known at the nodes, then they can be computed at any point in the mesh through the interpolation. The projection is achieved by means of a three-field finite element formulation. We begin with the energy functional

$$\Phi\left[\boldsymbol{\varphi}\right] := \int_{\Omega} A\left(\mathbf{F},\, \mathbf{z}\right) dV - \int_{\Omega} R\mathbf{B}\cdot\boldsymbol{\varphi} dV - \int_{\partial_T\Omega} \mathbf{T}\cdot\boldsymbol{\varphi} dS \tag{2.1}$$

for a body $\Omega \subset \mathbb{R}^3$ subjected to the motion described by the mapping $\mathbf{x} = \boldsymbol{\varphi}\left(\mathbf{X}\right)$ and body force, $\mathbf{B}$. The deformation gradient is defined as $\mathbf{F} := \nabla\boldsymbol{\varphi}$, $A\left(\mathbf{F},\, \mathbf{z}\right)$ is the Helmholtz free-energy density, $\mathbf{z}$ is the set of internal state variables, R is the mass density, and $\mathbf{T}$ are the tractions applied to the traction boundary $\partial_T\Omega$. Next we introduce a constraint to the functional to make the source internal variables, $\mathbf{z}$, equal to the target internal variables, $\overline{\mathbf{z}}$, through a Lagrange multiplier, $\boldsymbol{\lambda}$

$$\Phi\left[\boldsymbol{\varphi},\, \overline{\mathbf{z}},\, \boldsymbol{\lambda}\right] := \int_{\Omega} A\left(\mathbf{F},\, \overline{\mathbf{z}}\right) dV + \int_{\Omega} \boldsymbol{\lambda}\cdot\left(\overline{\mathbf{z}} - \mathbf{z}\right) - \int_{\Omega} R\mathbf{B}\cdot\boldsymbol{\varphi} dV - \int_{\partial_T\Omega} \mathbf{T}\cdot\boldsymbol{\varphi} dS \tag{2.2}$$

The Helmholtz energy is now evaluated with the target internal variables, which is admissible because of the constraint. Now we make the assumption that the fields belong to a space of square-integrable functions with square-integrable first derivatives; $\boldsymbol{\varphi}$ belongs to space $\mathcal{U}$ of $\mathbb{R}^3$ valued functions, while $\overline{\mathbf{z}}$ and $\boldsymbol{\lambda}$ belong to space $\mathcal{V}$ of $\mathbb{R}^q$ valued functions, where q is the number of internal variables. The governing equations result from minimizing the functional: let $\boldsymbol{\eta},\, \boldsymbol{\zeta} \in \mathcal{V}$ be test functions for $\overline{\mathbf{z}}$ and $\boldsymbol{\lambda}$, respectively, while $\boldsymbol{\xi} \in \mathcal{U}$ where

$\boldsymbol{\xi} = 0$ on $\partial_\varphi \Omega$. We start with the variation with respect to $\boldsymbol{\varphi}$:

$$\begin{aligned}
D\Phi\left[\boldsymbol{\varphi}, \overline{\mathbf{z}}, \boldsymbol{\lambda}\right](\boldsymbol{\xi}) &= \frac{d}{d\varepsilon}\left[\Phi\left(\boldsymbol{\varphi}+\varepsilon\boldsymbol{\xi}, \overline{\mathbf{z}}, \boldsymbol{\lambda}\right)\right]\Big|_{\varepsilon=0} \quad (2.3)\\
&= \int_\Omega \frac{d}{d\varepsilon}\left[A\left(\mathbf{F}, \overline{\mathbf{z}}\right)\right]\Big|_{\varepsilon=0} dV + \int_\Omega \frac{d}{d\varepsilon}\left[\boldsymbol{\lambda}\cdot\left(\overline{\mathbf{z}}-\mathbf{z}\right)\right]\Big|_{\varepsilon=0} - \\
&\quad \int_\Omega \frac{d}{d\varepsilon}\left[R\mathbf{B}\cdot\left(\boldsymbol{\varphi}+\varepsilon\boldsymbol{\xi}\right)\right]\Big|_{\varepsilon=0} dV - \int_{\partial_T\Omega}\frac{d}{d\varepsilon}\left[\mathbf{T}\cdot\left(\boldsymbol{\varphi}+\varepsilon\boldsymbol{\xi}\right)\right]\Big|_{\varepsilon=0} dS \quad (2.4)\\
&= \int_\Omega \frac{\partial A}{\partial \mathbf{F}} : \frac{\partial \mathbf{F}}{\partial \varepsilon}\Big|_{\varepsilon=0} dV - \int_\Omega R\mathbf{B}\cdot\frac{d\left(\boldsymbol{\varphi}+\varepsilon\boldsymbol{\xi}\right)}{d\varepsilon}\Big|_{\varepsilon=0} dV\\
&\quad - \int_{\partial_T\Omega}\mathbf{T}\cdot\frac{d\left(\boldsymbol{\varphi}+\varepsilon\boldsymbol{\xi}\right)}{d\varepsilon}\Big|_{\varepsilon=0} dS \quad (2.5)\\
&= \int_\Omega \frac{\partial A}{\partial \mathbf{F}} : \left(\frac{\partial \mathbf{F}}{\partial \nabla\boldsymbol{\varphi}}\frac{\partial \nabla\boldsymbol{\varphi}}{\partial \varepsilon}\right)\Big|_{\varepsilon=0} dV - \int_\Omega R\mathbf{B}\cdot\boldsymbol{\xi}dV - \int_{\partial_T\Omega}\mathbf{T}\cdot\boldsymbol{\xi}dS \quad (2.6)\\
&= \int_\Omega \frac{\partial A}{\partial \mathbf{F}} : \nabla\boldsymbol{\xi}dV - \int_\Omega R\mathbf{B}\cdot\boldsymbol{\xi}dV - \int_{\partial_T\Omega}\mathbf{T}\cdot\boldsymbol{\xi}dS \quad (2.7)
\end{aligned}$$

The other two terms are calculated similarly, and the resulting variations are

$$\begin{aligned}
D\Phi\left[\boldsymbol{\varphi}, \overline{\mathbf{z}}, \boldsymbol{\lambda}\right](\boldsymbol{\xi}) &= \int_\Omega \mathbf{P} : \nabla\boldsymbol{\xi}dV - \int_\Omega R\mathbf{B}\cdot\boldsymbol{\xi}dV - \int_{\partial_T\Omega}\mathbf{T}\cdot\boldsymbol{\xi}dS \quad (2.8)\\
D\Phi\left[\boldsymbol{\varphi}, \overline{\mathbf{z}}, \boldsymbol{\lambda}\right](\boldsymbol{\eta}) &= \int_\Omega \left(\boldsymbol{\lambda} + \frac{\partial A}{\partial \overline{\mathbf{z}}}\right)\cdot\boldsymbol{\eta}dV \quad (2.9)\\
D\Phi\left[\boldsymbol{\varphi}, \overline{\mathbf{z}}, \boldsymbol{\lambda}\right](\boldsymbol{\zeta}) &= \int_\Omega \left(\overline{\mathbf{z}}-\mathbf{z}\right)\cdot\boldsymbol{\zeta}dV \quad (2.10)
\end{aligned}$$

The function is minimized if these variations all vanish. Next, we approximate the solution by discretizing the fields and test functions in Equations 2.8-2.10. The original spaces are replaced with finite dimensional subspaces $\mathcal{U}_h \subset \mathcal{U}$ and $\mathcal{V}_h \subset \mathcal{V}$, to which the functions belong

$$\begin{aligned}
\boldsymbol{\varphi}_h &:= N_i\left(\boldsymbol{X}\right)\boldsymbol{\varphi}_i \in \mathcal{U}_h, \qquad \boldsymbol{\xi}_h := N_i\left(\boldsymbol{X}\right)\boldsymbol{\xi}_i \in \mathcal{U}_h, \quad (2.11)\\
\overline{\boldsymbol{z}}_h &:= M_i\left(\boldsymbol{X}\right)\overline{\mathbf{z}}_i \in \mathcal{V}_h, \qquad \boldsymbol{\eta}_h := M_i\left(\boldsymbol{X}\right)\boldsymbol{\eta}_i \in \mathcal{V}_h, \quad (2.12)\\
\boldsymbol{\lambda}_h &:= M_i\left(\boldsymbol{X}\right)\boldsymbol{\lambda}_i \in \mathcal{V}_h, \qquad \boldsymbol{\zeta}_h := M_i\left(\boldsymbol{X}\right)\boldsymbol{\zeta}_i \in \mathcal{V}_h, \quad (2.13)
\end{aligned}$$

The finite dimensional fields are given by $\boldsymbol{\varphi}_h$, $\boldsymbol{\xi}_h$ and the finite dimensional test functions are $\overline{\mathbf{z}}_h$, $\boldsymbol{\eta}_h$, $\boldsymbol{\lambda}_h$, $\boldsymbol{\zeta}_h$. Using the Galerkin method, the interpolation functions for the field and test functions are the same. Hence, N_i are the interpolation functions for $\boldsymbol{\varphi}$ and $\boldsymbol{\xi}$ and M_i are the interpolation functions for $\overline{\mathbf{z}}$, $\boldsymbol{\eta}$, $\boldsymbol{\lambda}$ and $\boldsymbol{\zeta}$. Summation convention is implied in the discretized equations above, where $\boldsymbol{\varphi}_h$ and $\boldsymbol{\xi}_h$ are summed up to the number of nodes in the source mesh, and $\overline{\mathbf{z}}$, $\boldsymbol{\eta}$, $\boldsymbol{\lambda}$ and $\boldsymbol{\zeta}$ are summed to the number of nodes in the target mesh. The discrete fields are first inserted into Equation 2.8

$$\begin{aligned}
\int_\Omega \mathbf{P} : \nabla N_i\boldsymbol{\varphi}_i dV - \int_\Omega R\mathbf{B}\cdot N_i\boldsymbol{\varphi}_i dV - \int_{\partial_T\Omega}\mathbf{T}\cdot N_i\boldsymbol{\varphi}_i dS &= 0\\
\int_\Omega \mathbf{P}\cdot\nabla N_i dV - \int_\Omega R\mathbf{B}N_i dV - \int_{\partial_T\Omega}\mathbf{T}N_i dS &= 0 \quad (2.14)
\end{aligned}$$

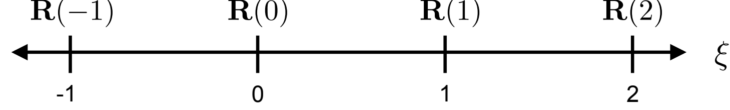


Figure 2.1: Location of known rotations, $\mathbf{R}(-1)$ and $\mathbf{R}(1)$, and those to be interpolated/extrapolated $\mathbf{R}(0)$ and $\mathbf{R}(2)$

then into Equation 2.9

$$\boldsymbol{\lambda}_h = -M_i \left(\int_B M_i M_j \mathbf{I} dv \right)^{-1} \int_B M_j \frac{\partial \mathbf{A}}{\partial \mathbf{z}} dV \tag{2.15}$$

and finally into Equation 2.10

$$\overline{\mathbf{z}}_h = M_i \left(\int_B M_i M_j \mathbf{I} dv \right)^{-1} \int_B M_j \mathbf{z} dV \tag{2.16}$$

The projection of the internal variables from the source to the target mesh is achieved through Equation 2.16. The shape functions M_i should be the same order or less than the shape functions N_i so that the projection equations may utilize the same integration scheme as the equilibrium equation and that no additional transfer of variables is necessary.

The projection scheme described here may be done in a global or local sense depending on what portion of the mesh is changed. If the mesh is only changed locally, e.g., in the vicinity of a crack tip, then the scheme would only need to transfer variables in that region. If however, the entire body is remeshed, due to large deformations of the finite elements for example, then the projection scheme would be applied globally.

2.2.2 Lie group interpolation

The second goal of the mapping procedure is to ensure that the internal variables remain in their admissible spaces after the mapping procedure. Certain internal variables belong to spaces which do not admit addition, therefore interpolation of these fields may result in a value of the internal variable that does not belong to the original space.

For example, consider the rotation tensor, which is computed and stored at integration points for certain material models. If the domain is remeshed during the analysis, then the rotation would need to be transferred from the integration points of the old mesh to those of the new mesh. Recall that a rotation $\mathbf{R}$ satisfies the properties $\mathbf{R}\mathbf{R}^T = \mathbf{I}$ and $\det\mathbf{R} = 1$. The properties define the special orthogonal Lie group, $SO(n)$, where n is the dimension of the space. Thus $\mathbf{R} \in SO(3)$ for the three-dimensional case. It would be incorrect to simply interpolate the rotation at the new node from old nodes, because the resulting value may not be in $SO(3)$. For example, consider the schematic shown in Figure 2.1, where the rotations $\mathbf{R}(-1)$ and $\mathbf{R}(1)$ are known

$$\mathbf{R}(-1) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad \mathbf{R}(1) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

If we use standard interpolation functions directly, then the resulting rotations, $\mathbf{R}(0)$ and $\mathbf{R}(2)$, will not be members of $SO(3)$. Let $N_1(\xi) = \frac{1}{2}(1-\xi)$ and $N_2(\xi) = \frac{1}{2}(1+\xi)$. The rotations at any point ξ are

$\mathbf{R}(\xi) = N_1(\xi)\mathbf{R}(-1) + N_2(\xi)\mathbf{R}(1)$, then

$$\mathbf{R}(0) = \begin{bmatrix} 0.5 & 0.0 & 0.5 \\ 0.0 & 0.5 & -0.5 \\ -0.5 & 0.5 & 0.0 \end{bmatrix}, \qquad \mathbf{R}(2) = \begin{bmatrix} -0.5 & 0.0 & 1.5 \\ 0.0 & 1.5 & 0.5 \\ -1.5 & -1.5 & 0.0 \end{bmatrix}$$

Notice that for both $\mathbf{R}(0)$ and $\mathbf{R}(2)$ $\det\mathbf{R} \neq 1$ and $\mathbf{R}\mathbf{R}^T \neq \mathbf{I}$, therefore $\mathbf{R}(0)$ and $\mathbf{R}(2)$ are not actually rotations. This simple example demonstrates that addition is not admitted on rotations, which is is the case for quantities that belong to a Lie group. To remedy this, a Lie group is mapped to its Lie Algebra via logarithmic maps, then back to its Lie group through exponential maps. Once a rotation is in the Lie algebra space, additional is permitted. In the example above, if the rotations are first mapped to their Lie algebra then interpolated, mapped back to their Lie group, then the resulting rotations will be in $SO(3)$. The Lie algebra of the special orthogonal group are skew-symmetric matrices, denoted by $so(3)$. We will denote the rotations in their Lie algebra by $\mathbf{r}$, then

$$\begin{aligned} \mathbf{r}(-1) &= \log\{\mathbf{R}(-1)\} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1.5708 \\ 0 & 1.5708 & 0 \end{bmatrix} \\ \mathbf{r}(1) &= \log\{\mathbf{R}(1)\} = \begin{bmatrix} 0 & 0 & 1.5708 \\ 0 & 0 & 0 \\ -1.5708 & 0 & 0 \end{bmatrix} \end{aligned}$$

We can interpolate and extrapolate using the functions from above to arrive at

$$\mathbf{r}(0) = \begin{bmatrix} 0 & 0 & 0.7854 \\ 0.0 & 0 & -0.7854 \\ -0.7854 & 0.7854 & 0 \end{bmatrix}, \qquad \mathbf{r}(2) = \begin{bmatrix} 0 & 0 & 2.5632 \\ 0.0 & 0 & -0.7854 \\ -2.5632 & 0.7854 & 0 \end{bmatrix}$$

Finally, we can apply the exponential map to take the quantities back to their Lie group

$$\begin{aligned} \mathbf{R}(0) &= \exp\{\mathbf{r}(0)\} = \begin{bmatrix} 0.7220 & 0.2780 & 0.6336 \\ 0.2780 & 0.7220 & -0.6336 \\ -0.6336 & 0.6336 & 0.4440 \end{bmatrix} \\ \mathbf{R}(2) &= \exp\{\mathbf{r}(2)\} = \begin{bmatrix} -0.6121 & -0.5374 & 0.5801 \\ -0.5374 & 0.8209 & 0.1934 \\ -0.5801 & -0.1934 & -0.7913 \end{bmatrix} \end{aligned}$$

Now $\det\mathbf{R} = 1$ and $\mathbf{R}\mathbf{R}^T = \mathbf{I}$ for both $\mathbf{R}(0)$ and $\mathbf{R}(2)$. There are several techniques for performing the logarithmic and exponential maps [135–137]. For the example shown above, the built in Matlab functions `logm` and `expm` used, which are based on scaling and squaring method combined with Padé approximation [138].

The internal variable interpolation procedure is accomplished through a combination of the Lie group/Lie Algebra mapping and the L_2-projection. Beginning from a point where the internal variables are known at the integration points, we perform the logarithmic mapping on those variables that belong to a Lie group.

It should be noted the variables that do not belong to a Lie group can be interpolated directly, so the mapping steps are not necessary. Next, the L_2-projection scheme is applied to move the element quantities from integration points of the source mesh to the nodes of the source mesh. With quantities now at the nodes, they can be computed anywhere through the standard finite element shape functions. The integration points of the target mesh are located with respect to the source mesh, then the appropriate nodes are used for interpolation. Finally, the exponential mapping is applied to those quantities that belong to a Lie algebra, taking them back to their original Lie group.

2.3 Computational framework for analysis of Lie-ground interposition scheme

In this section we discuss the numerical framework to perform large deformation analysis, remesh after element distortion, and map internal variables. The analysis is employed via a Total Lagrangian formulation, in which the problem is formulated with respect to the reference (undeformed) configuration. Once the finite element mesh is suitably distorted, the analysis is stopped, the domain is remeshed, and pushed back to the original reference configuration. Thus we have a new mesh on the original reference configuration. Then, the mapping procedure is invoked as follows:

- Internal state variables belonging to a Lie Group undergo logarithmic mapping to take them into their corresponding Lie Algebra
- Internal state variables are projected from the integration points of the source mesh to the finite element nodes of the source mesh using the L_2 projection scheme discussed in Section 2.2.1
- Nodal quantities on the source mesh (includes ISVs now) are interpolated to the integration points of the target mesh using the finite element shape functions
- Internal state variables in their Lie Algebra undergo exponential mapping to take them back into their original Lie Groups

After remeshing, the elements have the best aspect ratio in the current configuration and the new reference configuration potentially contains distorted elements. Therefore, we perform the mapping procedure above in the current configuration by pushing the solution forward through the displacement vector. Once the mapping is complete we pull back to the reference configuration.

The numerical implementation of this procedure relies on a combination of codes and analysis packages developed by the SIERRA Solid Mechanics Team of Sandia National Laboratory. The finite element analysis and L_2-projection are conducted using the Adagio production code [139]. Adagio is a Lagrangian, three-dimensional, implicit code for the analysis of solids and structures. It is suitable for problems with large deformations and material nonlinearities; the mapping of internal state variables is applicable for such problems. Furthermore, large systems may be analyzed with the parallel computing environment present in Adagio. The logarithmic mapping, interpolation scheme, exponential mapping, push forward, and pull back procedures are implemented in separate research codes, which are currently under development. They are built upon the Intrepid library, which is a package included in the Trilinos project [140], an open source, object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems. The Intrepid library contains tools for compatible discretization of PDEs. Meshing is performed with the CUBIT geometry and mesh generation toolkit [141].

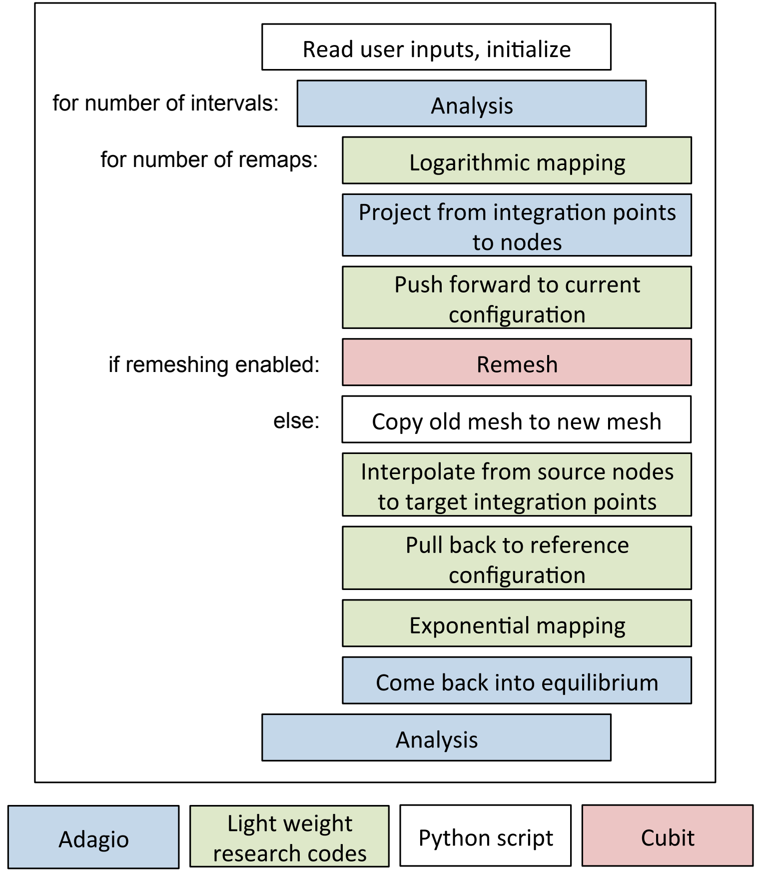


Figure 2.2: Flowchart of Python script for analysis of large deformation process with mapping internal state variables

In this work, we perform a series of numerical students to gain an understanding of the Lie-group interpolation scheme for practical applications. The nature of the studies is to perform a kind of "stress test" on the mapping procedure to examine how it performs in even the most difficult circumstances. It is computational untenable to perform such studies by hand because dozens of analysis codes need to be executed for each instance of the remapping procedure. Therefore, we automated the mapping procedure through a single Python script. All of the analysis modules are called from a single script that is easily executed by the end user.

Two versions of the script were developed. The first version does not include any remeshing, rather the source mesh is simply copied to the target mesh. By removing the remeshing step, the mapping procedure can be examined in detail without interference from changes to the finite element domain. The second version of the script includes remeshing as part of the procedure. Both versions are used in the numerical investigations detailed in Section 2.4. A flowchart of each script is shown in Figure 2.2 and the complete scripts are presented in Appendix A.

The user specifies a few inputs to the scripted procedure:

- Number of analysis intervals
- Number of mapping procedures between each interval
- End time (start time is assumed to be 0.0)

- Time step
- Number of processors

As an example, the user could specify that analysis is performed from time t_1 to t_2 over 5 intervals, and after each interval the problem is remeshed and internal variables mapped to the new mesh 3 times. Since the number of intervals and mapping procedures is specified by the user, the criteria for remeshing is not built in to the script and it is assumed that an external criterion will be applied to determine when the mesh needs to be reconstructed.

Post processing modules

First, visualization of the deformed geometries and fringe plots was performed with Paraview, a multi-platform data analysis and visualization application. Paraview can analyze extremely large data sets and using distributed memory computing resources.

Several post processing scripts were developed as part of this work to quantify/visualize the results on the investigations. One set of post processing scripts involved modifying the input and output files for the analysis modules. The files utilize the Exodus II data model [142], which is a finite element data file format. The format offers a common database for multiple application codes, such that new files do not need to be written for each utility. For example, in this work we use the Exodus II files with Adagio, Cubit, and Paraview, in addition to the light weight research codes. We wrote several Python scripts and utilized the Exomerge tool [143] to modify and visualize specific quantities of interest. Exomerge is provides functions for reading, manipulating, and writing Exodus II files. It is built upon a Python wrapper around the Exodus II API functions. One type of Python script we developed read two exodus files (using Exomerge), took the difference, and wrote the results to a new Exodus II file. This allowed us to visualize the differences between two types of analyses, two steps of the same analysis, before and after the mapping procedure, etc. Since these scripts were written in Python, they were easily incorporated into the script for automated remeshing and mapping, see Appendix A.

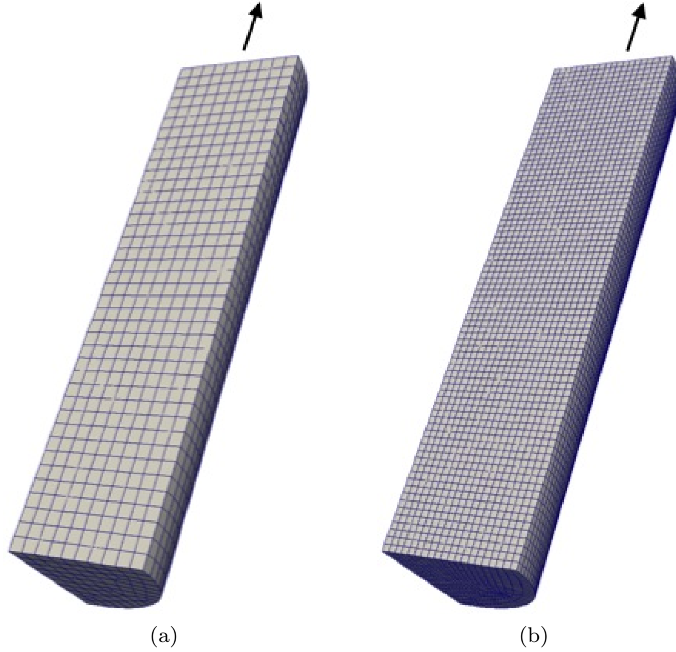
Additional Python scripts were also written to parse Exodus files and gather data for plotting load-displacement curves, dissipation curves, etc. in Matlab. Again these parsing scripts utilize the Exomerge tool. Certain parsing operations could not be performed with the Exomerge tools, so they were done in a stand alone script. For example, the residuals after each mapping and equilibrium steps could not be gathered with a straight forward Exodus II API function call, so parsing was done in a newly developed python script.

2.4 Analysis of the Lie-Group Interpolation Scheme

We aim to understand the Lie-group interpolation scheme discussed previously through a series of numerical studies. The model problem used in the numerical studies is first described. Next, we detail a series of numerical studies and discuss the implications of the results for large deformation analysis when element distortion is handled by remeshing.

2.4.1 Model problem

The numerical studies are performed on a model system of a cylindrical bar with circular-cross section subjected to uniaxial tension via displacement loading. A slight geometric imperfection is introduced in the



(a) (b)

Figure 2.3: (a) coarse and (b) fine meshes of model problem for numerical studies

center of the bar to induce necking. Symmetry boundary conditions allow only 1/8 of the problem to be modeled. The model problem is similar to benchmark studies conducted in other works [122, 124]. Two discretizations are considered, as shown in Figure 2.3, a coarse mesh of 0 elements across the thickness of the base and a fine mesh of 20 elements across the thickness. The problem is assumed to be quasi-static, i.e., without inertial effects. We use a phenomenological plasticity model that captures recrystalization [144]. The specific material parameters are given in [145] and correspond to the precipitation hardened stainless steel, PH13-8Mo H950.

2.4.2 Mapping without remeshing

To focus the analysis on the mapping scheme, we will first eliminate the remeshing and perform the mapping scheme from the deformed mesh to itself. This will provide a baseline to evaluate the scheme without the additional influence of a changing mesh. The coarse mesh is considered for the studies below unless otherwise noted.

2.4.2.1 Study 1: Two intervals with many mapping procedures

The mapping scheme will change the internal variables, which will move the system out of equilibrium. Thus, a fundamental question about any mapping procedure is whether or not it is necessary to come back into equilibrium after mapping. Essentially, this is the same questions as in the split ALE procedure described in Section 2.1. To study this effect we perform our own numerical experiments. We break the analysis into two intervals, $t = 0 \rightarrow 0.125$ and $t = 0.125 \rightarrow 0.25$, performing the remapping procedure several times consecutively without moving forward in time after the first interval. The end time of 0.25 corresponds to a relatively low strain. We intentionally keep the distortion relatively low so that the mapping scheme is kept as isolated as possible.

First we verify that there is no change in the solution when equilibrium is not established after a remap. In this scenario, the equilibrated fields are projected to the nodes then interpolated to the integration points of the same mesh, i.e. the mapping procedure is performed, then the fields are immediately projected to the nodes and interpolated to the integration points once again without coming into equilibrium. We expect there to be no change in the internal state variables in this scenario since once the fields are projected the first time, they remain in the same space throughout the mapping procedure. The maximum equivalent plastic strain is plotted in Figure 2.4. From Figure 2.4(a) it is clear that the internal state variable does not change after the first mapping, as expected.

Next, the equilibrium step is taken at the end of each mapping step. We expect that the mapping procedure will pull the solution out of equilibrium, but would like to quantify if this induced residual is significant. The equilibrium step has an effect on the internal state variable, especially after the first few mapping procedures, however the effect seems to decrease with more maps. The maximum equivalent plastic strain is plotted in Figure 2.4(b). An individual model parameter is not necessarily a reliable measure of the equilibrium of the system, so we instead examine the effect of the mapping procedure with the equilibrium step on the global residual.

The projection and interpolation steps of any mapping procedure essentially have a smoothing effect on the state variables. The proposed scheme ensures that internal state variables remain in their admissible spaces, however the internal variables are nonetheless changed in the mapping procedure, so of course they are pulled out of equilibrium. With the current implementation, it is not possible to achieve equilibrium while minimizing error in the projection. In order to achieve both simultaneously, the implementation would need to reflect the equilibrium requirement in the projection constraint. Due to the computational expense, we do not enforce this at the moment.

Figure 2.5(a) demonstrates that the mapping procedure does in fact pull the problem out of equilibrium, thus a zero velocity step (zero velocity on the far-field boundary but internal degrees of freedom are free to move) is necessary to equilibrate the system. For a visual representation of the residual, we also plotted its magnitude on the deformed shape of the model problem after the first remapping procedure in Figure 2.5(b). The location of the elevated residual is clearly in the areas of largest deformation, where necking is beginning. While there are several orders of magnitude difference between the residual after the first mapping procedure and the equilibrium residual, the residual monotonically decreases to the equilibrium tolerance in only a few steps. Therefore, if the equilibrium step is not taken after mapping, we do not expect that there will be a significant an impact on the convergence when moving forward with the analysis. This finding is in agreement with the split ALE findings.

Educational perspective on script development

During the development and testing of the python script for this study some issues related to the numerical implementation of the analysis modules were uncovered. For a strictly educational purpose, we will discuss the process of identifying the presence of the bugs, locating the source of the bugs, and providing a fix.

Linear solver tolerance After the projection steps, certain checks are performed to ensure variables are in the correct spaces. One such example is on rotation matrices. As discussed in Section 2.2.2 a rotation, $\mathbf{R}$ is a member of $SO(3)$. After logarithmic mapping is performed, the logged rotations are members of $so(3)$, which are skew symmetric matrices. Thus, one of the checks is to ensure that the logged rotations are actually skew symmetric.

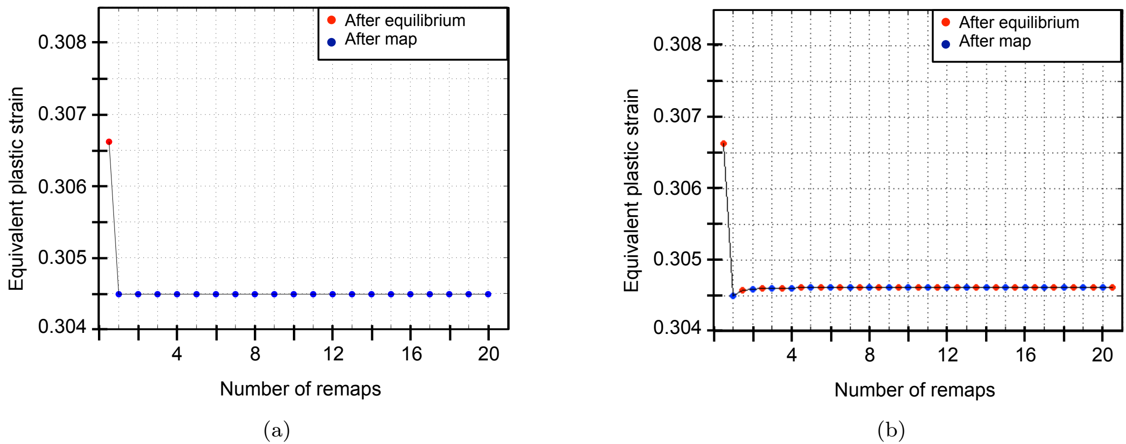


Figure 2.4: Maximum equivalent plastic strain after each mapping procedure (a) without an equilibrium step after the mapping procedure and (b) with an equilibrium step after the mapping procedure

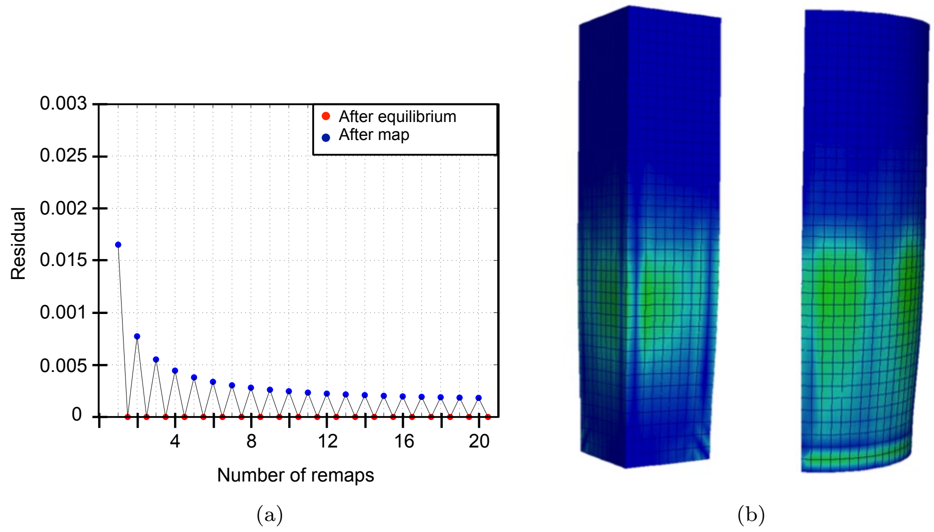


Figure 2.5: (a) Residual after equilibrium and mapping steps (b) Magnitude of residual plotted on deformed shape of model problem after first mapping procedure, the green locations indicate areas of higher residuals, blue are lowest

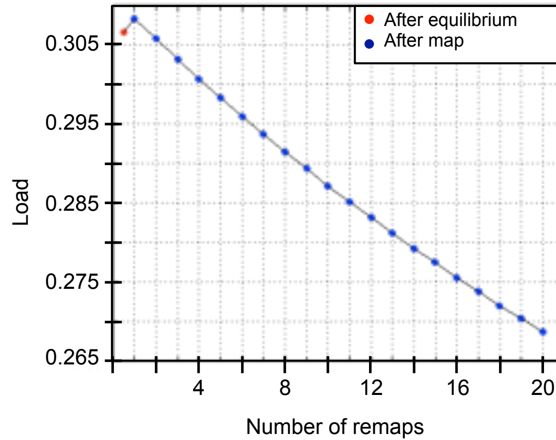


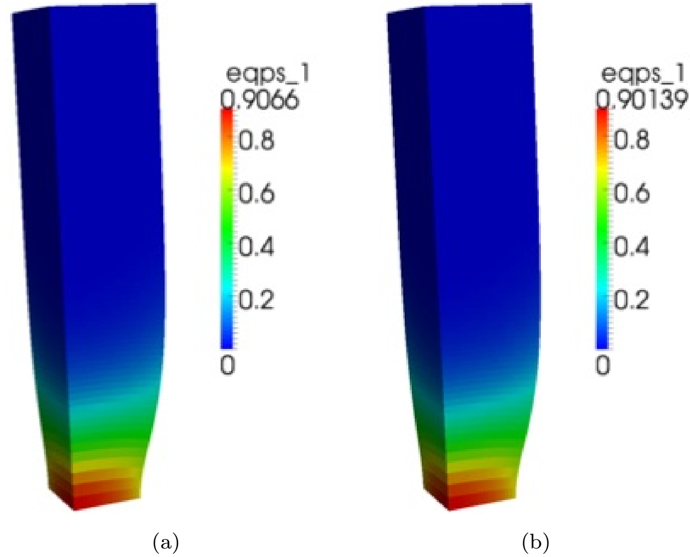
Figure 2.6: Dissipation in internal variable (equivalent plastic strain) with consecutive remaps indicates a problem in one of the analysis modules

Initially, when we attempted to perform multiple remaps without moving forward with the analysis, this check of skew symmetry failed. This issue led to an investigation into of several components of the analysis modules. Eventually, we discovered that the problem was with the inputs defining the linear solver in the analysis module. Originally we attempted to use a parallel iterative solver, however the tolerance on the solver was too loose, which led to very small numerical errors that propagated throughout results such that the skew symmetry was not numerically achieved. Rather than changing the tolerance for each problem individually, we opted to switch to a parallel direct solver, which solved the issue. Another approach would have been to use a non-parallel solver, however we wanted to take advantage of the multi-core machines and reduce computational time, so a parallel solver was most practical.

Dissipation in internal variables between consecutive remaps Once the linear solver bug issue was addressed, we could perform multiple mapping procedures consecutively without moving forward with the analysis. However, a first examination of the multiple remap process indicated numerical dissipation in the internal state variables with each mapping procedure. As discussed above, the mapping procedure should not induce any numerical dissipation after the first time, so we knew this was a problem. The dissipation in the equivalent plastic strain, shown in Figure 2.6, was quite regular, indicating that the same bug was likely occurring at every step. Again, several potential causes of the bug were examined, until we narrowed down that the problem was occurring in the interpolation module. We discovered that the issue was simply incompatible numbering of integration points between the analysis code, Adagio, and the interpolation module. While we only show results for hexahedral elements in this thesis, we were simultaneously doing work on other element types at this time of these studies. It was the process of switching element types that led us to the realization of the numbering discrepancy. Once this issue was resolved, the results in Figure 2.4(a) were obtained.

2.4.2.2 Study 2: Several intervals with one mapping procedure each

In a second study, we break the analysis from $t = 0 \rightarrow 0.25$ into 1, 10, 25, and 100 intervals and perform one mapping procedure after each interval. For example, if the analysis is broken into 100 intervals, then



(a) (b)

Figure 2.7: Equivalent plastic strain at one integration point per element at the end of the analysis, $t = 0.25$, for (a) single interval and (b) 100 intervals

the mapping procedure is performed 99 times, at every 0.0025 time steps. No remapping is performed for the case of one interval. The equivalent plastic strain at one integration per element at $t = 0.25$ is plotted in Figure 2.7. We observe numerical dissipation as the number of intervals increases, however the effect is relatively low, i.e., less than 1% difference in the maximum equivalent plastic strain when the analysis interval is broken into 100 segments with mapping procedures between each one.

The resulting load-displacement curves are shown in Figure 2.8(a) for the coarse mesh and 1, 10, 25 and 100 intervals. Numerical dissipation is evident through softening of the load-displacement curve, especially as the displacement increases. The same studies were conducted on the fine mesh and the load-displacement curve is shown in Figure 2.8(b). While the softening effect is present, it is much less significant than in the coarse mesh, suggesting that the difference may be due to discretization error.

Clearly splitting the analysis period into 100 intervals in completely impractical, but these results give us confidence that even in the most severe case the mapping procedure has a minimal effect on the overall response, provided a suitably fine mesh is used. These studies give us confidence that the mapping procedure will not induce signifiant error, especially when realistic engineering judgement is employed.

2.4.3 Remeshing and mapping

In the next set of studies, we remesh the domain after some deformation has occurred. To ensure we are not adding too many levels of complexity at a time, we apply the load for the same amount of time and use hexahedral elements as in the previous studies. Therefore the only change now is that the mesh is that the domain is remeshed.

Automation of remeshing

The numerical framework for the automated remapping procedure remains largely the same as in the previous studies without remeshing. The only difference is that when each interval includes a remeshing step inside of the mapping procedure. In these studies the time at which remeshing occurs is set *a priori* by the user. This

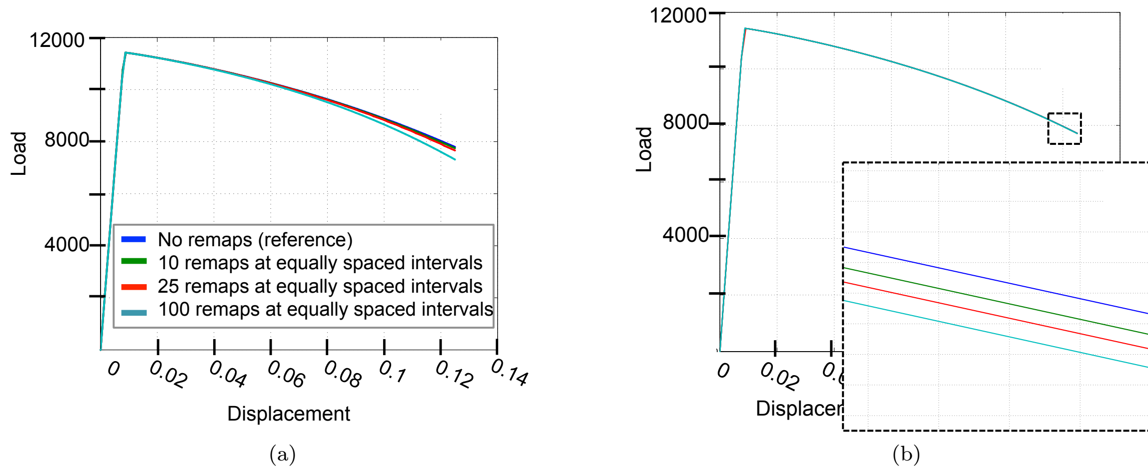


Figure 2.8: Load displacement curve for (a) coarse mesh and (b) fine mesh with no remeshing after each interval

is not consistent with how remeshing would be performed on an actual application. In that case, remeshing would be performed when some measure of mesh quality indicating that the element distortion is significant enough to cause excessive numerical error on the original mesh. Potential metrics could include evaluation of Lo's parameter [146], algebraic metrics based on the element Jacobian [147], element internal angles [148], etc.

For the studies presented in this chapter, the size of the elements remain approximately the same before and after remeshing, but their locations and shapes will change. The deformed shape (Exodus II filetype) is passed to the meshing software, Cubit. We specify the size of the mesh using the length of the ligament on the bottom of the specimen, as shown in Figure 2.9. For the coarse mesh, we specify 10 elements across the bottom ligament, and we specify 20 elements for the fine mesh. We use the same number of elements across the bottom ligaments in each of the coarse and fine meshes when we remesh.

Numerical results

For this set of studies we remesh and perform only one mapping procedure between each analysis interval. Unlike the previous section, we do not investigate the case of several remaps and remeshes without moving forward with the analysis. Due to limited available computational time, we only examine the case of breaking the time interval into 25 segments with a remesh and mapping procedure performed between each segment.

The load displacement curves with and without remeshing on the coarse and fine mesh is shown in Figure 2.10. Consistent with the findings of mapping only, the remeshing and mapping procedure results in numerical dissipation that is more prevalent in the coarse mesh than in the fine.

These series of studies serve as a proof of concept for the Lie group interpolation and variational recovery scheme. Even under the worst case scenarios of frequent remeshing and mapping, the scheme performs well and the global results agree well with the case of a single analysis interval. Therefore when applied to an actual engineering problem where reasonable judgement is used to decide when to remesh and map, we are confident that this scheme will perform well. As a follow up to the studies performed for this thesis, further investigation are warranted and underway as part of an ongoing collaboration between the author and Sandia National Laboratories. Please see Section 6.2.1 for a discussion of future work related to the

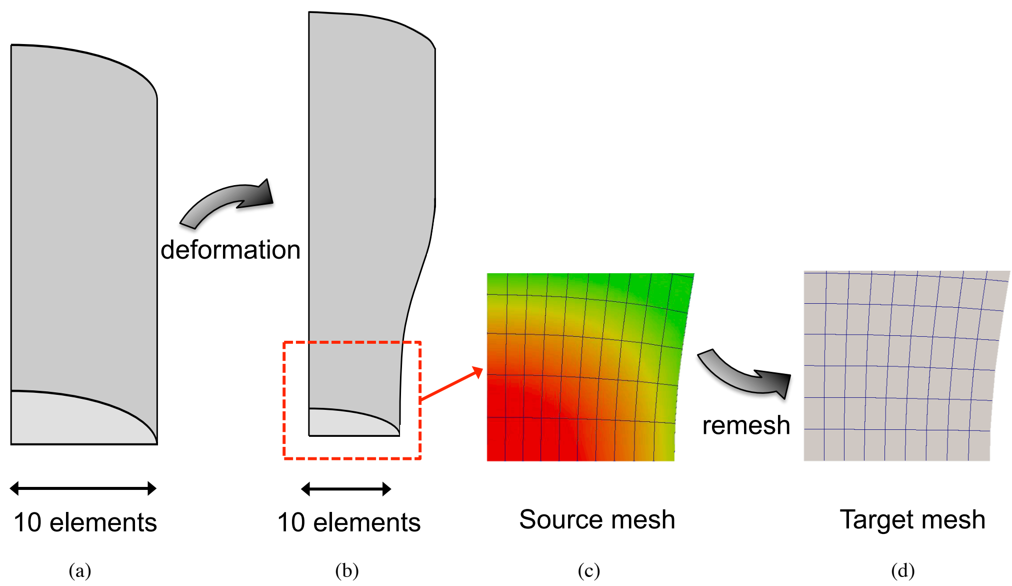


Figure 2.9: Construction of new mesh on model problem (a) Undeformed configuration is meshed with 10 elements across the bottom ligament (b) Deformed configuration to be remeshed with 10 elements across the bottom ligament (c) Zoom in of deformed source mesh (d) Deformed domain is remeshed

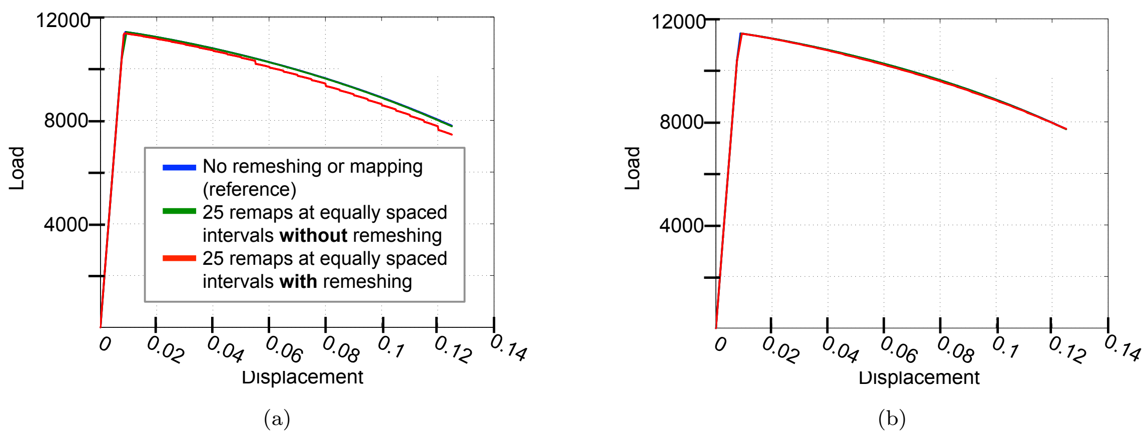


Figure 2.10: Load displacement curve for (a) coarse mesh and (b) fine mesh with remeshing after each interval

Lie-group interpolation and variational recovery scheme.

Chapter 3

Towards reduction in mesh bias using adaptive splitting of polygonal finite elements

In the previous chapter we addressed issues that may arise when large deformations occur before the onset of fracture, i.e. creation of new surfaces. Now, we shift our focus to modeling the actual dynamic fracture process. In this chapter, we present a new technique to reduce mesh bias and improve fracture patterns using the inter-element cohesive zone model and adaptive splitting of polygonal finite elements.

Before proceeding to the proposed method, we provide an overview of inter-element cohesive fracture in Section 3.1. This section will also be relevant to future chapters of this dissertation; we provide as much detail as is needed here and go into more as needed in later chapters. Next, we present the proposed method using polygonal elements and adaptive element splitting in Section 3.2. Then, in Section 3.3 we evaluate the geometric qualities of the polygonal element meshes with and without adaptive splitting. Finally, the performance of polygonal element splitting for cohesive dynamic fracture is investigated with the benchmark compact compression specimen (CCS) numerical example in Section 3.4.

3.1 Inter-element cohesive fracture

The cohesive zone model approach (see Section 1.2.9) can be incorporated into a number of numerical frameworks, e.g. extended finite elements, generalized finite elements and inter-element cohesive. In this work we limit our attention to the inter-element cohesive zone model approach, in which the cohesive elements are only present at the bulk finite element boundaries. The method has been used with much success to capture complex crack patterns in dynamic simulations [93, 149–153]. In the following section, we broadly review the numerical implementation of the inter-element approach and discuss some implications that motivate the use of polygonal elements.

3.1.1 Mathematical formulation of dynamic cohesive fracture

Consider the case of an arbitrary domain, Ω, that is subjected to surface tractions, $\boldsymbol{T}$, along the boundary, Γ, and cohesive tractions, $\boldsymbol{T}^{\text{coh}}$, along the fractured surfaces, Γ^{coh}, illustrated in Figure 3.1. The equations of motion are derived using the principle of virtual work, which states

$$\int_{\Omega} \delta\mathbf{u}^T \rho \ddot{\mathbf{u}} + \delta\boldsymbol{\varepsilon}^T \boldsymbol{\sigma} d\Omega \quad = \quad \int_{\Gamma} \delta\mathbf{u}^T \mathbf{T} d\Gamma + \int_{\Gamma^{\text{coh}}} \delta\boldsymbol{\Delta}^T \boldsymbol{T}^{\text{coh}} d\Gamma \tag{3.1}$$

where ρ is the mass density, $\boldsymbol{\sigma}$ is the stress, $\mathbf{u}$ is the displacement, $\ddot{\mathbf{u}}$ is the second time derivative of the displacement (i.e. acceleration), $\delta\boldsymbol{\Delta}$ is the virtual separation across the fractured surfaces, $\delta\mathbf{u}$ is the virtual displacement, and $\delta\boldsymbol{\varepsilon}$ is the virtual strain. The stress and strain measures used in this work are the 2nd Piola-

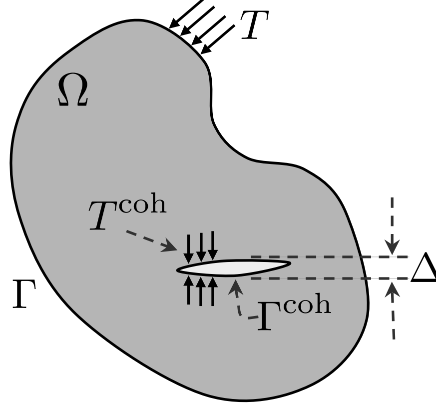


Figure 3.1: Arbitrary domain with applied boundary conditions

Kirchhoff stress tensor and the Green-Lagrange strain tensor, respectively. The effects of body forces and damping are neglected. The continuous problem is converted to a discrete problem via the standard Galerkin approximation. Finite deformations are taken into account by means of the total Lagrangian formulation, in which all quantities are measured with respect to the initial configuration [154].

To evaluate the dynamic response of a system, we use an explicit time integration scheme: the well known explicit central difference method; which is a subset of the classical Newmark method [155]. After discretizing Equation 3.1, we can write the general form of the elasto-dynamic equation as

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{f} \tag{3.2}$$

where damping is not considered, $\mathbf{M}$ is the mass matrix, $\mathbf{K}$ is the stiffness matrix, $\mathbf{u}$ is the displacement vector, $\ddot{\mathbf{u}}$ is the second time derivative of the displacement (acceleration), and $\mathbf{f}$ is the vector of external force (the external loads and cohesive forces are lumped into the $\mathbf{f}$ term). The general form of the Newmark method for the solution of Equation 3.2 is

$$\mathbf{M}\ddot{\mathbf{u}}_{\mathrm{n}+1} + \mathbf{K}\mathbf{u}_{\mathrm{n}+1} = \mathbf{f}_{\mathrm{n}+1} \tag{3.3}$$

$$\mathbf{u}_{\mathrm{n}+1} = \mathbf{u}_{\mathrm{n}} + \Delta t \dot{\mathbf{u}}_{\mathrm{n}} + \frac{\Delta t^2}{2}\left[(1-2\beta)\,\ddot{\mathbf{u}}_{\mathrm{n}} + 2\beta \ddot{\mathbf{u}}_{\mathrm{n}+1}\right] \tag{3.4}$$

$$\dot{\mathbf{u}}_{\mathrm{n}+1} = \dot{\mathbf{u}}_{\mathrm{n}} + \Delta t\left[(1-\gamma)\,\ddot{\mathbf{u}}_{\mathrm{n}} + \gamma \ddot{\mathbf{u}}_{\mathrm{n}+1}\right] \tag{3.5}$$

where the subscript denotes the time step. The choice of parameters γ and β determine the accuracy and stability conditions of the method. Unconditional stability is met when $2\beta \leq \gamma \leq 2$. If $\gamma \geq 1/2$ and $\beta < \gamma/2$, then the method is conditionally stable and the time step must satisfy $\omega \Delta t \leq \Omega_{\mathrm{crit}}$, where Ω_{crit} is the critical sampling frequency, given by

$$\Omega_{\mathrm{crit}} = (\gamma/2 - \beta)^{1/2} \tag{3.6}$$

when damping is not present. We will further examine the time step requirement for dynamic fracture in Section . The central-difference method is achieved when $\gamma = 1/2$ and $\beta = 0$. Notice that it is only conditionally stable. The mass matrix is diagonal in order for the central difference method to be explicit, hence we apply a standard mass lumping technique in which the diagonals of the consistent mass matrix are calculated and scaled [156, 157]. This method is appropriate for the dynamic fracture simulation because

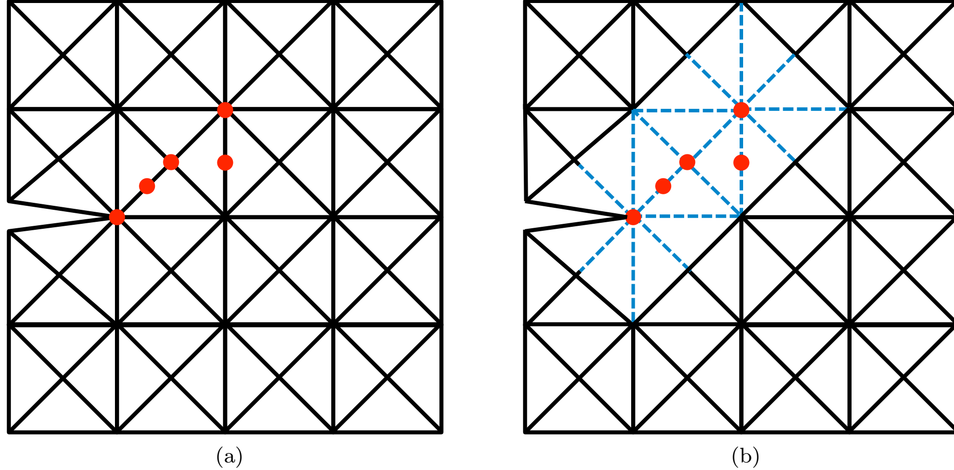


Figure 3.2: Schematic of insertion of extrinsic cohesive elements using a stress-based approach (a) Nodes with principle stress greater than 90% of the cohesive strength are flagged - shown in red (b) Normal and tangential tractions are computed at the facets adjacent to the flagged nodes - shown in dashed blue. If the averaged normal or tangential stress is greater than the respective cohesive strength, then a cohesive element is inserted along the facet

the resolution of many frequencies is required, therefore a sufficiently small time step is necessary. The high frequency behavior is captured thanks to the small time step requirement and a linear system does not need to be solved at every time step so computational time is saved.

3.1.2 Extrinsic cohesive model

Two types of cohesive models exist: the intrinsic and extrinsic approach. In the former case, the cohesive zone elements are inserted into the domain *a priori* and their activation criterion is internal to the formulation. In the extrinsic cohesive zone model, which is the approach utilized in the remainder of this dissertation, the criteria to insert or activate the cohesive element is external to the constitutive model.

The activation criteria employed in this dissertation is a stress-based criteria. It is implemented as follows: we first compute stresses at the integration points, then extrapolate to the nodes. Any node with principle stress greater than 90% of the cohesive strength is flagged for further investigation, as shown in the schematic in Figure 3.2(a). Normal and tangential components of the tractions along the facets adjacent to the flagged nodes are computed by averaging the contribution of each node. If the normal or tangential traction is greater than the normal or tangential cohesive strength, respectively, then the cohesive element is inserted. This approach has been used in several studies [88, 89, 101, 153, 158], and similar strength-based approaches have also been used with success [151, 159–161]. Once activated, the interface is governed by a traction-separation relationship.

The use of an extrinsic cohesive zone model requires on-the-fly mesh modification during the simulation. The present work uses the consistent topological data structure, TopS [162, 163], to efficiently represent the finite element mesh. More details on the TopS data structure can be found in the next chapter, where it is used heavily.

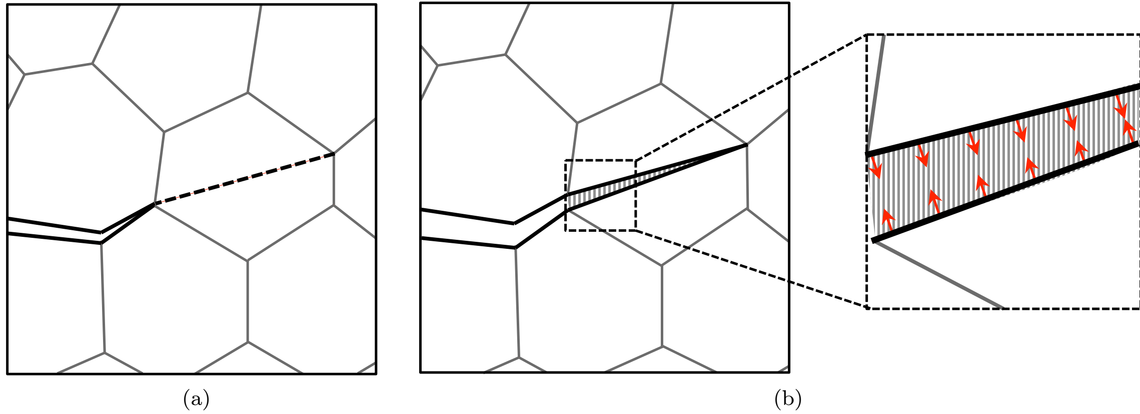


Figure 3.3: Schematic of a cohesive element being inserted at the new facets of split polygonal element. (a) The element ahead of the crack tip meets the criteria to be split, indicated by the dashed line. (b) The polygonal element is split and a cohesive element is inserted at the new facets; the zoomed in region shows a schematic of the cohesive tractions provided by the PPR model

3.1.3 PPR cohesive constitutive models

When cohesive elements are inserted, a traction separation relation is activated at the interface. The cohesive constitutive model utilized in this work is the Park-Paulino-Roesler (PPR) potential-based model [164]. This model is derived from a potential, is suitable for mixed mode failure, adheres to necessary boundary conditions and constraints, and provides flexibility for implementation with other numerical models. The adaptive insertion of the cohesive element in a split polygonal element is illustrated on the CVT mesh in Figure 3.3 (a), and a schematic of the PPR tractions is shown in Figure 3.3 (b).

The details of the PPR model will not be shown here as they are well explained in [164] and are not the focus of this work. Rather, we will briefly discuss the its construction and physical modeling capabilities. In a potential-based model, the normal and tangential tractions are determined by taking the derivative of the potential with respect to the normal opening and tangential opening, respectively. The PPR potential was constructed such that the tractions adhere to the following physically relevant boundary conditions:

- Complete normal failure occurs when the normal separation is greater than the final normal crack opening width or tangential separation is greater than the final tangential conjugate opening
- Complete tangential failure occurs when the the tangential is greater than the final tangential crack opening width or normal separation is greater than the final normal conjugate opening
- The area under the traction curve equals the fracture energy in either mode
- The maximum tractions equal the cohesive strengths in either mode
- The maximum tractions occur at the critical opening displacements

The user has control over several parameters to tune the model to the specific material and configuration of the problem. These parameters are:

- ϕ_n and ϕ_t - normal and tangential fracture energies, respectively
- $\sigma_{\max}$ and $\tau_{\max}$ - normal and tangential cohesive strengths, respectively

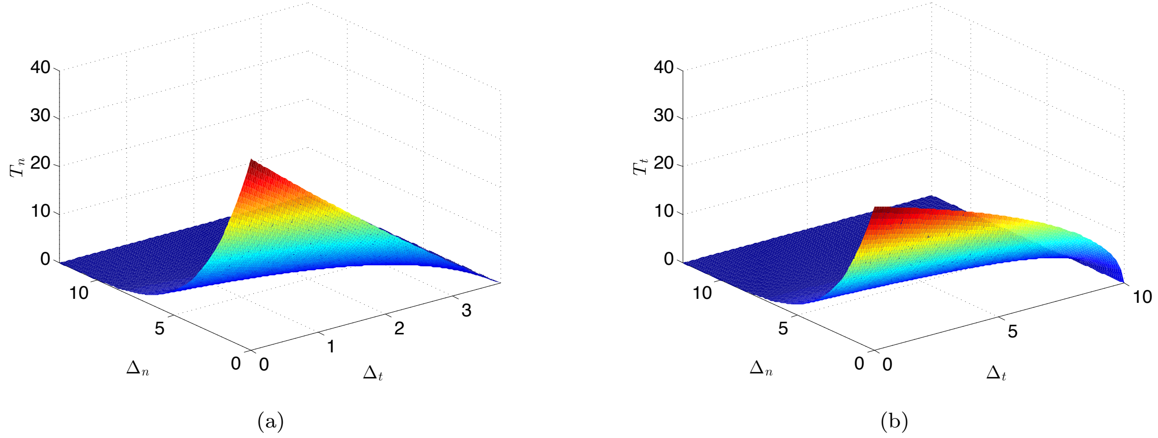


Figure 3.4: Traction-separation relations for (a) extrinsic model, normal opening ($\phi_n = 100N/m$, $\sigma_{max} = 40MPa$, $\alpha = 5.0$), (b) extrinsic model, tangential opening ($\phi_t = 200N/m$, $\tau_{max} = 30MPa$, $\beta = 1.5$)

- α and β - softening shape parameters in the normal and tangential directions, respectively

The normal and tangential traction-separation are plotted in Figure 3.4 for the parameters given in the figure caption. As mentioned previously, in this example, we assume that the mode I and mode II fracture properties are the same. To achieve the nearly linear softening curve, we utilize shape parameters of 2.

Additional models for unloading/reloading and contact may be implemented without loss of the PPR properties. In this work, we utilize a penalty stiffness to handle the contact condition, i.e., when the the normal cohesive traction is negative. The unloading/reloading relationship is uncoupled, meaning that the unloading in the normal direction is independent of the unloading in the tangential direction. The traction separation relations with linear unloading occurring at 20% are shown in Figure 3.5 for the extrinsic case.

Furthermore, one could conceivably include additional physics or even adapt the PPR model as a simulation is evolving. In the former case, additional physics such as fatigue could be incorporated. Rather than using a Paris relation approach to modeling fatigue, internal, rate-dependent damage variables that describe degradation under cyclic loading could be incorporated into the cohesive model. Alternatively, the PPR could be modified to account from friction by including internal slip rate variables in the description of the model. The failure behavior of a material may change after the onset of fracture or after some amount of damage is accumulated. Thus, one could also experiment with adapting the PPR model on the fly.

One consequence of potential-based models is that the work to fracture is independent, i.e., for a given separation vector, the work of separation is independent of the loading history [165, 166]. However, the numerical implementation of the PPR model results in a path-dependent work to fracture. This is achieved because the model is only valid inside an area of influence, which is determined based on input parameters and is not specified by the user. Outside this zone of influence, the tractions are not derived from the potential, rather they are set to 0 in order to satisfy the boundary conditions. Thus the model is not strictly potential-based, and path-dependance can be achieved.

3.1.4 Implementation of inter-element cohesive fracture

A flowchart of the code at a given time step is shown in Figure 3.6. First the nodal displacements are calculated using data from the previous step according to Equation 3.4. Steps 2-5 are related to the insertion

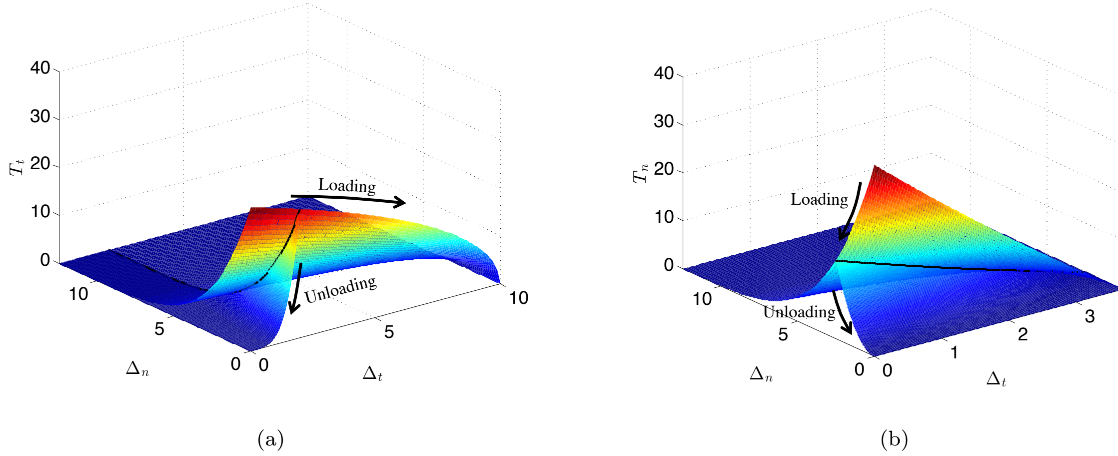


Figure 3.5: Traction-separation relations with linear unloading at 20% for (a) extrinsic model, normal opening ($\phi_n = 100N/m$, $\sigma_{max} = 40MPa$, $\alpha = 3.0$), (b) extrinsic model, tangential opening ($\phi_t = 200N/m$, $\tau_{max} = 30MPa$, $\beta = 5.0$)

of cohesive elements. The check for insertion of cohesive elements is performed at a user-defined interval. If the current step is an increment of the interval, then the stresses are computed at the nodes. Using the nodal stresses, the stresses along the facets are then computed. If the stress along the facet is greater than the cohesive strength, then a cohesive element is insert at that face. Next the internal force vector, i.e. $\mathbf{K}\ddot{\mathbf{u}}$, is computed in step 6. The cohesive force vector is then computed in step 7 using the PPR traction-separation relation. Next the nodal velocity and acceleration are computed according to Equations 3.5 and 3.3, respectively. Energy quantities are computed for post-processing needs in step 10. Next, the boundary conditions are updated. Finally, quantities of interest are printed for post-processing needs before moving on to the next time step.

3.1.5 Mesh bias in inter-element cohesive fracture

The inter-element cohesive zone modeling approach is known to suffer from mesh bias, as cracks are forced to propagate along element boundaries [4, 92, 153]. Unstructured meshes are preferred for fracture applications because they possess no preferential path directions (i.e. isotropic), which can result in more realistic crack patterns than their structured counterparts [167, 168]. Methods to create arbitrary meshes for crack propagation problems have been studied extensively. Ingraffea, Wawrzynek and coworkers, for example, have developed an algorithm to generate meshes on arbitrary domains for fracture applications [169] and have recently presented a parallel technique for mesh generation [170]. However, structured meshes have their benefits too: they can be systematically generated and may possesses a hierarchical subdivision, making them suitable for refinement schemes. Hence, researchers have used structured meshes with special properties or additional adaptivity operators in an attempt to gain the benefits of a structured mesh with the isotropy of an unstructured mesh. The pinwheel mesh possesses the isoperimetric property, meaning that, as the element sizes tend to zero, it is able to represent any arbitrary crack path [4, 171]. However, applications of the pinwheel mesh are limited because the convergence of crack lengths and angles exists only in the limit sense and may not be achieved with practical ranges of mesh size. The 4k mesh with adaptivity operators proposed in [88] is a practical alternative to the pinwheel mesh. Hence, we will use it for comparison with

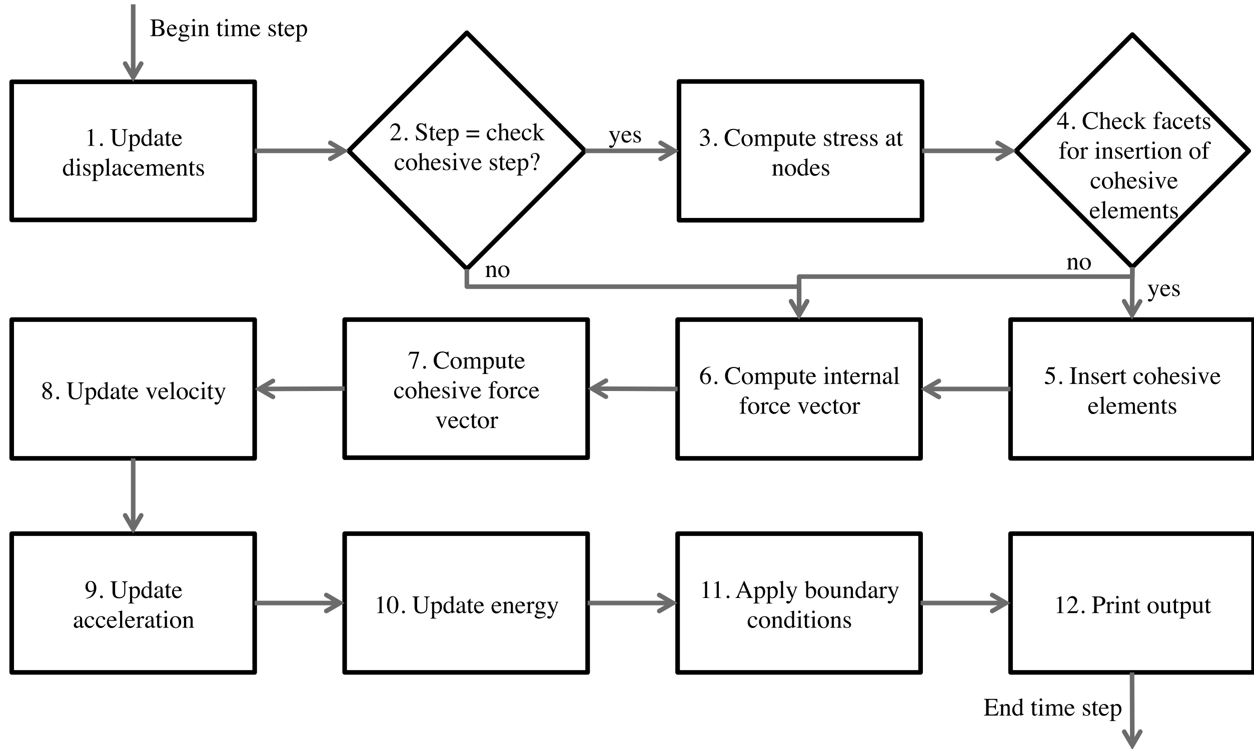


Figure 3.6: Flowchart of extrinsic, cohesive, dynamic fracture code

the present approach.

To reduce bias in the 4k mesh, Paulino et al. [88] proposed the nodal perturbation and edge-swap topological operators, which utilize the TopS data structure. Nodal perturbation, shown in Figure 3.7, results in an unstructured geometry by randomly perturbing the internal nodes. The edge-swap operator reduces undesirable crack patterns by supplying all internal element vertices with the same number of potential crack directions, shown in Figure 3.8.

3.2 Proposed method to reduce mesh bias in inter-element cohesive fracture

In order to reduce the mesh bias inherent to the inter-element cohesive model approach, we propose the use of unstructured polygonal finite element with element splitting employed when and where needed. Polygonal elements are generated by two different methods, but splitting methodology and numerical implementations are identical.

3.2.1 Polygonal element mesh generation

Two discretizations are investigated, the Centroidal Voronoi Tessellation (CVT) mesh which is a relatively regular discretization and a random polygonal mesh, which is similar to the near Maximal Poisson Disk Sampling (near-MPS) mesh.

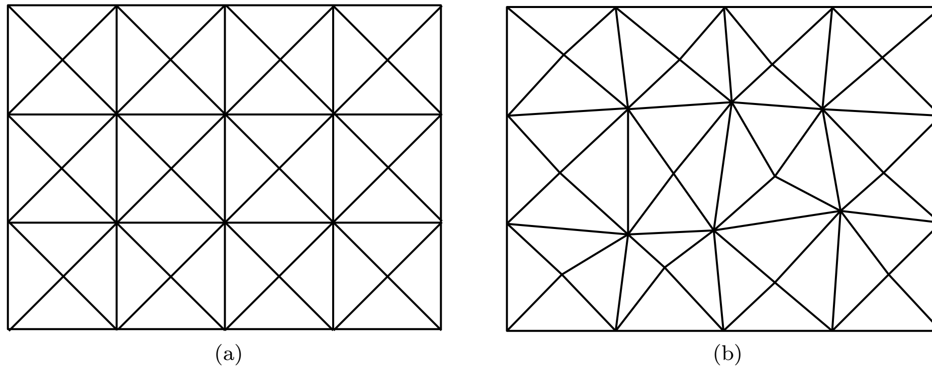


Figure 3.7: 4k mesh (a) without and (b) nodal perturbation

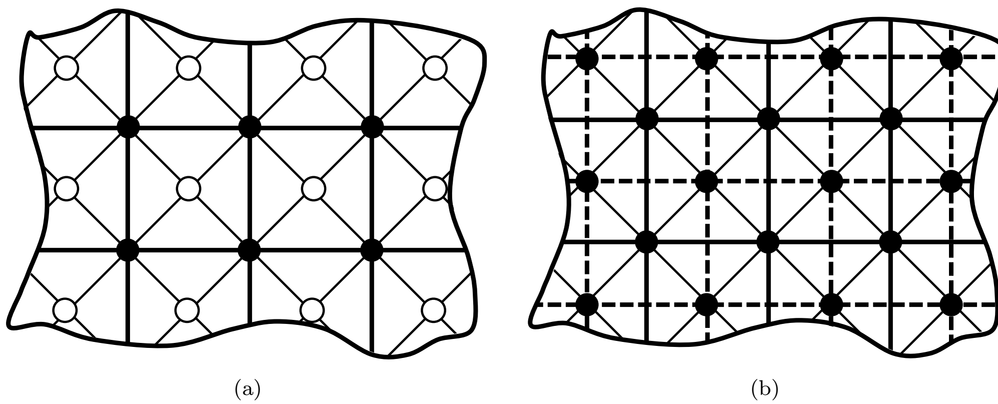


Figure 3.8: Potential crack path directions on a 4k mesh where the solid nodes have eight potential crack directions and the white nodes have only four directions. (a) Bold edges can be swapped; (b) After the edge-swap operator is applied the dashed edges become available crack directions

Generation of CVT meshes

The domains are discretized into Centroidal Voronoi Tessellation (CVT) convex polygons with the PolyMesher MATLAB code [148]. The PolyMesher code generates CVT meshes on arbitrary closed domains via the following procedure:

1. User specifies the number of elements, N
2. N mesh seeds are randomly placed inside the domain (signed distance functions are used to determine if the seed is inside or outside of the domain)
3. Voronoi cells are generated for each of the seeds
4. The centroids of the Voronoi cells are computed
5. The generating seeds of the Voronoi cells are replaced by the cell centroid
6. Steps 3-6 are repeated until the distance between the generating seed and centroid of the cell are within a prescribed tolerance

The CVT results in a discretization in which there is little variation in the range of element sizes. We will compare this type of discretization with a polygonal mesh generated using randomly placed seeds, in which there is much more variation in element edge size.

Generation of random meshes

It is hypothesized that the CVT mesh will induce some mesh anisotropy [172] , thus we examine a completely random polygonal element mesh. The PolyMesher MATLAB code is modified to generate a random mesh with some restrictions for element regularity. Rather than using Lloyd's algorithm to move the mesh seeds to the centroid of the Voronoi cell, the seeds are only constrained to be at least a minimum distance away from all other seeds in the mesh. The procedure is detailed below:

1. User specifies the number of elements, N
2. Seeds are inserted one at a time inside the domain (signed distance functions are used to determine if the seed is inside or outside of the domain)
3. The distance between the seed inserted in step 2 and all other existing seeds are computed
4. If the seed inserted in step 2 is closer than a prescribed distance to any other seed, it is rejected, otherwise it is kept
5. Steps 2-4 are repeated until N seeds are present
6. Voronoi cells are generated for each of the seeds

Similar meshes [168] have been utilized in dynamic fracture applications [172].

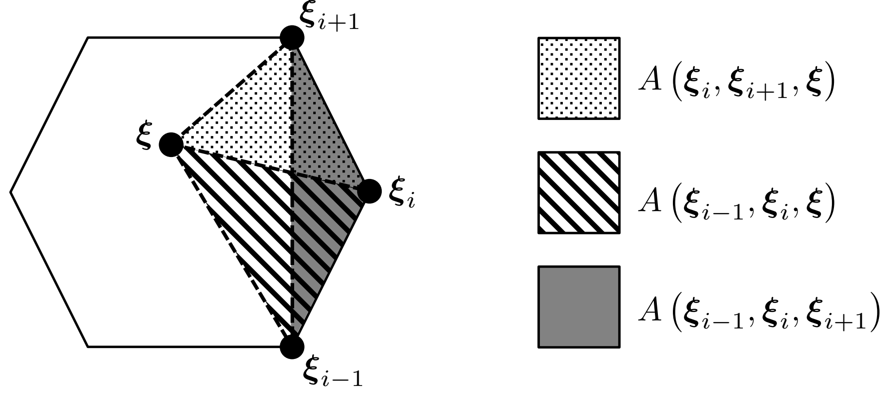


Figure 3.9: Triangular areas used to define α_i

3.2.2 Shape functions for polygonal elements

By the virtue of the Galerkin approximation, the same functions are used for the test and trial functions. Field quantities are determined at any point in the domain via interpolation from nodal quantities. The separation across the cohesive elements is interpolated using linearly varying shape functions.

In this work, we use Wachspress shape functions [173] that satisfy the desirable properties of shape functions [174]. Isoparametric mapping is employed to map quantities from a regular n-gon in the reference coordinate system to the arbitrary convex polygon in the physical coordinate system. The Wachspress shape functions [174] for a reference n-gon are expressed as

$$N_i\left(\boldsymbol{\xi}\right) = \frac{\alpha_i\left(\boldsymbol{\xi}\right)}{\sum_{j=1}^{n} \alpha_j\left(\boldsymbol{\xi}\right)} \tag{3.7}$$

where the interpolants α_i are of the form

$$\alpha_i\left(\xi\right) = \frac{A\left(\boldsymbol{\xi}_{i-1}, \boldsymbol{\xi}_i, \boldsymbol{\xi}_{i+1}\right)}{A\left(\boldsymbol{\xi}_{i-1}, \boldsymbol{\xi}_i, \boldsymbol{\xi}\right) A\left(\boldsymbol{\xi}_i, \boldsymbol{\xi}_{i+1}, \boldsymbol{\xi}\right)} \tag{3.8}$$

and A denotes the area of the triangle made up of the points in its arguments. For example, Figure 3.9 shows a reference hexagon with shaded triangular regions to indicate the areas used in the computation of α_i in the shape function of the node at $\boldsymbol{\xi}_i$ evaluated at point $\boldsymbol{\xi}$. For a detailed discussion of the construction and implementation of the Wachspress shape functions for finite element applications, the reader is directed to [175]. The Wachspress shape functions are generated once, before the fracture simulation begins, and the necessary quantities (i.e. shape functions and their derivatives evaluated at the integration points) are stored and accessed on an as-needed basis.

3.2.3 Adaptive splitting through polygonal elements

Crack patterns on polygonal element meshes are quite restricted because each node typically only has a few facets emanating from it; therefore there are limited possible directions for a crack to propagate from a crack tip node. In fact, each node of the CVT mesh is connect by only three facets on average. To circumvent the limited number of potential crack directions, we propose an element splitting operator that increases the potential crack directions at each crack tip, as illustrated in Figure 3.10.

In addition to allowing the element to be split with any node, we also investigate a case where the nodes

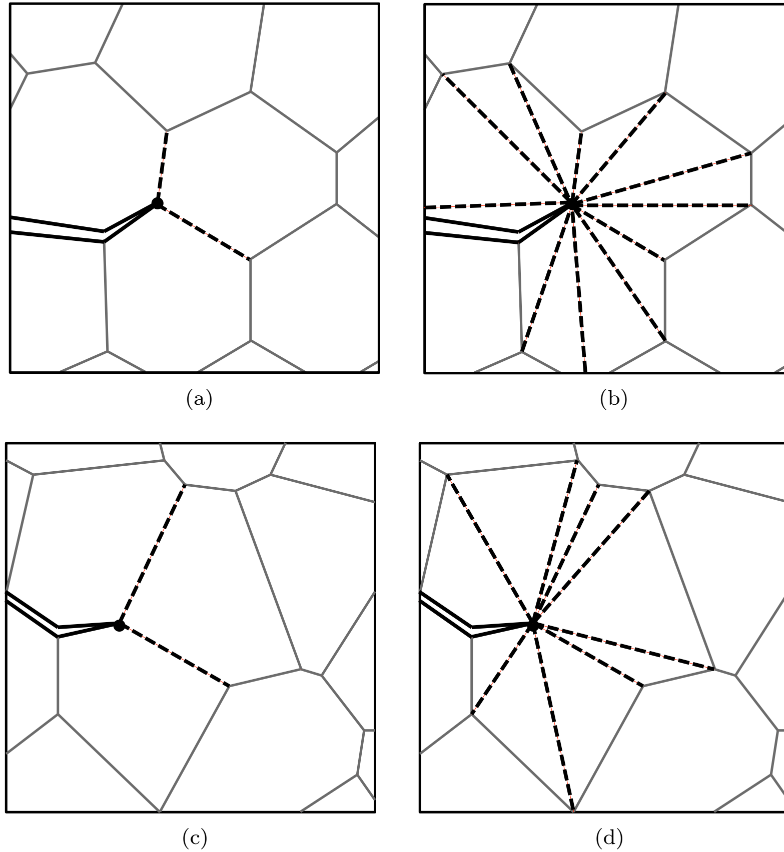


Figure 3.10: Schematic of a crack in a polygonal mesh, where crack faces are illustrated with solid black lines, the crack tip is indicated by a black circle, and potential crack paths are shown as dashed lines. (a) Potential crack directions on plain CVT mesh (b) Potential crack directions on CVT mesh with adaptive splitting employed (c) Potential crack directions on plain near-MPS mesh (d) Potential crack directions on near-MPS mesh with adaptive splitting employed

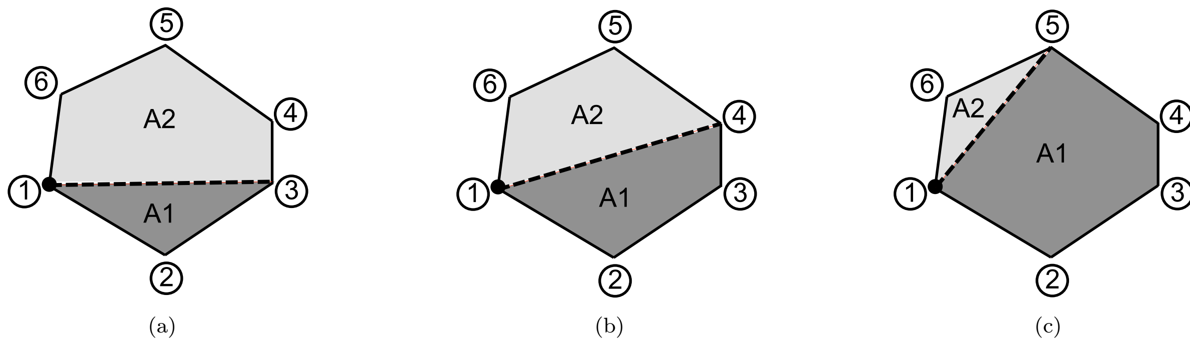


Figure 3.11: Schematic of potential new elements that would result from splitting element between node 1 and (a) node 3, (b) node 4 and (c) node 5. The configuration shown in (b), where the element is split with node 4, minimizes the difference between the areas, A1 and A2, of the resulting new elements; once an element is split, the resulting two elements cannot be split again

with which an element may be split are restricted. Since element edge size is directly related to the critical time step for the explicit dynamic time integrator, this approach may help avoid additional reduction of the already small time step. More specifically, we chose the node that minimizes the difference in the areas between the two newly created elements. For example, in Figure 3.11, node 1 is flagged and the element could be split with either node 3, 4 or 5, as shown in parts (a), (b), and (c), respectively. The resulting areas are indicted by the shaded regions and labeled, A1 and A2, in Figure 3.11. The element is only allowed to be split with node 4 because it produces two new elements whose difference in area is minimized when compared to splitting with node 3 or 5.

Use of the TopS data structure ensures that the splitting operation is computationally efficient (i.e. node and element identification and connectivity are automatically updated). The original polygonal element is removed then two new polygonal elements are inserted using appropriate functions available in the TopS API. Once an element is split, the resulting two elements cannot be split again. Thus, recurring element splitting is not allowed in the present work.

3.2.3.1 Activation of element splitting

At the same time that the insertion of cohesive elements is checked by the stress-based criterion, bulk elements are also evaluated for splitting. Each element adjacent to a flagged node is checked for splitting (recall that a node is flagged if its principal stress greater than 90% of the cohesive strength). Starting from the flagged node, all adjacent nodes are visited and tractions along the facet that would result between the crack tip node and adjacent node is computed. The element will actually be split if the average normal or tangential traction along the facet that would connect the splitting nodes is greater than the normal or tangential cohesive strength, respectively. If multiple potential facets meet this criteria, then the one with the highest traction is selected.

3.2.3.2 Transfer of field variables

When an element is split, the original polygonal element is deleted from the model and two new elements are inserted. Linear polygons are used in this work, so no new nodes are created when elements are split; therefore, it is not necessary to interpolate new nodal values. If higher order elements were used, mid-side

Table 3.1: Cases for geometric study comparison

Base Mesh	Adaptivity Operators	Restrictions
4k	none	n/a
4k	edge swap	n/a
perturbed 4k*	none	n/a
perturbed 4k*	edge swap	n/a
CVT polygonal	none	n/a
CVT polygonal	element splitting	minimize area difference
CVT polygonal	element splitting	none
Random polygonal	none	n/a
Random polygonal	element splitting	minimize area difference
Random polygonal	element splitting	none

* A value of 30% perturbation was applied to the 4k meshes
in accordance with the recommendation in Paulino et al. [88].

nodal quantities would need to be interpolated from existing nodes. Moreover, since all of the polygons in the original mesh are convex, then the polygons which result from splitting the element along its nodes are also convex. Thus, we do not encounter issues of degenerate finite elements.

When the two new elements are inserted their element attributes are initialized and stiffness matrices are computed. The location of integration points on the new elements are recomputed such that integration is done in a consistent way among all elements (split polygons and unsplit polygons).

In the current work, we use a bulk material model in which the state of the material is completely characterized by the displacement field, and cohesive elements are never split. Thus, there is no need to map history-dependent internal state variables from the original set of integration points to the new set. However, if a more complex model were used, a technique to map internal state variables, such as that described in Chapter 2, would be necessary.

3.3 Geometric studies on crack representation

The goal of the extrinsic inter-element cohesive modeling approach is to capture crack patterns that are now known a priori, thus, the finite element mesh should be able to resolve any crack pattern. In [4], researchers define spatial convergence of the mesh as the ability of the mesh to represent any crack path in the Hausdorff distance sense and in the length of the crack sense. They investigate the 'isoperimetric' pinwheel mesh, which meets the convergence criteria as the element edges approach zero, through geometric and numerical studies. In this work however, we are concerned with meshes of practical size, thus we perform geometric studies on length, angle and Hausdorff distance on the different mesh types shown in Table 3.1.

The 4k mesh is widely used in finite element analysis as they are easily generated and lend themselves well to mesh refinement and coarsening. To reduce bias in the 4k mesh, Paulino et al. [88] proposed the nodal perturbation and adaptive edge-swap topological operators. Nodal perturbation, shown in Figure 3.12, results in an unstructured geometry by randomly perturbing the internal nodes. The edge-swap operator reduces undesirable crack patterns by supplying all internal element vertices with the same number of potential crack directions, shown in Figure 3.13.

As indicated in [88], the 4k mesh performs as well or better than the isoperimetric mesh for elements of practical size. As we show in the following section, the polygonal meshes perform as well or better than the 4k mesh, which leads us to conclude that while the polygonal meshes have not been shown to have the

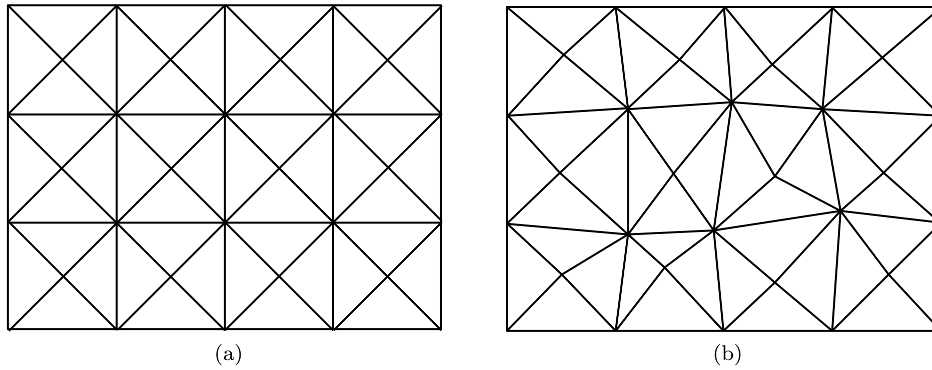


Figure 3.12: 4k mesh (a) without nodal perturbation and (b) with nodal perturbation

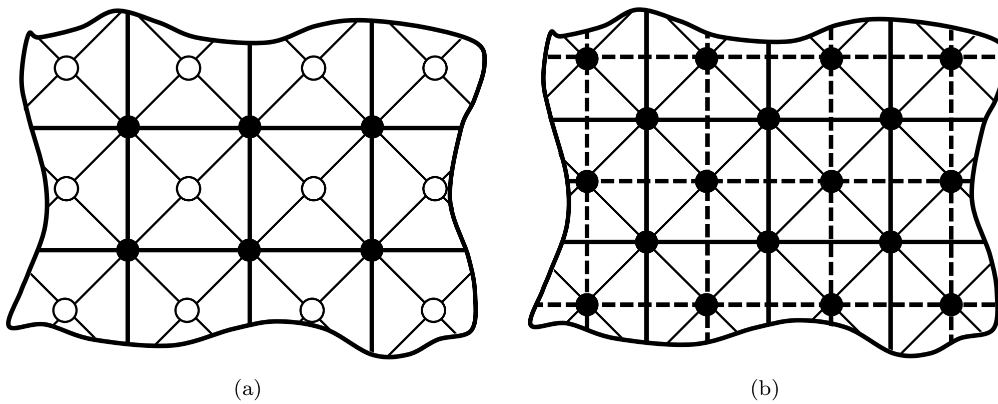


Figure 3.13: Potential crack path directions on a 4k mesh where the solid nodes have eight potential crack directions and the white nodes have only four directions. (a) Bold edges can be swapped. (b) After the edge-swap operator is applied the dashed edges become available crack directions

isoperimetric property, they are suitable for fracture applications in which the crack direction is now specified a priori.

3.3.1 Crack length studies

First, we investigate the influence of the mesh influence on representing a crack of a certain length. The fracture energy, E_f, required to generate a crack is directly proportional to the crack length, i.e. $E_f = L_c G_c$, where L_c is the crack length and G_c is the energy release rate. Hence, the length of the crack should not be altered by the finite element representation such that the fracture energy artificially increases [4].

3.3.1.1 Methodology

We perform studies similar to those of [176]. The shortest distance measured along the finite element edges between the start point and end point is computed using Dijkstra's algorithm [177] then compared to the euclidean distance. Square domains of 2 units by 2 units, where the center node is at point (0, 0) are used for all studies. The start point is (0, 0) and the end point is chosen as the closest node to the point given by $(r \cos\theta,\ r \sin\theta)$, for $\theta = 0^\circ$ to 180° where $r = 1$. The domain was selected to be a square in order to fairly compare the 4k and polygonal element meshes. However, to ensure all paths from $\theta = 0^\circ$ to 180° are approximately the same, the shortest path algorithm is terminated when the path reaches a node nearest to the edge of a circle of radius 1.

For illustrative purposes, Figure 3.14 shows the paths of minimum distance for $\theta = 71.6^\circ$ on a CVT mesh of a circular domain with and without the restricted element splitting as measured with Dijkstra's algorithm. The distance without element splitting is over 23% longer than the Euclidean distance, while the distance with splitting is less than 4% longer. It should be emphasized that the metric of this study is only the length of the path and not the distance *between* the finite element path and a straight line. In section 3.3.3 we perform a study on Hausdorff distance to address this issue. We also emphasize that the example shown in Figure 3.14 is for illustrative purposes only; the domain used for the studies was rectangular and the mesh is finer than that shown in the figure.

The deviation in crack length, η, is quantified as the percent difference between the length of the path created by the finite element edges and the euclidean distance. The deviations for angles $\theta = 0^\circ$ to 180° are shown in Figure 3.15. A study of mesh refinement revealed that the deviation did not decrease significantly as the element size decreases, hence we chose a refinement level that allowed for computationally efficient calculations. Meshes of 10,000 elements are used for all studies. We also report mesh size according to a non-dimensional metric, λ, defined as [176]

$$\lambda = \frac{\frac{1}{N}\sum_{i=1}^{N} h_i}{L_e} \tag{3.9}$$

where L_e is the euclidean distance between the start and end points, N is the total number of edges in the mesh, and h_i is the length of edge i. For the meshes used in these studies $\lambda \approx 1/80$. The polygonal meshes and perturbed 4k meshes are random, so we perform the geometric studies on several meshes.

3.3.1.2 Results

As illustrated in Figure 3.15(a), even with the nodal perturbation and edge swap operators, all of the 4k meshes suffer from mesh induced anisotropy [176]. Meshes that do not suffer from mesh anisotropy have

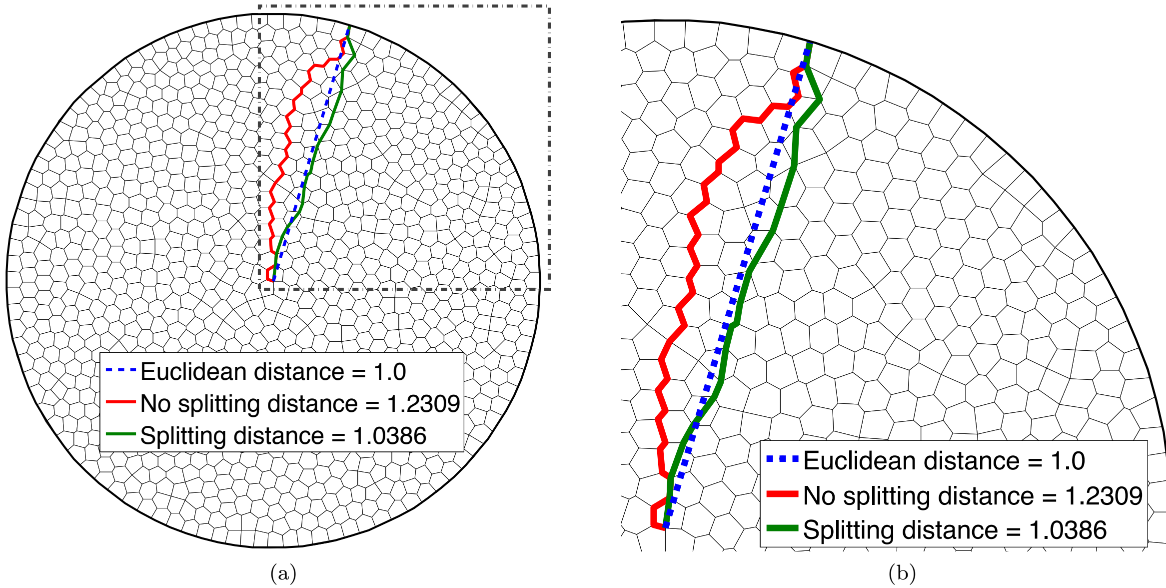


Figure 3.14: The shortest distance between the start point (0, 0) and end point $(\cos(71.6°), \sin(71.6°))$, measured along the finite element edges vs. the euclidean distance for a polygonal mesh with and without restricted element splitting. (a) Full domain, (b) zoom in of region inside the dotted, grey box in (a). The circular domain shown here is for illustrative purposes only, the studies were conducted on rectangular domains with an inscribed circular boundary, as described in the text

approximately equal deviations for all crack mesh angles. Furthermore, it is desirable for the deviation to also tend to zero such that the energy is not larger than expected.

The random polygonal meshes and CVT meshes without element splitting do not appear to suffer from mesh anisotropy, as expected, but the deviation is significantly higher than it is in any of the 4k mesh variations, as illustrated in Figures 3.15(c)-(d) and shown in Table 3.2. When restricted element splitting is employed, the deviation is reduced to an average value in the range of that of the 4k mesh with nodal perturbation and edge swap. Notice that the random polygonal element mesh has a lower average deviation for the cases of no splitting and restricted splitting, however the CVT meshes perform better in this geometric study when element splitting is unrestricted. This is because the very small edges that are permitted on the random polygonal element mesh, allowing the crack to propagate to any vertex may not actually improve the crack direction very much. The edge size on the CVT mesh, however, is regulated by means of the mesh generation algorithm, so each additional vertex provides more improvement to the crack path than in the case of the random mesh. Furthermore, we hypothesize that the random mesh is not well suited for element splitting because of the potential presence of small edges; we attempt to avoid these due to the restrictions upon the critical time step. We will investigate whether these elements are viable in the next section.

It it worth noting that both polygonal elements mesh with restricted splitting achieve a mean deviation near that of the k-means mesh studied by Rimoli and Rojas [176] (see Figure 12 in [176]), and a comparable deviation to the conjugate-directions mesh when unrestricted splitting is allowed. However, as stated earlier, since the element splitting is done adaptively, the computational storage costs are less than what they would be for either the k-means or conjugate-directions meshes.

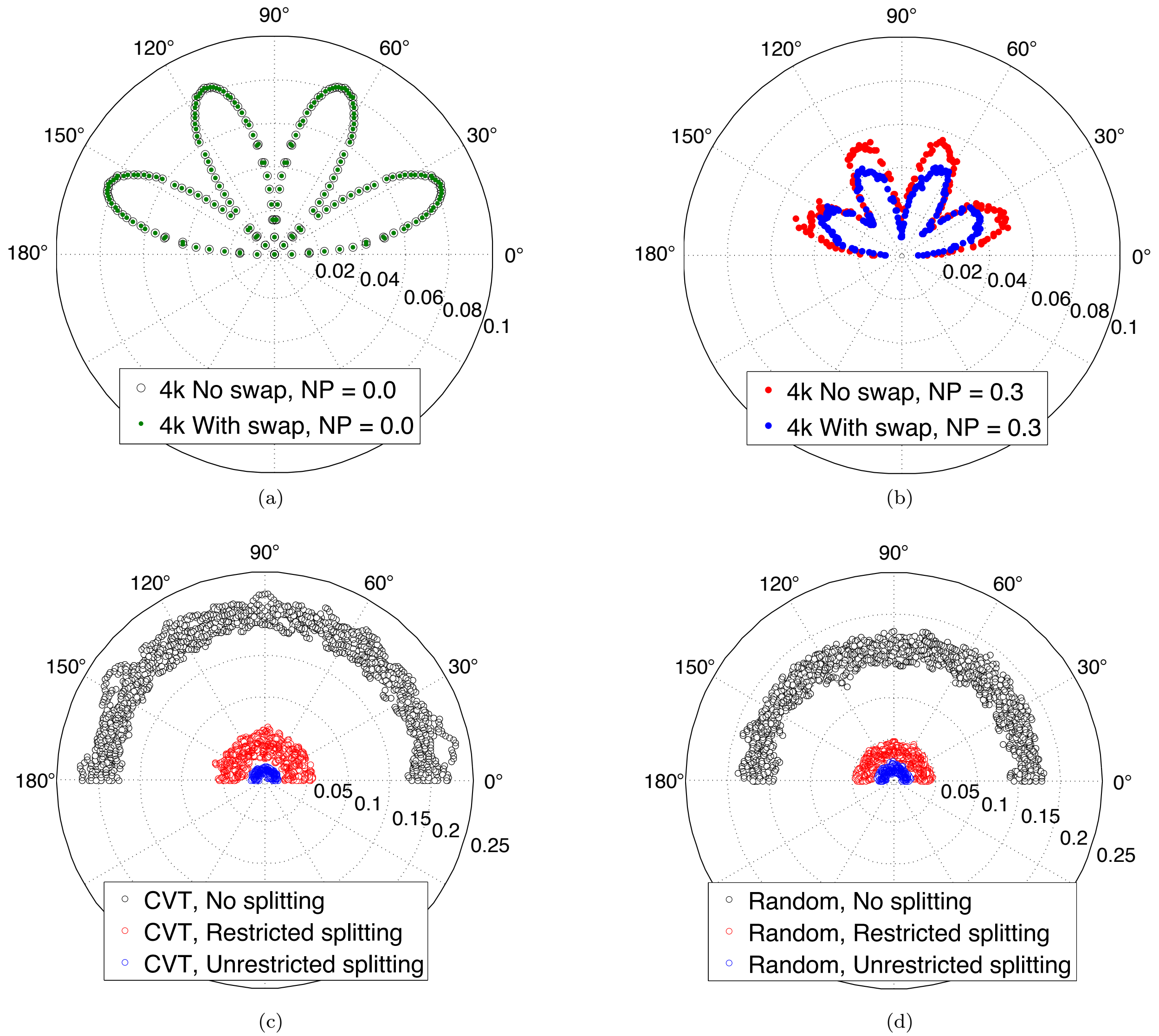


Figure 3.15: Comparison of mesh deviation, η, for meshes with $\lambda \approx 1/80$. Deviations of (a) 4k meshes with and without edge swap, (b) 4k meshes nodal perturbation factors of 0 and 0.3 with and without edge swap, (c) CVT meshes without element splitting, with restricted element splitting and with unrestricted element splitting, (d) random polygonal meshes without element splitting, with restricted element splitting and with unrestricted element splitting

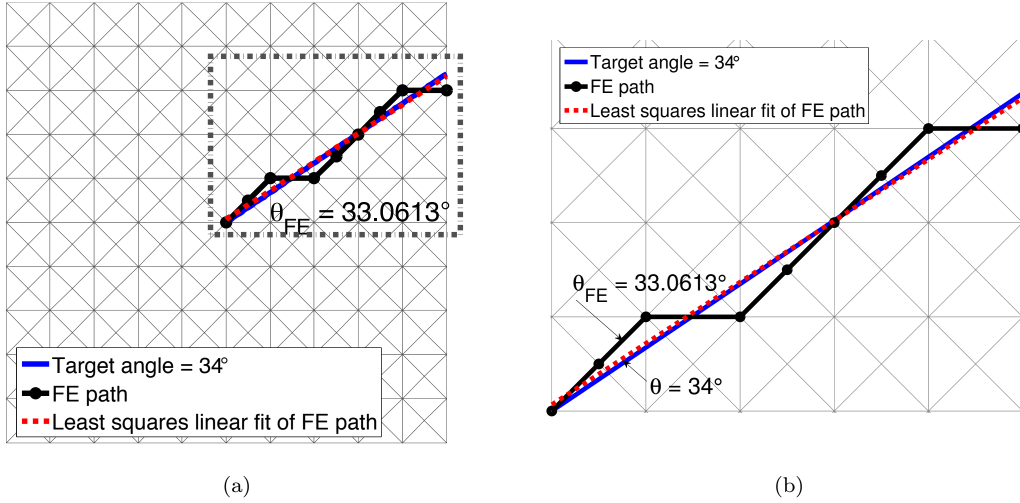


Figure 3.16: Comparison of an arbitrary angle and geometric angle, $\theta = 34°$, for (a) unperturbed 4k mesh with and without edge swapping, (b) perturbed 4k meshes with and without edge swapping

3.3.2 Crack angle studies

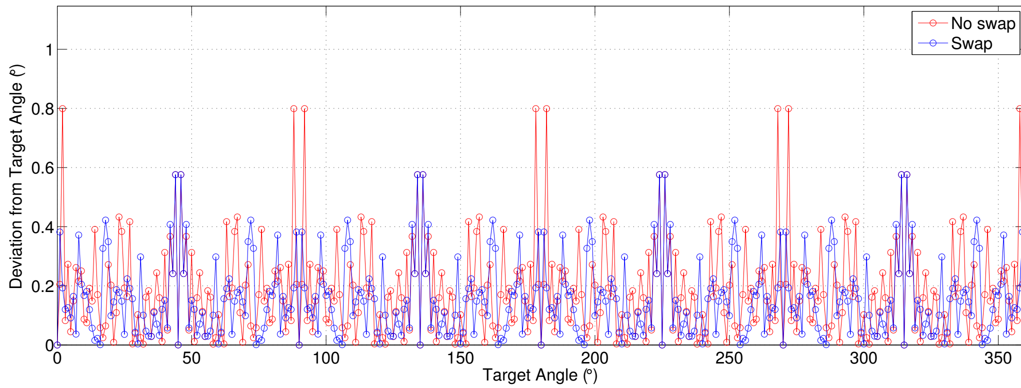
Ideally, a finite element mesh would be able to represent cracks at any angle without any predefined crack direction. While it is impossible to allow completely arbitrary crack propagation in the present framework, we hypothesize that the unstructured polygonal element meshes will have lower error in representing an arbitrary crack angle than a structured mesh. This error in crack angle is investigated next.

3.3.2.1 Methodology

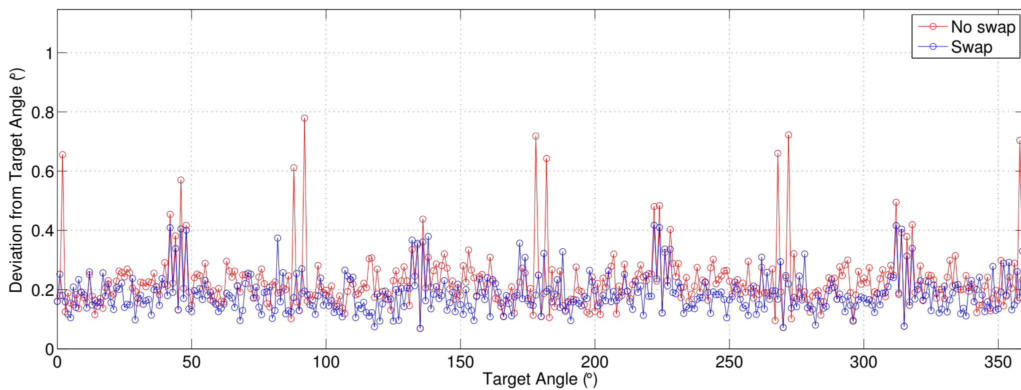
We measure the difference between an arbitrary angle θ and the corresponding geometric angle, θ_{FE}. Again, the studies are performed on square domains of 2 units by 2 units with 10,000 elements, where the starting node is at point (0, 0). The procedure to determine the geometric angle, θ_{FE}, is as follows: from the starting location, a local search is performed over the edges adjacent to that node. The angle formed by the start node and the node at the opposite end of the edge is computed. The edge that results in the angle closest to the target angle is selected and the current location is updated to the node at the opposite end of the selected edge. Then the same procedure is applied until the end point is reached; the end point is the first node whose distance to the starting point is greater than or equal to the target radius of one. The series of nodes and edges from the starting point to the boundary represents the path from which the geometric angle is determined. A least squares linear fit is applied to the path and the angle is obtained. For illustrative purposes, Figure 3.16 shows the paths obtained for the case of $\theta = 34°$ for a coarse, unperturbed 4k mesh without edge swapping.

3.3.2.2 Results

The absolute value in the deviations from target angles 0° to 359° are plotted in Figure 3.17(a) for the 4k mesh with and without element splitting, in Figure 3.17(b) for the perturbed 4k mesh with and without element splitting, in Figure 3.18(a) for the CVT polygonal mesh with and without element splitting and in Figure 3.18(b) for the random polygonal element mesh with and without element splitting. A single mesh



(a)



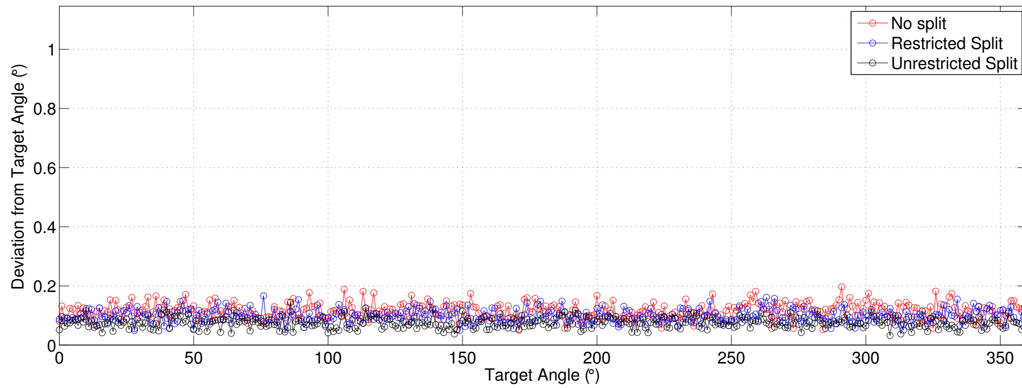
(b)

Figure 3.17: Absolute value of difference between geometric angle and target angle for radial paths from from 0° to 359° at 1° increments for (a) 4k without nodal perturbation, (b) 4k with nodal perturbation; results for the perturbed mesh are averaged from from 100 meshes

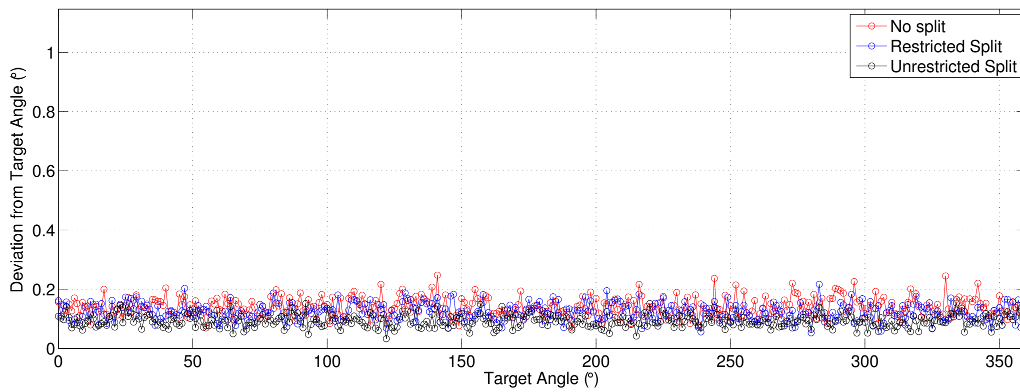
was used to calculate the deviations for the unperturbed 4k mesh, while the absolute values in the angle differences in the randomly perturbed 4k and polygonal meshes results are averaged over 100 meshes. All of the 4k meshes display mesh bias, where the lowest deviation is for angles of 0°, 45°, 90°, etc., and highest for angles of $0° \pm 1°$, $45° \pm 1°$, $90° \pm 1°$, etc. The benefit of the edge swap operator is clear when comparing Figures 3.17(a) and 3.17(b) at 0°, 90°, 180°, and 270°; however, bias still exists. Conversely, the polygonal meshes with and without splitting display no mesh bias, and their average deviations are much lower than those in the 4k meshes. A summary of the deviations from the target angle are shown in Table 3.2. While the 4k meshes have minimum deviations as low as or lower than the polygonal meshes, the range of deviations for the polygonal meshes is smaller than that of the 4k meshes.

3.3.3 Hausdorff distance studies

The last set of geometric studies examines the distance between the finite element path and a straight line. A low Hausdorff distance will give the appearance that the finite element path is close to the mathematical path, which is desirable for fracture applications. The same quantity was investigated in [4, 88].



(a)



(b)

Figure 3.18: Absolute value of difference between geometric angle and target angle for radial paths from from 0° to 359° at 1° increments for (a) CVT polygonal mesh, (b) Random polygonal mesh; results are averaged from from 100 meshes

3.3.3.1 Methodology

The paths obtained for the crack angle study are also used to investigate the Hausdorff distance. Given two curves, P and Q, in the same two-dimensional space, the Hausdorff distance, H, is the maximum of the minimum distances from all points on curve P, denoted p, to all points on curve Q, denoted q. It is defined as [178]

$$H(P, Q) = \max\left(h(P, Q), h(Q, P)\right) \tag{3.10}$$

where

$$h(P, Q) = \max\left[\min_{p \in P}\left[\operatorname*{dist}_{q \in Q}(p, q)\right]\right]. \tag{3.11}$$

3.3.3.2 Results

The bias in the 4k meshes has already been established, so here we report the occurrence of Hausdorff distances in the histograms in Figures 3.19(a) - 3.19(d). Again, one mesh was used for the unperturbed 4k mesh case, while 100 were used for the perturbed 4k and polygonal mesh cases. The occurrences are normalized such that the area of each histogram is one. The distribution of the Hausdorff distances for the perturbed meshes are smoother than those of the unperturbed 4k mesh; however, there is some tradeoff when comparing the perturbed and unperturbed 4k meshes. Higher Hausdorff distances occur in the perturbed case, especially when edge swap is not employed: 95.6% of the Hausdorff distances for the unperturbed case are below 0.02, but only 42.5% of the distances for the perturbed case fall below this threshold. There is improvement when edge swap is used: 100% of the Hausdorff distances for the unperturbed case and 98.5% for the perturbed case are lower than 0.02.

The striking difference comes in the comparison of the 4k meshes to the polygonal meshes. As illustrated in Figure 3.19(c) the Hausdorff distances for CVT mesh with no splitting, restricted splitting, and unrestricted splitting are lower and the distribution is tighter than for the 4k meshes. The random polygonal element meshes behave similarly to the CVT meshes in this study, as shown in Figure 3.19(d). Although, because of the presence of small edge, the distribution for each case is larger than those of the CVT meshes.

3.3.4 Summary of geometric studies

Table 3.2 summarizes the findings of three geometric studies for the four types of meshes investigated (i.e. 4k, perturbed 4k, CVT polygonal, random polygonal) with two types of adaptivity operators (i.e. edge swapping on the 4k meshes and element splitting on the polygonal meshed). The polygonal meshes with element splitting are shown to be superior to 4k meshes with nodal perturbation and adaptive edge-swap operators in geometric studies. However, because we hypothesize numerical difficulties associated with the critical time step when small edges are present, we cannot yet make a conclusion as to the applicability of the element in a dynamic fracture setting. Thus, in the next section we evaluate the performance of each type of polygonal element on a benchmark dynamic fracture problem.

3.4 Numerical Investigations

In this section, we perform numerical fracture simulation investigations of the two types of polygonal element discretizations with and without element splitting. We use the benchmark Compact Compression Specimen (CCS) problem for which published numerical and experimental results exist.

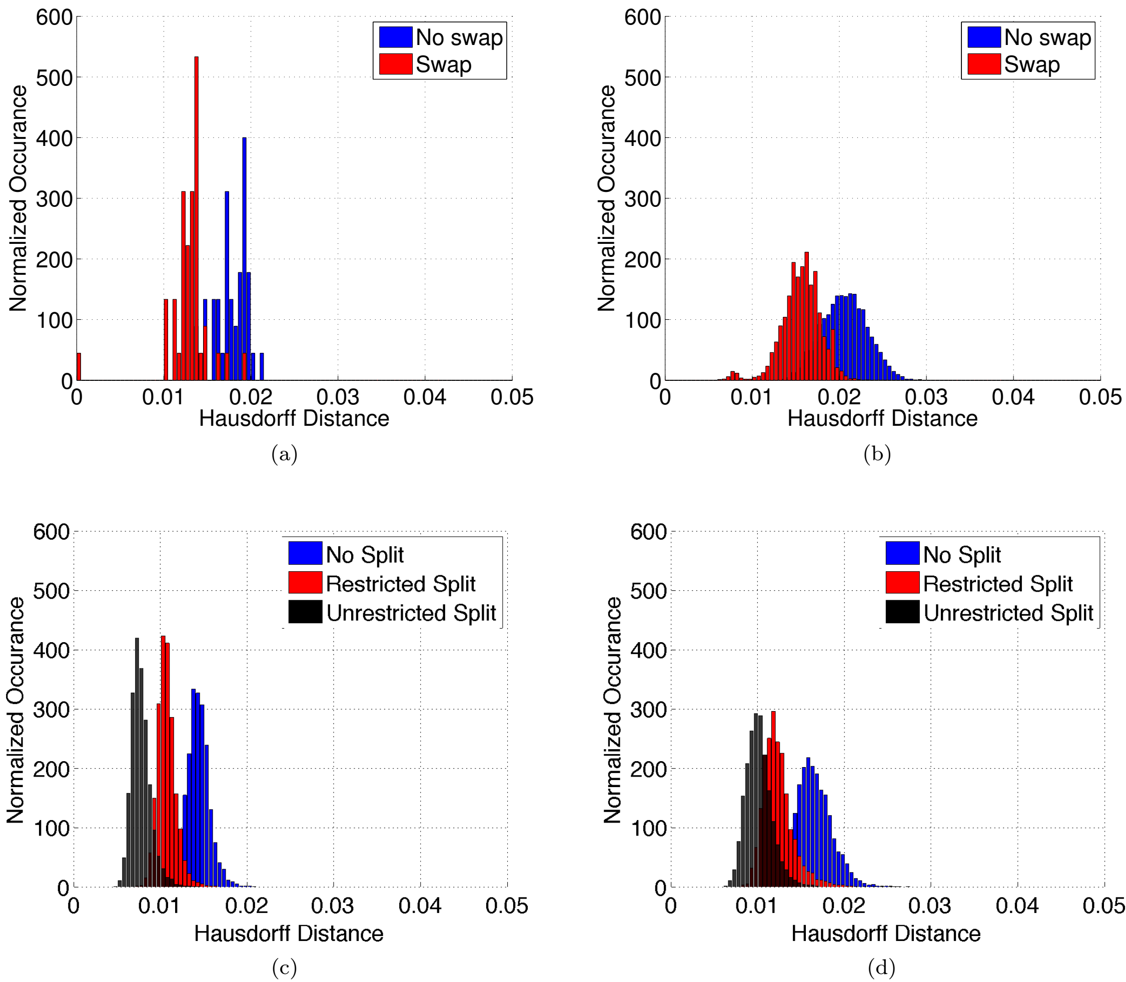


Figure 3.19: Histograms of Hausdorff distances for radial paths from 0° to 359° at 1° increments for (a) unperturbed 4k mesh with and without edge swapping, (b) perturbed 4k meshes with and without edge swapping and (c) CVT polygonal mesh with and without element splitting, (d) random polygonal mesh with and without element splitting; results for the 4k discretization are obtained from one mesh, while the results from 100 meshes are averaged for each the perturbed 4k and polygonal discretizations

Table 3.2: Summary of geometric studies meshes with $\lambda \approx 1/80$

Mesh Type			**Dev. in crack length**		**Dev. in crack angle**		**Hausdorff Distance**	
Base Mesh	**Adaptivity Operators**	**Restrictions**	**Avg.**	**Std dev**	**Avg.**	**Std dev**	**Avg.**	**Std dev**
4k	None	n/a	0.0543	0.0250	0.18°	0.16°	0.017	0.003
4k	Edge swap	n/a	0.0543	0.0250	0.16°	0.13°	0.013	0.003
Perturbed 4k	None	n/a	0.0372	0.0120	0.23°	0.10°	0.020	0.003
Perturbed 4k	Edge swap	n/a	0.0302	0.0100	0.19°	0.06°	0.016	0.002
CVT	None	n/a	0.200	0.003	0.11°	0.03°	0.014	0.001
CVT	Element splitting	Minimize area difference	0.043	0.002	0.10°	0.02°	0.011	0.001
CVT	Element splitting	None	0.012	0.0005	0.08°	0.02°	0.008	0.001
Random polygonal	None	n/a	0.163	0.003	0.14°	0.03°	0.017	0.002
Random polygonal	Element splitting	Minimize area difference	0.037	0.001	0.12°	0.03°	0.012	0.002
Random polygonal	Element splitting	None	0.014	0.0008	0.09°	0.02°	0.010	0.001

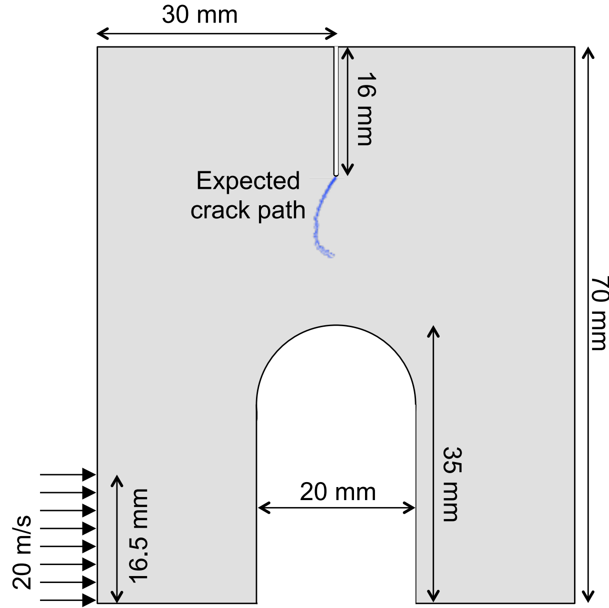


Figure 3.20: Schematic of Compact Compression Specimen (CCS) problem, the expected crack pattern from [4] is shown in blue

3.4.1 Compact Compression Specimen

The CCS specimen is chosen for numerical investigation because (1) the curved domain is readily meshed with polygons, and (2) the expected crack is curved and could not be captured fully by a structured mesh.

3.4.1.1 Description of problem

A schematic of the problem with the geometry, applied loading, and expected fracture path is shown in Figure 3.20. In the original experiment, the specimen was impacted with a Hopskins bar at the location that the velocity load is applied in Figure 3.20. However, to avoid the additional contact formation that would have been needed to simulate the bar, we opt to apply the velocity load directly to the specimen. The PMMA material is modeled with an elastic modulus of 5.76 GPa, a Poisson's ratio of 0.42 and a density of 1180 kg/m^3. The fracture properties are adopted from the work of Paulino et al. [88]. For both mode I and mode II the cohesive strength is 129.6 MPa, fracture energy is 352.3 N/m, and the shape of the softening curve is nearly linear.

3.4.1.2 Mesh generation

The domain was meshed with both random ad CVT polygonal elements. The following sections detail the construction of each.

Random polygonal element mesh

The random polygonal element mesh on the CCS domain was constructed in stages to achieve the necessary level of refinement near the notch tip and at the load application location. A total of 10,000 seeds were places throughout the domain. The Polymesher code was utilized and adapted for the random polygonal

elements. The domain was first defined as a series of rectangular and circular domains using the union and intersection commands available in Polymesher.

First, approximately 5% of the seeds are placed within a small radius of the notch at a minimum distance apart. Then we begin to move away from the notch and increase the minimum spacing between seeds. 10% of the total seeds are placed within a slightly larger radius of the notch, then another 30% are placed in the third radius around the notch. 10% of seeds are inserted in a rectangular region new the load application location, and 8% of seeds are placed along to boundary of the notch to ensure it is properly resolve. The remaining 37% of elements are placed throughout the domain at least the critical minimum distance apart (the final minimum distance is larger than any of the previous minimum distances). An alternative approach would have been to use a density function to control the placement of the mesh seeds, as was done for the CVT mesh generation in the next section.

Care also needed to be taken on the boundaries of the domain. At the straight portions of the boundary, the elements are simply cut to match the boundary. Near the curved edges, especially at the notch tip, nodes that fell outside/inside the curved boundary were adjusted such that they landed exactly on the boundary. This adjustment was mostly avoided thanks to the fine mesh resolution at the notch tip.

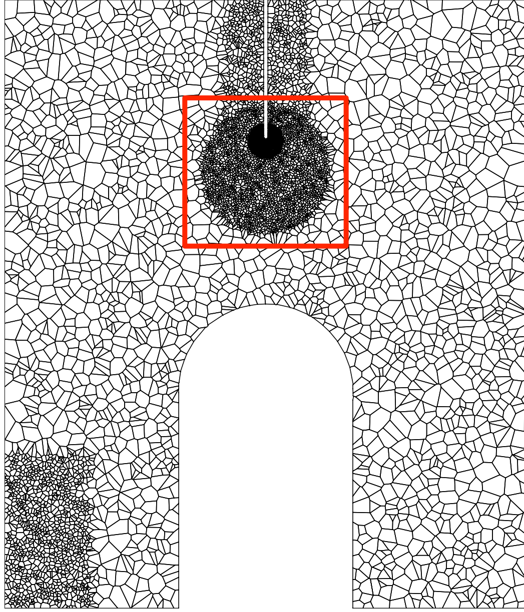
The final mesh consists of 10,000 linear polygonal elements with 20,200 nodes. The distribution of element type and edge size is shown in Table 3.3.

A coarser mesh was not used with the random polygonal elements for two reasons: (1) meshing around the small notch with the random polygonal elements was difficult, dozens of meshes were generated, and many points around the notch would have needed to be manually adjusted which is undesirable since the fracture is expected to initiate from this location, (2) Large irregular shaped elements, such as those that result from the random polygonal mesh generation featured elements with large edges (potentially larger than those of the CVT mesh) and fracture initiation may not happen because the stress would need to build up more than is physically reasonable.

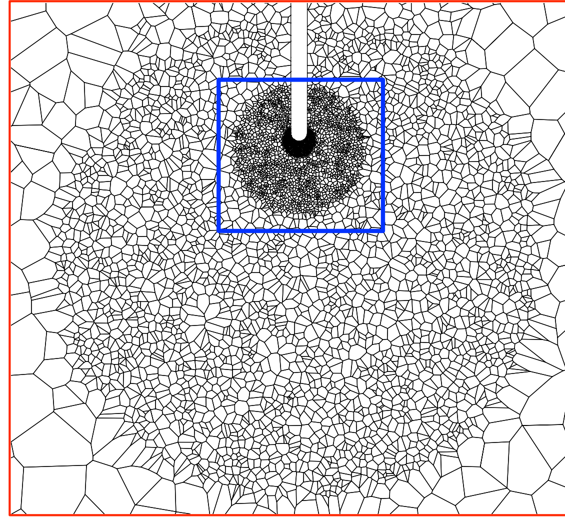
CVT polygonal element mesh

The CVT discretization of the CCS domain was generated using the Polymesher software [148]. As for the random mesh, the domain was was defined by a series of rectangle and circles with the union and intersection functions. Rather than using a constant density seed density throughout the domain, we defined a density function to concentrate nodes at the notch tip. Use of the density function results in a mesh with smooth transitions between the fine mesh of the notch and coarser far field mesh, as illustrated in 3.22. Because of the small size of the elements around the notch tip and the nature of the CVT algorithm, little adjustment was needed to ensure the nodes remained on the boundary. The Lloyd's algorithm tolerance was set at 5e-6. The final mesh contains 6,000 elements and 11,702 nodes. Other pertinent statistics can be found in Table 3.3. The same mesh is used for the numerical studies with and without element splitting.

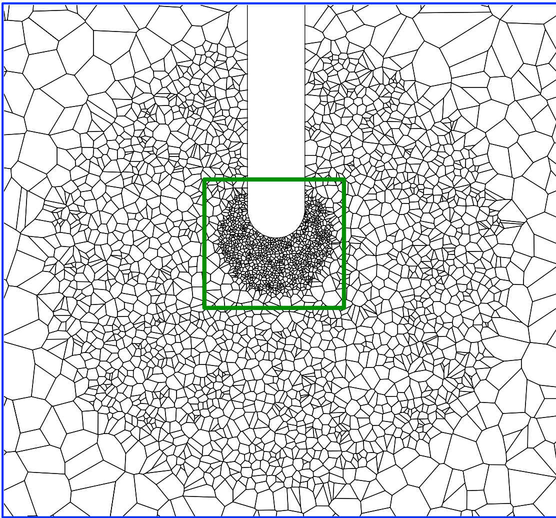
From Table 3.3, we notice an important difference between the CVT and random meshes. Iterations of Lloyd's algorithm make elements tend towards hexahedron whereas there is no apparent element preference in the random mesh. While it is not entirely accurate to compare edge sizes (because the random mesh has more elements) it is worth noting that in order to obtain a random mesh where fracture initiated, the smallest element size is much smaller than that of the CVT mesh. This is directly related to the time step and stability requirement, which will be be discussed in more detail in Section 3.4.2.



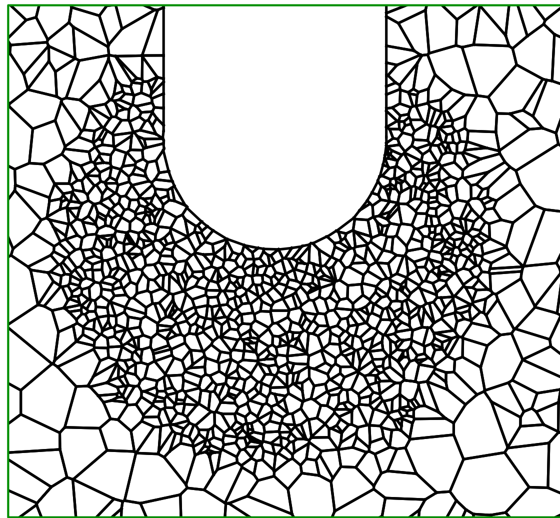
(a)



(b)



(c)



(d)

Figure 3.21: Random polygonal element mesh of CCS domain (a) full domain, (b) zoom in of red region indicated in subfigure (a), (c) zoom in of blue region indicated in subfigure (b), (d) zoom in of green region indicated in subfigure (c)

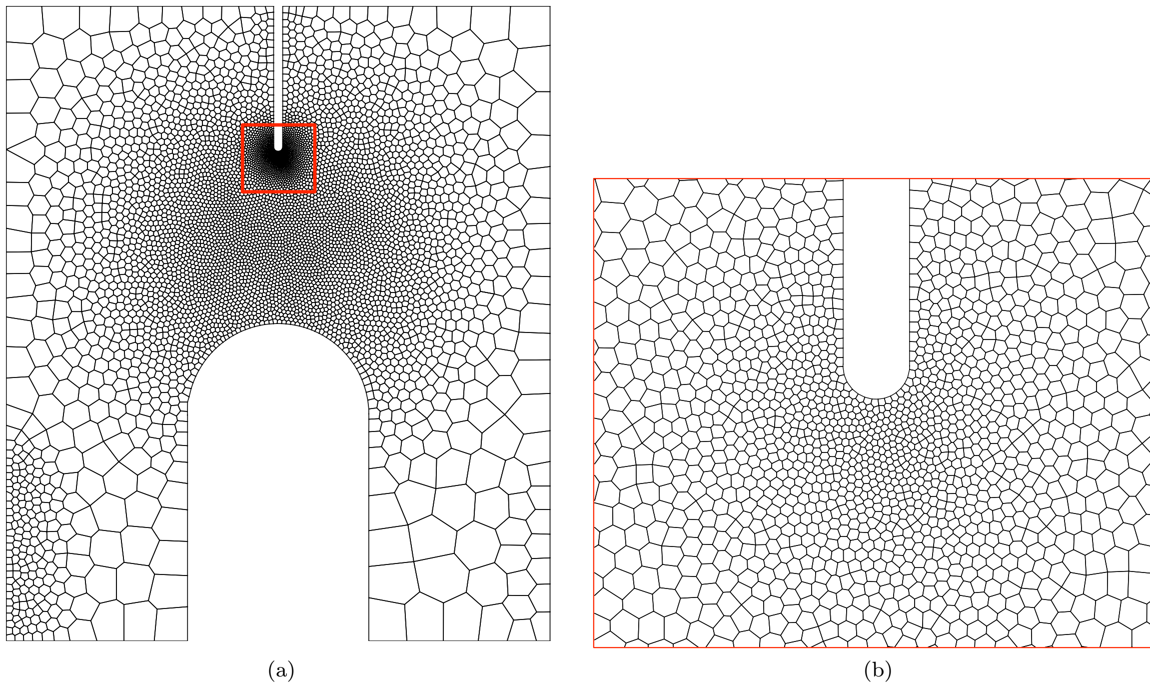


Figure 3.22: CVT polygonal element mesh of CCS domain (a) full domain, (b) zoom in of red region indicated in subfigure (a)

Table 3.3: Summary of mesh statistics for the CCS problem

	Random	**CVT**
Number of nodes	20,200	11,702
Number of elements	10,000	6000
3-sided elements	1.4%	0%
4-sided elements	11.1%	0.5%
5-sided elements	26%	20.1%
6-sided elements	29.5%	73.1%
7-sided elements	19.5%	6.3%
8-sided elements	8.6%	0%
9-sided elements and up	3.9%	0%
Minimum edge size	0.01 μm	6.0 μm
Maximum edge size	3.5 mm	5.5 mm

3.4.2 Simulation results

The results for the each polygonal element on the CCS domain are shown next. While the geometric studies suggested that unrestricted element splitting on either mesh type will lead to the best fracture pattern, the simulation results reveal that practical considerations necessary to make the problem tractable prevail.

3.4.2.1 Results on random polygonal element mesh

The random polygonal elements performed very well in the geometric studies, however the same cannot be said for their performance on an actual numerical simulation. The mesh contains elements with small edges, which resulted in time step restrictions that made the simulation nearly untenable. Many random meshes were generated and used to perform the simulation, however the results for each were similar: numerical instability occurred at some point during the simulation due to the small edge size.

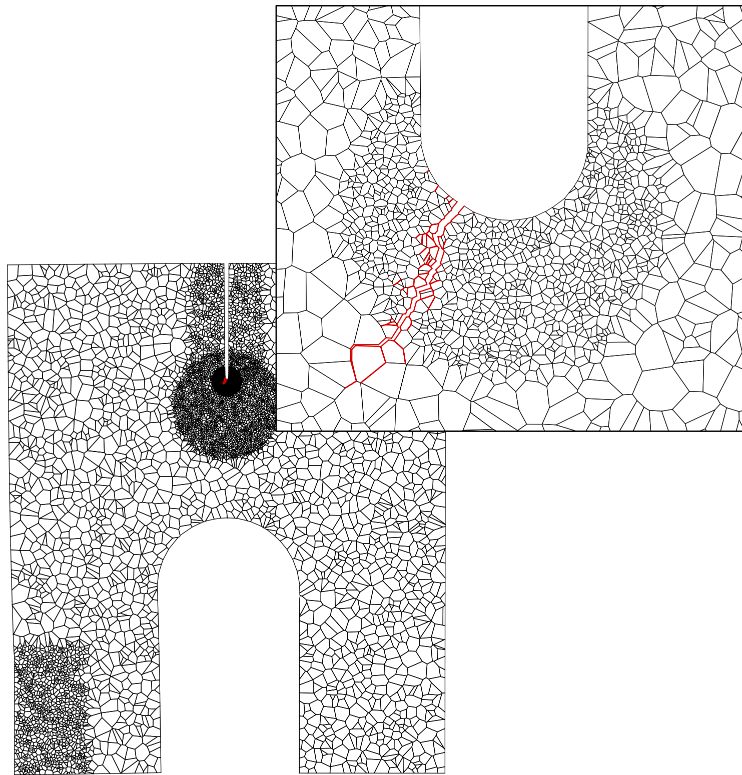
For demonstrative purposes, we show a partial result here. The full wall time would have been nearly 96 hours, however the simulation was not actually completed because of numerical instability encountered around 72 hours computational time. Results at the early stages of fracture are shown in Figure 3.23. The red facets show where cohesive elements have been inserted. It should be noted that these elements are not necessarily open cracks as many of the elements are still in the softening region of the PPR traction-separation relation. The crack begins in the expected location and propagates appropriately. There are several cohesive elements inserted shortly after crack nucleation, suggesting that the stress is building up but the crack cannot progress beyond the softening stage. Eventually, the crack passes the region where many elements are inserted, however, the direction begins to diverge from that which is expected based on the previous experiment and numerical results. The expected curvature is not present, so the crack can no longer propagate. However, stress continues to build up triggering the activation of more cohesive elements. The separation of excessive cohesive elements and the time step restriction results in numerical instability shortly after the image shown in Figure 3.23(b).

We chose not to move forward with the random polygonal element mesh with unrestricted or restricted element splitting. After several attempts at performing the simulation with element splitting on many different random meshes, the problem became unstable too quickly because of the presence of ill-shaped elements, which cause too much restriction on the time step. Since the results without splitting already took so long, adding splitting and further reducing the time step was not practical.

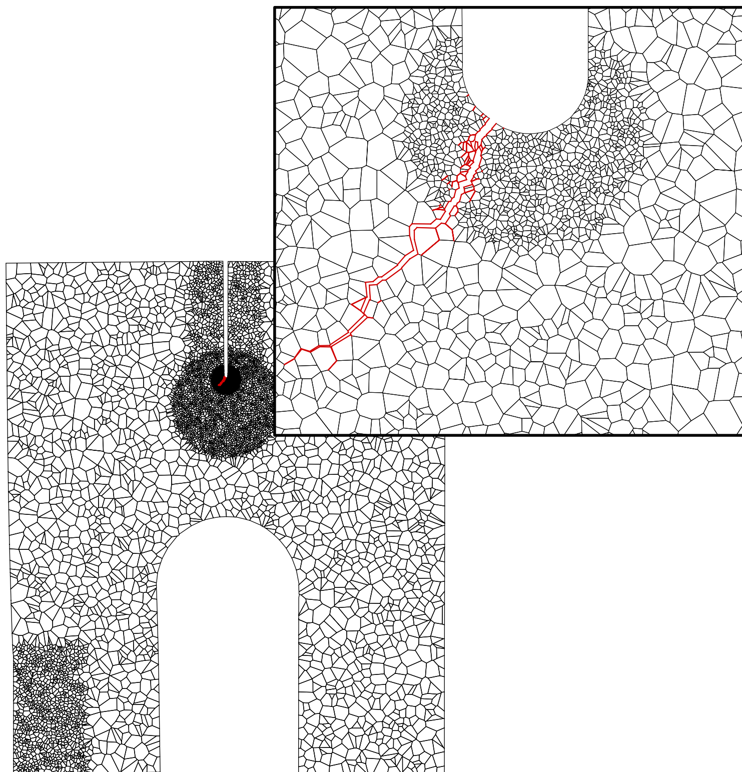
Future development using the random polygonal elements could lead to more practical meshes. For example, the minimum radius between element seeds could be increased which would lead to more regular elements. However, this was not pursued in this work because the CVT mesh provided an excellent alternative given the existence of the mesh generation software.

3.4.2.2 Results with CVT polygonal element mesh

The CVT polygonal element mesh did not perform as well as the random mesh for the cases of no splitting and restricted splitting, however the regularity of the elements makes them more successful in the CCS numerical simulation. First, we present the result without splitting in Figure 3.22. Unlike the random mesh, the CVT mesh did not pose great restrictions on the time step, so the wall time for the simulation was much more reasonable. Additionally, the lack of small edges protected against numerical instability. While the simulation executed successfully, the final result is not ideal. When comparing the result without splitting to the accepted numerical results in the literature, we see a clear lack of curvature in a couple of locations. Due



(a)



(b)

Figure 3.23: Crack patterns of CCS problem with random polygonal element mesh at (a) 64.02 μsec and (b) (a) 65.31 μsec

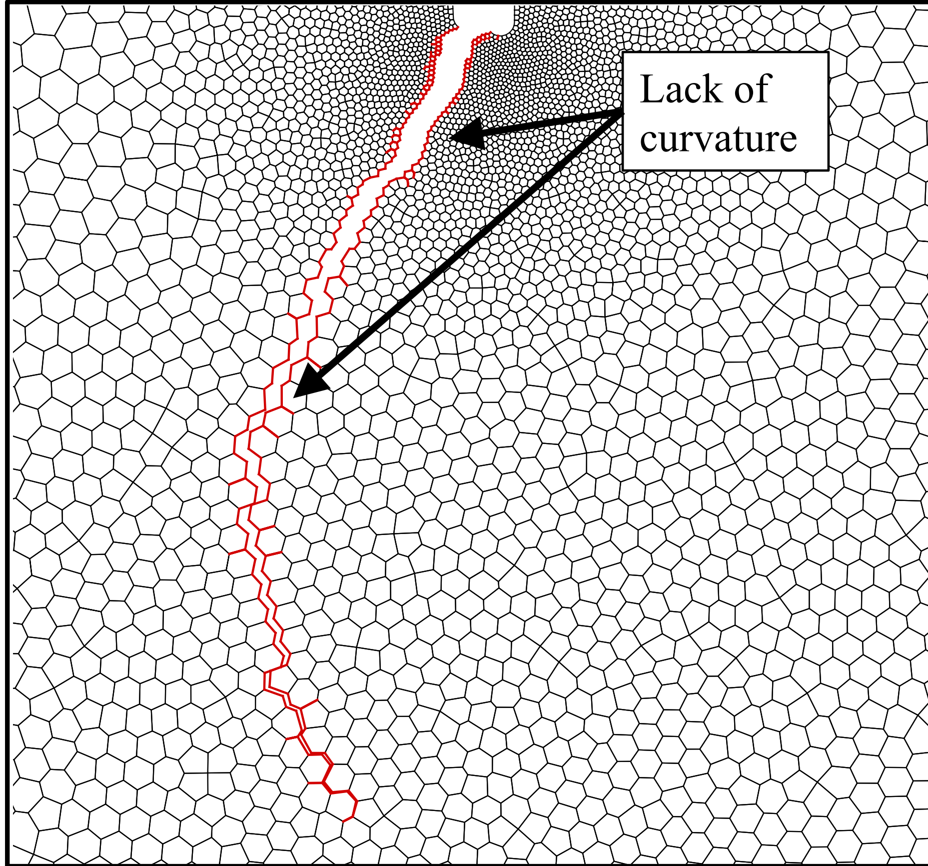


Figure 3.24: Fracture pattern on CCS mesh (zoomed in near the notch tip) with the CVT polygonal element mesh without element splitting

to the lack of available directions at each node, the crack could not turn in the direction needed. Moreover, the crack pattern is rather jagged especially as the crack propagates away from the notch tip and into the region of coarser elements. we can clearly see how allowing the element to split would improve the curvature and overall crack path. Thus, we move forward with the CVT mesh with adaptive element splitting.

The CVT mesh with restricted adaptive element splitting result is shown in Figure 3.25. Recall that restricted splitting means that the element can only be split with the node that reduces the difference in areas between the two resulting polygonal elements. Unlike the case without splitting, we see the appropriate curvature in the final crack pattern. The crack is also smoother in the region of coarser elements. We also see some additional softening and micro-cracking near the tail of the crack. It is important to note that the time step did not need to be adjusted for the restricted splitting case because the element sizes do not change greatly from the no splitting case.

Finally, we investigate the CVT with unrestricted splitting; the polygonal element can be split along any node. The result shown in Figure 3.26 is clearly different than that of restricted splitting even though the base mesh is identical. We postulate that this difference could be due to the fact that the order in which facets are visited when unrestricted splitting is enabled is different than when splitting is restricted. Numerical variations will accumulate as quantities are gathered in different orders, which could explain the difference. This concept is further explained and demonstrated later in this document in Section 5.4, where we employ parallel computing and can investigate these numerical variations with great efficiency.

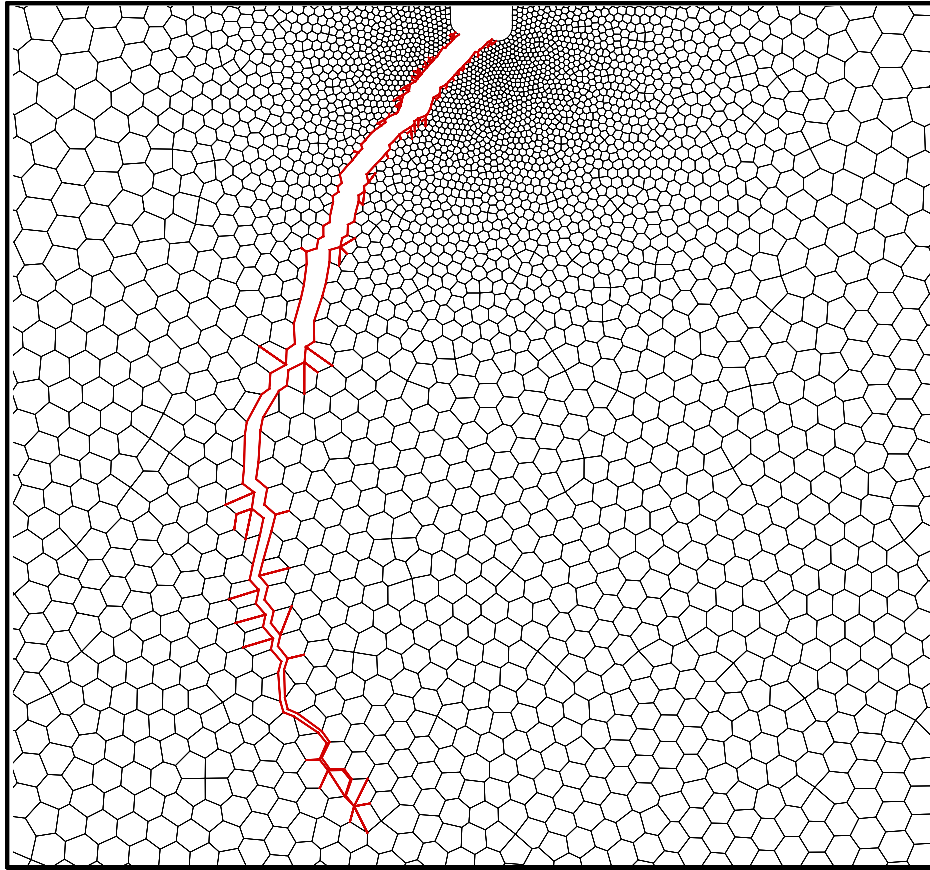


Figure 3.25: Fracture pattern on CCS mesh (zoomed in near the notch tip) with the CVT polygonal element mesh with restricted element splitting

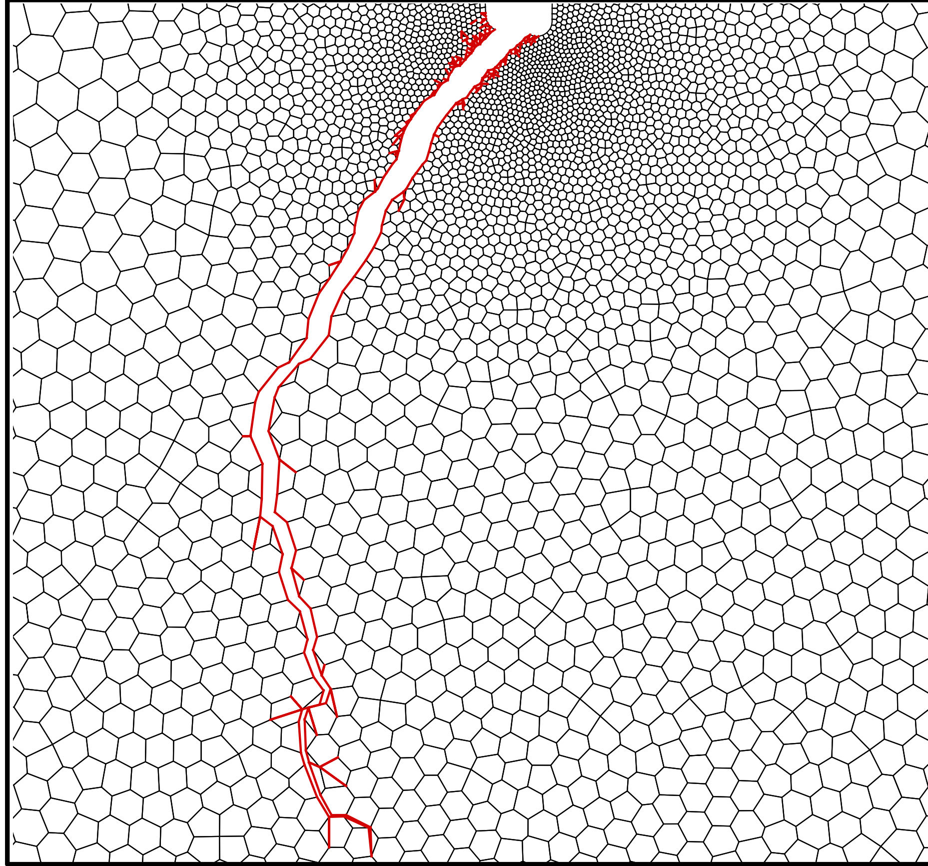


Figure 3.26: Fracture pattern on CCS mesh (zoomed in near the notch tip) with the CVT polygonal element mesh with unrestricted element splitting

Besides the numerical variation, the additional crack directions greatly contributes to the different fracture pattern. The expected curvature and smooth fracture pattern is present in this result. However, the results presented was the outcome of several simulation attempts. The unrestricted splitting poses some restriction on the time step that can lead to numerical instability. Unrestricted splitting may not be computationally practical. Based on the observations in the geometric and numerical studies, we suggest that the CVT polygonal elements with restricted element splitting achieve a good balance between improvement to results over the no splitting case and practicality of the simulation over the unrestricted splitting case.

The use of polygonal elements for dynamic fracture simulation have shown to be very promising. Several future research directions could be proposed using this work as a start. One extension that utilizes the element splitting on an adaptively refined CVT mesh was already investigated in [89].

Inspired by the remeshing concept of the previous chapter, local remeshing of the CVT mesh could be explored. Given the crack direction a region around the current crack tip could be remeshed with polygonal element such that the facets ahead of the crack tip align with the crack direction. The results in this section also brought up issues related to the critical time step imposed by the explicit time stepping scheme. Thus, and additional area of future work would involve incorporating a time step sub-cycling scheme with the polygonal elements. Please see Sections6.2.3 and 6.2.5 for more details about these potential future research directions.

Chapter 4

Adaptive refinement and coarsening on structured 3D meshes

In order to model the failure of large quasi-brittle systems, fully three-dimensional simulations must be conducted. Certain physical effects, for example, out of plane stresses and strains, cannot be faithfully captured with a two dimensional model. Since the ultimate goal of this research is to enable predictive failure simulation, we move to three dimensional finite elements. However, the computational storage and cost increases dramatically when a simulation is extended into three dimensions, thus adaptive schemes become imperative to make the problems computationally tenable.

As indicated in the previous chapter, the 4k mesh in 2D admits some bias, and the same will be the case for 3D. However, in the interest of making large scale simulations feasible, the first efforts will be focused on reducing computational cost through an adaptive mesh refinement on the 3D 4k mesh.

4.1 4k structured meshing in three-dimensions

We utilize the 3D extension of the so-called 4k mesh discretization. The 2D version of this mesh was utilized in the previous chapter for comparison to the 2D polygonal element mesh for dynamic fracture applications. Here we investigate the 4k mesh, which is well suited for adaptive mesh refinement and coarsening because of its hierarchical structure.

The 3D 4k mesh is essentially a mesh of hexahedra discretized into 24 tetrahedron, as shown in Figure 4.1. Further subdivisions of the element results in hexahedron of six and 12 tetrahedron. This subdivision is illustrated in Figure 4.2. First we begin with one hexahedron divided into six tetrahedron (Figure 4.2(a)), then the six elements are split along the interior edge (as illustrated in Figure 4.3) which results in 12 tetrahedra per hexahedron (4.2(b)). Next, the diagonal edges on the outside of the hexahedra are split resulting in 24 tetrahedra per hexahedron (4.2(c)). The next subdivision changes the single hexahedron to 2x2 hexahedron each subdivided into six tetrahedra (4.2(d)), then the same process repeats (4.2(e)-(f)). In this process each discretization has an associated level as shown in Figure 4.2, which contains Levels 0-5. A refined mesh on a grid can be created very quickly by simply specifying the number of hexahedra in each direction, then the level of subdivision per hexahedra. Notice that a level of refinement above 2 will increase the number of underlying hexahedra. For example, if a grid of 10x10x10 hexahedra is specified at level 6, then the resulting mesh will contain 384,000 tetrahedra finite elements (40x40x40 underlying hexahedron each comprised of 6 tetrahedra finite elements). This procedure of subdividing the mesh is exactly the procedure utilized for adaptive mesh refinement and the reverse is used for adaptive mesh coarsening, which will discussed in detail later in this chapter.

The number of elements in the mesh increases exponentially with additional levels of refinement, and the rate of increase in the 3D mesh is much greater than that of the 2D mesh, as shown in Figure 4.4. This growth in number of elements motivates the adaptive mesh refinement scheme. A relatively coarse mesh

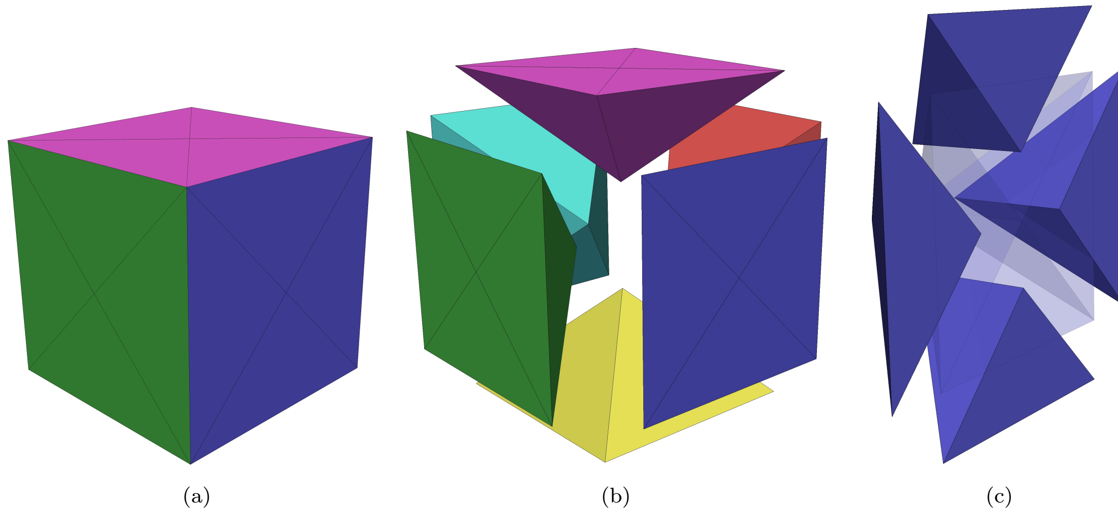


Figure 4.1: 4k mesh constructed by subdividing a hexahedron into 24 tetrahedra (a) each of the 6 faces of the hexahedron is divided into 4 tetrahedra (b) exploded view of each face of the hexahedron (c) exploded view of one face of the hexahedron contains 4 tetrahedra

would be used throughout the domain (i.e., far field regions), then regions with high displacement gradients (i.e., at crack tips and near application of external loads) would be refined. Then as the cracks propagate, regions that no longer need a fine discretization can be coarsened.

4.2 Implementation of adaptive cohesive fracture with the TopS data structure

The use of mesh adaptivity requires on-the-fly mesh modification during the simulation. In order to handle these mesh modifications, the present work uses the consistent topological data structure, TopS [162, 163], which allows for efficient modifications to the adjacency relations on an as-needed basis. Adjacency information is retrieved and cohesive elements are dynamically inserted in time proportional to the number of entities retrieved/inserted.

4.2.1 Finite element mesh representation using TopS

Node and element entities are explicitly stored in memory, while edges, vertices, and facets are implicit entities. From the application point of view, there is no difference between implicit and explicit entities as they are accessed in the same way. TopS is a complete topological data structure, meaning that from any entity, the adjacency of all entities can be determined (e.g. from a given node all adjacent nodes, elements, facets, edges, and vertices can be found). Additional entities, called "uses", are oriented and indicate the use of an edge, vertex or facet by an element. The entity definition and storage used in the TopS data structure makes it applicable for both 2D and 3D mesh representation. Several finite elements are supported by TopS, including classic elements (e.g., linear and quadratic triangles and quads, linear and quadratic tetrahedron and bricks, etc.) and 2D linear polygonal elements that may contain any number of nodes. It should be noted that edges and facets are equivalent and nodes and vertices are equivalent in 2D. Linear and quadratic

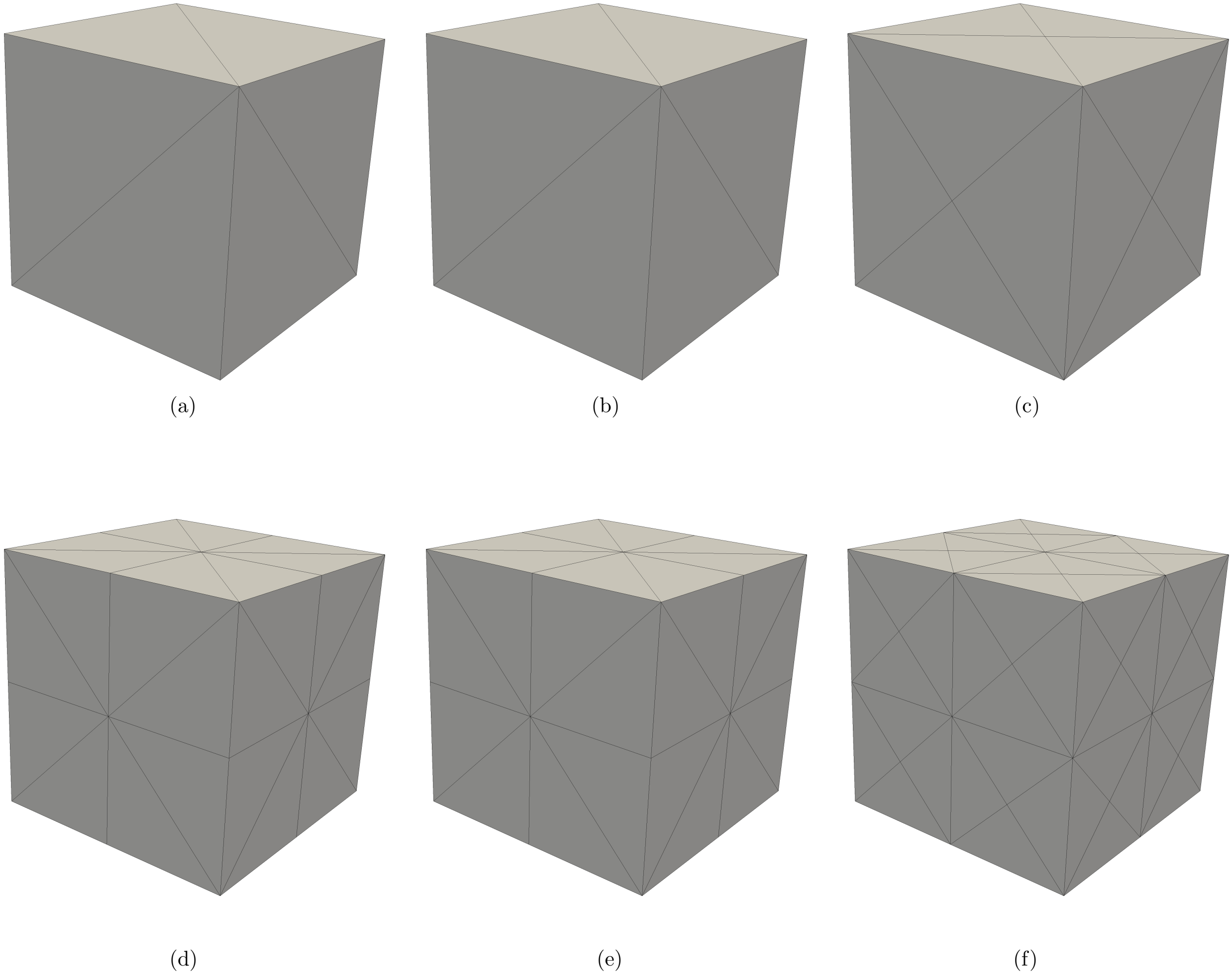


Figure 4.2: Levels of refinement on a unit cube, each increase in level doubles the number of linear tetrahedral elements from the previous level. (a) Level 0 - 6 elements, 8 nodes, (b) Level 1 - 12 elements, 9 nodes, (c) Level 2 - 24 elements, 15 nodes, (d) Level 3 - 48 elements, 27 nodes, (e) Level 4 - 96 elements 35 nodes, (f) Level 5 - 192 elements 71 nodes (Note that on the exterior level 0 looks identical to level 1, and level 3 looks identical to level 4 because the splitting occurs on the interior edges as illustrated in Figure 4.3)

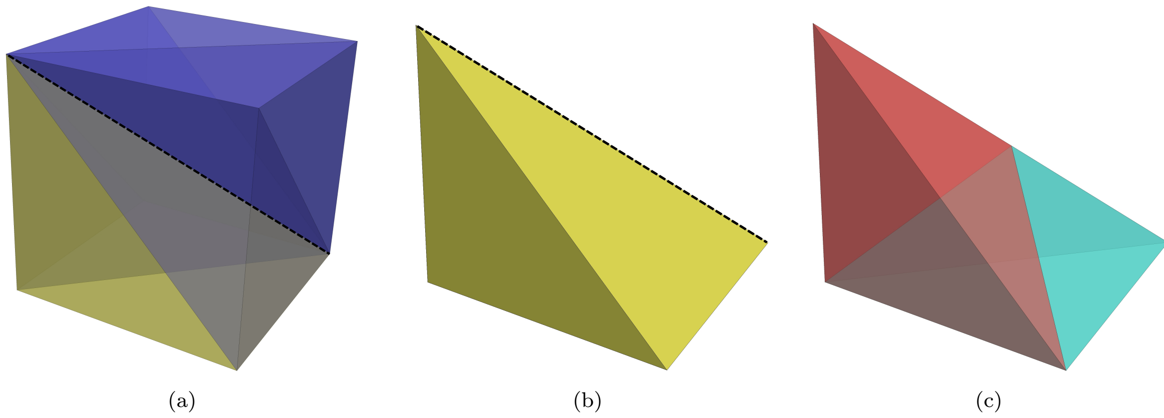


Figure 4.3: Subdivision of hexahedron from 6 to 12 tetrahedra (a) the six element hexahedron is subdivided along the interior edge, show as a dashed line (b) a single tetrahedron is isolated and the edge upon which it will be split is shown by the dashed line (c) new tetrahedra resulting from splitting

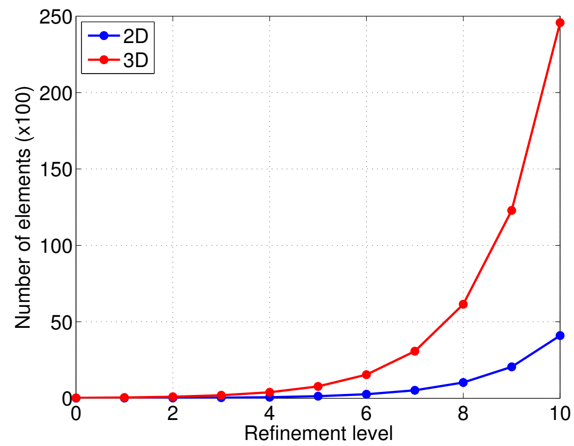


Figure 4.4: Rate of increasing in mesh size for a 2D vs. 3D mesh when all elements in the mesh are refined

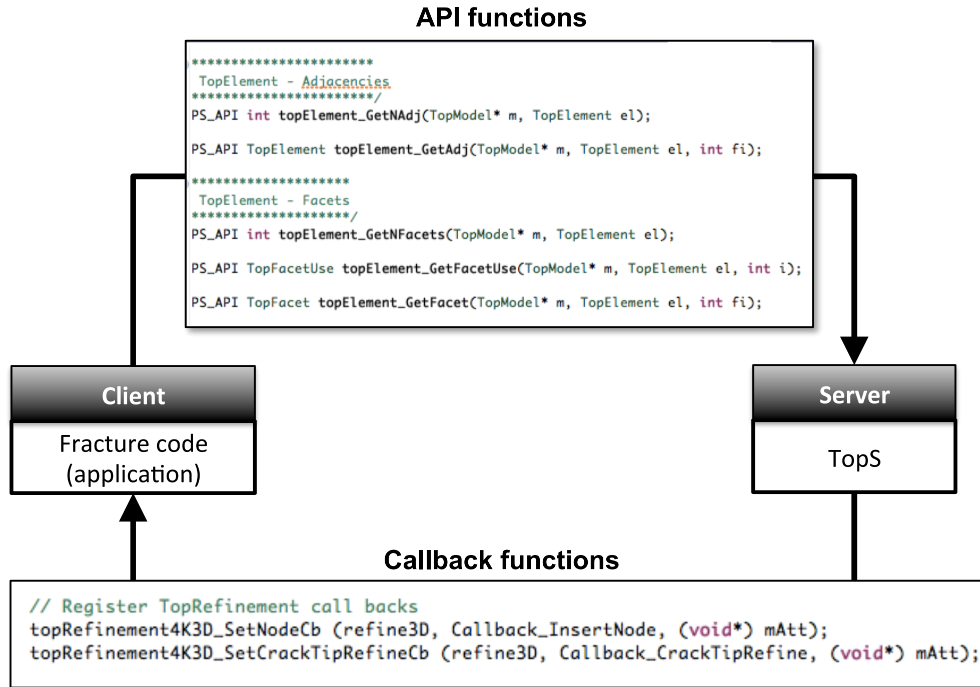


Figure 4.5: Schematic of the client-server approach between the TopS API and the application

cohesive elements are also supported by TopS. The cohesive element is chosen such that it is compatible with the bulk finite elements.

The Tops data structure can be thought of as a server that is invoked by a client application, which in this case, is the explicit dynamic finite element analysis engine and fracture code. While the data structure contains all of the information about finite element mesh geometry and topology, it is completely decoupled from the client code, so it does not contain any information about the kinematics, mechanics or physics of the problem. Communication between the server and client is critical and is achieved through the TopS application programming interface (API) and callback functions. The client application calls the server through the API functions; for example, the application can add a node to the model by calling API function `topModel_InsertNode`. When a node is added, updates to the geometry and topology will be handled by TopS, but the client application will also need to make updates such as interpolating the nodal displacement from existing nodes to the new node. The client is notified of these changes through a callback function which TopS calls during the insertion of a new node. The callback function is implemented in the client code, giving the developer complete control of what needs to be done when the mesh is modified. A schematic of the client-server approach is shown in Figure 4.5.

The unoriented mesh entities are equipped with attributes, which are simply void pointers. The client application may initialize and assign quantities to these attributes as needed. For example, nodal attributes contain displacement, velocity, mass, etc. and are attached to the topological nodal entity. Thus, when the client application accesses an entity through the TopS API function, it can also access its attribute and read or modify the data.

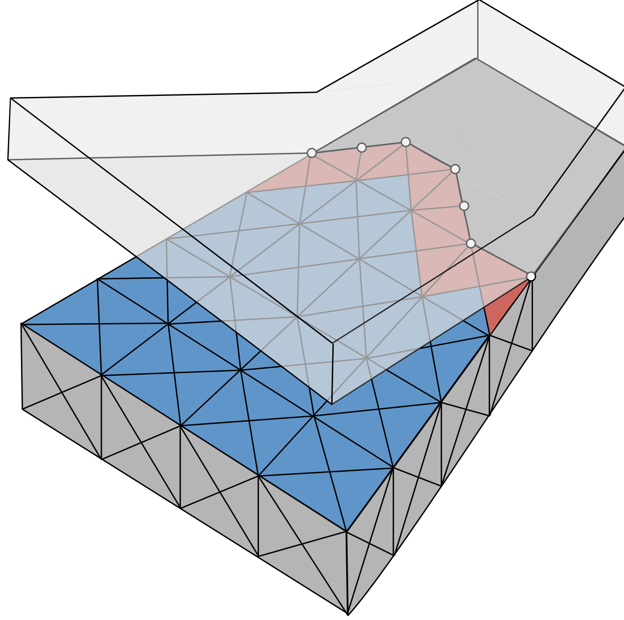
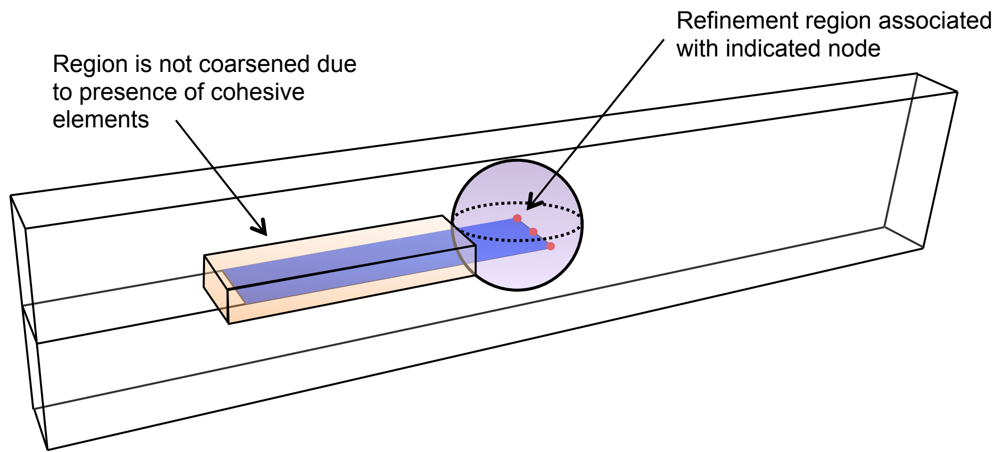


Figure 4.6: 3D view of crack front, cohesive elements are shown in blue and red, where the red elements are crack tip elements; the crack tip nodes, un-duplicated nodes on cohesive elements are indicated in red

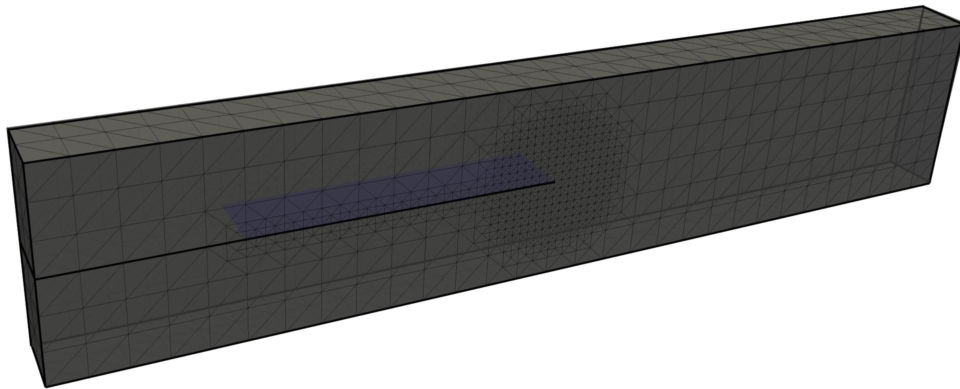
4.2.2 Automatic crack tip tracking

In the simulation of fracture in 3D, it is necessary to keep track of the crack front. The definition and tracking of crack tips are essential for the adaptive mesh refinement and coarsening strategy developed in this work. We use the *a priori* assumption that regions around crack tips (i.e. regions of high gradient of the primary field) must be the finest in the simulation [158]. Thus, by tracking the crack tips we are implicitly defining the regions that must be refined and those than may be coarsened throughout the simulation. The crack front is defined as cohesive element nodes that are not duplicated. In Figure 4.6, for example, the crack tips are indicated by the white nodes, which are the the unduplicated nodes of the crack front cohesive elements in red. Elements whose centroid fall within a spherical regions of a user-defined radius around the crack tips are subdivided until a user-defined level of refinement is reached (see Section 4.4 for more details on the refinement scheme). The example in Figure 4.6 contains seven crack tip nodes, thus it would contain seven regions of refinement. The crack progresses by duplicating nodes on the cohesive elements. Once duplicated, a node is no longer a crack tip, then the refinement region associated with it is destroyed, and the region may be coarsened. All of the nodes on the cohesive elements shown in blue in Figure 4.6 have been duplicated, thus none of them are crack tips elements around them may be coarsened (see Section 4.5 for more details on the coarsening scheme). Figure 4.7 shows an example of the active and inactive refinement regions, where elements outside the active refinement regions elements may be coarsened. Notice that elements near the cohesive elements are not fully coarsened. In order to avoid transfer of interval instate variables and coarsening of the crack path, we do not coarsen out cohesive elements in this work. Therefore, to maintain mesh compatibility, it is necessary to have a transition zone between the refined cohesive elements and coarse bulk elements, which is shown in Figure 4.7(b).

Given our definition of the crack tips, we track the crack front in the dynamic problem with the 3D 4k refinement manager of the TopS data structure. Essentially the refinement manager stores the nodal location of crack tips and allows the application code to refine or coarsen the mesh based on the regions defined by the



(a)



(b)

Figure 4.7: Active and inactive crack tip regions identified by the ToPS refinement manager (a) All crack tip nodes have an active spherical region of refinement associated with them and any region outside the refinement regions is inactive and coarsened as much as possible. Regions around cohesive elements will never be fully coarsened because cohesive elements stay at the most refined level and bulk elements around them must transition from fine to coarse. (b) Resulting mesh with refinement regions only at the crack tips

crack tips. The application creates the refinement manager, which will be associated with the TopS model (i.e. the finite element mesh), and enables crack tip tracking using the appropriate API functions. Then, it registers the call back functions described below, which are necessary for automatic crack tip tracking. The C implementation of the callback functions is given in Appendix B.

- topRefinement4K3D_SetMustCreateRegionCb
  - This callback function is invoked when a crack tip node (i.e unduplicated node on a cohesive element) is identified by TopS. The application notifies TopS if a refinement region should be created for this node. If so, the application assigns the refinement radius and level (e.g. smallest element size).
- topRefinement4K3D_SetNodeCb
  - Whenever a new node is inserted due to refinement this callback function is called so that the application code can add the node to the model and appropriately handle the transfer of data from the existing nodes to this new one. See Section 4.4.3 for more details on the insertion of new nodes into the model.
- topRefinement4K3D_SetMustDestroyRegionCb
  - This callback function is called when a previous crack tip node is duplicated, thus it is no longer a crack tip. The application indicates to TopS whether the refinement region associated with the node may be destroyed or not.
- topRefinement4K3D_SetCanCollapseNodeCb
  - If a previously refined node lies outside of an active refinement region it may be removed and its adjacent elements coarsened. In this callback function, the application notifies TopS if the node may in fact be removed. See section 4.5.3 for details on the criteria to coarsen a patch of elements.
- topRefinement4K3D_SetMergeElemCb
  - This callback function is invoked when a node has already been marked for removal in the topRefinement4K3D_SetCanCollapseNodeCb callback. Here, the application handles transferring data from the old, refined nodes and elements to the new, coarse ones. See section 4.5.3 for details on the transfer of data between the fine and coarse mesh.

The automatic crack tip tracking can only occur when cohesive elements are inserted. Thus, after a cohesive element is inserted, the application is responsible to update the mesh by calling the TopS API functions topRefinement4K3D_UpdateMeshCoarsening and topRefinement4K3D_UpdateMeshRefinement. These functions activate the crack tip tracking and refine and coarsen the mesh as indicated by the application's responses to the callback functions listed above.

4.2.3 Numerical implementation

The explicit dynamic extrinsic cohesive fracture framework is implemented in the C programming language. The TopS source code is implemented in C++, however the API is available for C/C++ applications. Although this type of application is well suited for an object-oriented framework, the current version is non-object oriented (as evident by the use of the C programming language). However, the code uses structures heavily so that a list of variables is stored on the same block of memory and accessed through a single pointer. The structures are primarily used to store attributes of model entities, i.e. node attribute, bulk element attribute, cohesive element attribute. For example, the nodal attribute holds information such as the nodal mass, displacement, velocity, acceleration, stresses, strains, force vectors, and various flags. The application code allocates space for attributes when the associated entity is added to the TopS model. Then the attribute is initialized and attached to the TopS model. When an entity is removed from the model, the application is responsible for destroying the attributes and freeing the memory allocated for them. Since the models become quite large in 3D the memory demands are quite high, thus memory must be used conservatively and responsibly (e.g. all allocated memory is freed such that no memory leaks are present).

The serial application code is outlined in Algorithm 1. The first eight lines of the code are associated with allocating space, initializing variables, and building the model. Similarly, the last three lines are related to destroying the allocated memory. The majority of the application code is contained inside the while loop shown from lines 10-37 in Algorithm 1. This is the numerical implementation of the Central-Difference time integration scheme discussed in Section 3.1.1 of the previous chapter. We will focus on the crack propagation and resulting mesh modification portion of the code, which is contained to lines 12-30 in Algorithm 1.

The user specifies the increment at which cohesive elements may be inserted. For most of our applications, we allow cohesive elements to be inserted every 10 steps based on the recommendation in [153]. When one of these increments is reached, the activation criteria for insertion of cohesive elements is evaluated. Strength criteria is most often utilized for quasi-brittle fracture applications as this is physically relevant and straightforward to implement. The stress criteria we have implemented here is implemented in the same way as it was for the 2D investigation of the previous chapter. Inserting cohesive element changes the mesh topology and requires transfer of field variables. To avoid successive and expensive changes to the mesh, we first gather all of the facets that meet the activation criteria in a particular step, then revisit those facets to insert cohesive elements. In this way, all of the mesh changes are made after the evaluation of the activation criteria. The insertion of cohesive elements begins a series of call back function calls, during which TopS changes the topology to reflect node duplication and the application transfers necessary data to keep the physical model consistent. More details on the callback functions for the insertion of cohesive elements can be found in Section 4.3. After cohesive elements are inserted, the application signals TopS to detect crack tips and coarsen and refine the mesh appropriately. Sections 4.4 and 4.5 have the detailed information on the call back and API function calls that occur after new crack tips are detected for refinement and coarsening, respectively. Finally, once the mesh topology is updated and field variables are transferred as needed, the mass and stiffnesses of the new mesh entities are computed. The dynamic time integration loop is then completed and proceeds to the next step until the maximum number of steps is reached.

4.3 Adaptive insertion of cohesive elements in 3D

Numerical implementation of the extrinsic cohesive zone model requires that cohesive elements are inserted on an as-needed basis, thereby dynamically changing the number of elements, nodes, and connectivity in

Algorithm 1 C Implementation of adaptive cohesive fracture code

```
1  Create and initialize TopS model and model attribute
2  Read input file
3  Build model
4  Create Refinement Manager
5  Register call back functions for insertion of cohesive elements
6  Register call back functions for mesh refinement
7  Register call back functions for mesh coarsening
8  Enable crack tip tracking
9
10 While current step < max steps
11     Update displacements
12     If step = number of steps to check insertion of cohesive elements
13         Compute stress at nodes
14         If principle stress > 90% of cohesive strength
15             Flag node
16         End If
17         For each flagged node
18             Compute normal stress along adjacent facets
19             If normal stress > normal cohesive strength
20                 Insert cohesive element (See section 4.3)
21             End
22         End For
23         If cohesive elements were inserted
24             Update mesh coarsening (See section 4.5)
25             Update mesh refinement (See section 4.4)
26             Calculate stiffness and mass matrices of new elements
27             Calculate mass of new nodes
28             Apply boundary conditions to new nodes
29         End If
30     End If
31     Compute internal force vector
32     Compute cohesive force vector
33     Update velocities and accelerations
34     Compute energy
35     Update boundary conditions
36     Write output
37 End While
38
39 Disable crack tip tracking
40 Destroy refinement manager
41 Destroy model and model attributes
```

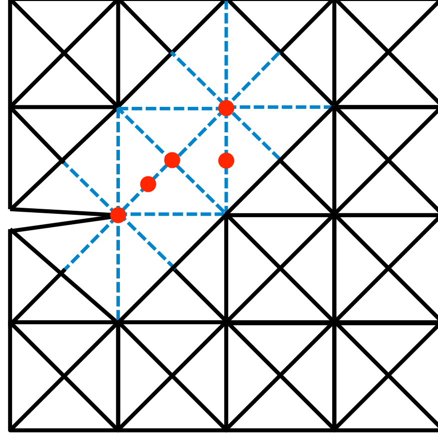


Figure 4.8: Nodes are flagged (shown in red) when the principle stress exceeds 90% of the cohesive strength of the material, then adjacent edges (shown in dashed blue) are visited and stresses along them are computed to determine if a cohesive element should be inserted

the mesh [152, 179]. Without an efficient way to manage the changing mesh, the computations associated with large scale adaptive, dynamic failure simulation would quickly become untenable. Thus, the TopS data structure was designed to provide efficient retrieval and modification of mesh entities [163].

Insertion of cohesive elements involves duplicating the nodes of the adjacent bulk elements and adding interface elements to the mesh. However, all adjacent nodes are not necessarily duplicated, so a systematic procedure for determining which nodes to duplicate is necessary. The systematic procedure described in [163] will be explained here by example . Consider the portion of a 3D mesh shown in Figure4.9(a), where a cohesive element is inserted at the facet between the red and blue elements. In figure 4.9(b) the mesh has been pulled apart to show the relevant oriented and unoriented entities. The facet, labeled f1 in Figure 4.9(b) is bounded by the purple dashed edges. When the cohesive element is inserted, the adjacency information is updated accordingly and E1 and E4 are no longer adjacent. Through this example, we will determine if edge 1, labeled e1, should be duplicated.

Start by examining the first interfacing element to the new cohesive element, labeled E1. Next, access this element's interfacing facet-use, which is labeled fu 1(1). The first integer in the label is the facet-use number of the element (each brick element has six facet uses) and the integer in parentheses corresponds to the element number to which the facet use belongs. So, fu 3(2) would be facet-use 3 on element 2. Then, from fu 1(1), obtain the first edge-use labeled eu 1(1). From this edge-use, all other uses of the same edge, i.e., eu(4)2, eu(3)3, and eu(2)4, are obtained through a local search of edge-uses associated with fu(1)1 [162]. The elements associated with each of these edge-uses are visited. If element, E4, is not reached, then edge 1 is duplicated. However, in this cace, E4 is reached, so the edge is not duplicated. Clearly if e1 was a boundary edge or if elements adjacent to it were already separated, then it would be duplicated. This procedure is carried out for all other edge-uses of fu1 to determine if the remaining edges of f1 need to be duplicated. An analogous procedure is carried out for node duplication. We rely on this node duplication for the definition of the crack tip in fracture simulations. A crack tip is identified as an unduplicated node on a cohesive element.

Using the procedure described above, Paulino et al. [163] have demonstrated that cohesive elements are dynamically inserted in time proportional to the number of entities inserted. Moreover, adjacency information is retrieved in proportional time.

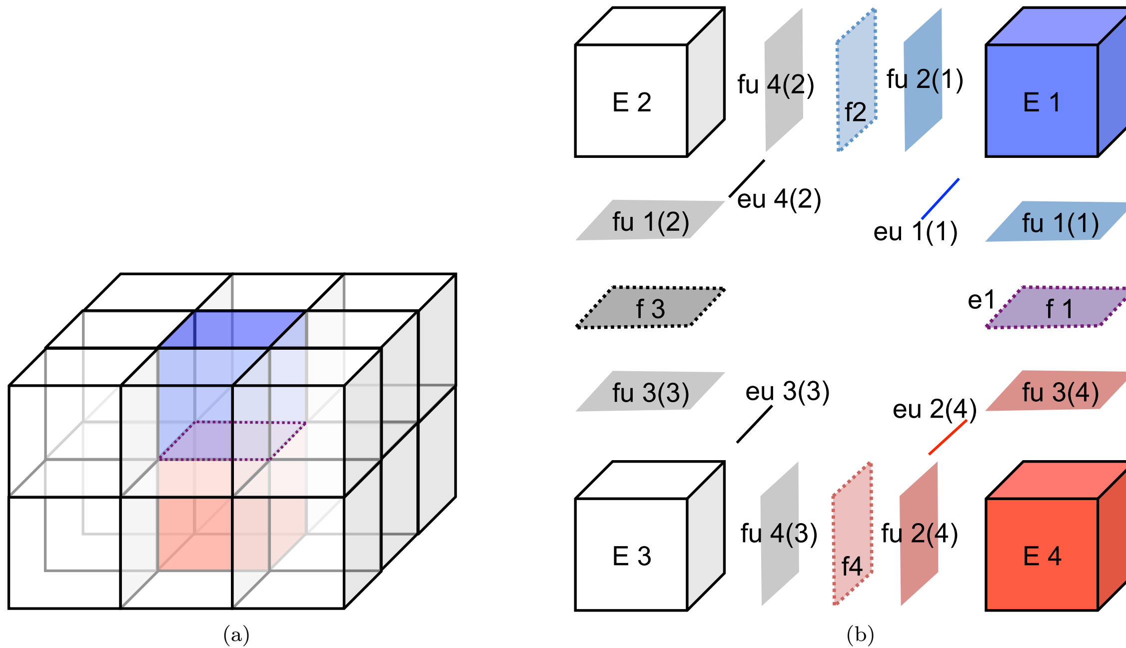


Figure 4.9: Insertion of a cohesive element in a 3D mesh (a) cohesive element is inserted at the purple facet between the blue and red elements, (b) mesh is blow up and entities separated

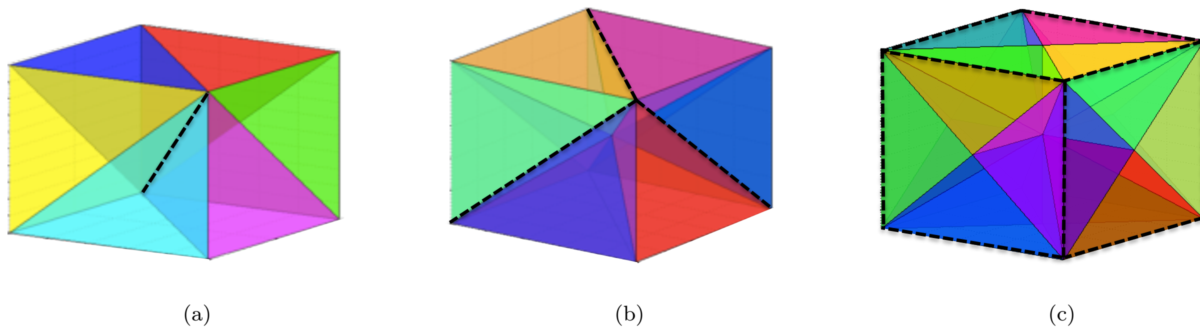
Before the simulation begins, the application registers the node duplication call back function (see Algorithm 1), which is called by TopS each time a node is duplicated. Inside the callback function the new node is assigned a nodal ID, and the attributes from the original node are simply copied to the new node. Any unduplicated nodes on cohesive elements are identified the by refinement manager as potential crack tips around which the mesh may be adaptively refined.

4.4 Adaptive mesh refinement

Adaptive mesh refinement greatly reduces the computation storage and processing time requirements for the dynamic fracture simulation. We address the main geometric and physical aspects of adaptively refining the 3D 4k mesh and detail the numerical implementation into the application code.

4.4.1 Geometric aspects of adaptive mesh refinement

Given a point in space and radius, all elements whose centroids fall within the sphere are refined to a level specified by the client application. The maximum level of refinement may be set at the time the refinement region is created by the TopS refinement manager, however for this work we utilize a constant minimum level of refinement for all regions. The algorithm for refinement consists of inserting a node a the mid point of the longest edge of an element, then dividing all adjacent elements with their existing nodes and the newly inserted node. Depending on the configuration of the mesh, splitting an edge may result in 2, 4, 6, or 8 adjacent elements being split. As shown in Figure 4.10(a), when the hexahedra contains 6 elements, the longest edge is the interior edge, and when it is split 6 adjacent elements are split. For the case of 12 elements per hexahedra, as shown in Figure 4.10(b), the longest edges are the diagonals on the outside of the hexahedron. Notice that if the hexahedron is on the boundary then only 2 elements will be split, but if



(a) (b) (c)

Figure 4.10: Splitting along longest edge in a patch

the hexahedron is internal to the mesh, then 4 elements will be split. Finally, if the hexahedron contains 24 elements, the longest edges are the edges of the hexahedron. Notice that if the edge is internal to the mesh, then 8 elements will be split. If the edge is on a boundary face of the mesh then 4 elements will be split, and if the edge is a boundary edge of the mesh then only 2 elements will be split.

A transition region between the fully refined elements to coarse elements ensures that the mesh maintains its conformity. Figure 4.11 shows a sample mesh where adaptive refinement has been applied at the center node to level 11. The elements that fall within the refinement radius are fully refined and shown in red. The coarse outside the region remain at level 0 and are shown in grey. The transition region maintains conformity between the fine and coarse elements and is shown in blue.

4.4.2 Implementation of adaptive mesh refinement with the TopS data structure

Each time a cohesive element is dynamically inserted to the mesh, the crack tips location change and the discretization of the problem must be changed accordingly. As described in Section 4.2.2, the application code utilizes the crack tip refinement manger of TopS to automatically track crack tips.

A flow chart of the procedure is shown in Figure 4.12 and it will be described here. After inserting cohesive elements and updating mesh coarsening (see Section 4.5), the application calls the TopS API function `topRefinement4K3D_UpdateMeshRefinement` to identify the new crack tips and potentially refine the mesh around them. TopS calls `Callback_CrackTipRefine`, which is implemented in the client application, and passes the location of a potential new crack tip. The application code notifies TopS if the region around this node should be refined. In the present implementation, the client confirms that all potential crack tip nodes are in fact crack tips, and allows refinement about all of them. After the client confirms that a node is a crack tip and should be refined, then TopS makes the geometric and topological changes to that region of the mesh. In doing so, TopS creates new elements and new nodes then calls `Callback_InsertNode` so that the client can update the field quantities as described in section 4.4.3. TopS passes the new node and its nodal data to the client application to `Callback_InsertNode`, which is implemented in the client application. Inside the client application the new node is assigned a node ID, then the elements adjacent to the node are gathered using the iterator, `topNodeElemItr`. These elements will also be newly added to the mesh (for the case of the Tetra 4 elements), so they are assigned element IDs. For each new element, the client application saves its parent element, i.e. the old element that was divided to get the new element, ignoring duplicates. The

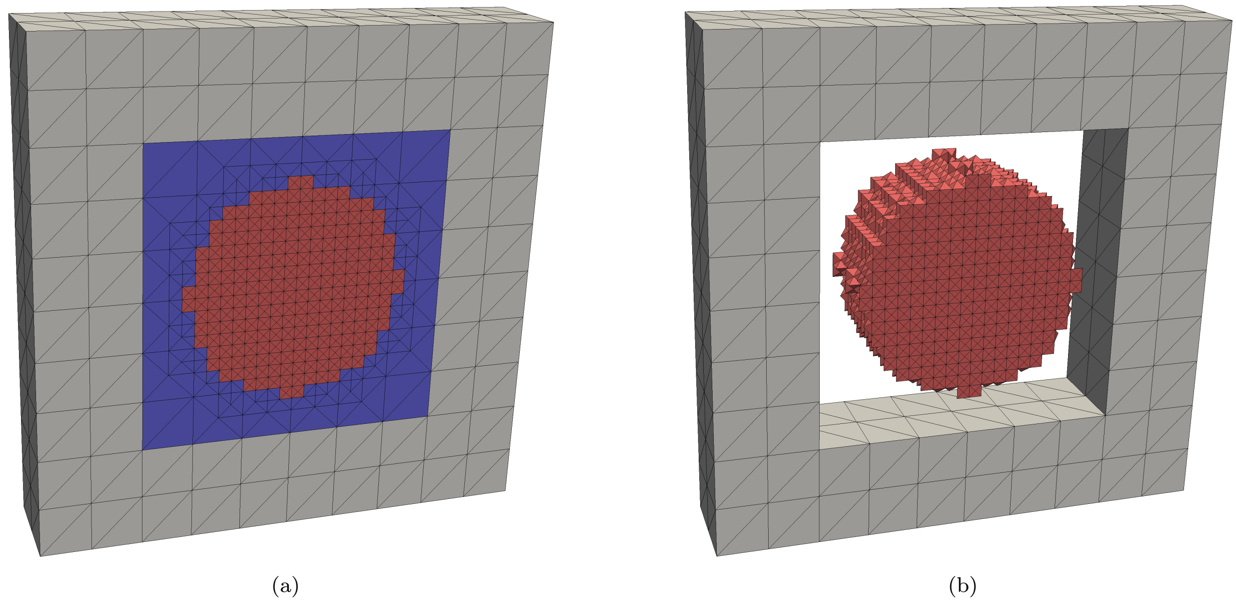


Figure 4.11: Mesh refined only in a select region to level 11. The elements that a fully refined to level 11 are shown in red, the coarse elements at level 0 are shown in grey, and the transition region of partially refined elements between levels 0 and 11 are shown in blue. (a) All levels (b) Transition region removed

parent elements are utilized to transfer field quantities to the new nodes, which will be described in Section 4.4.3.

4.4.3 Physical aspects of adaptive mesh refinement

Mesh refinement adds new nodes and elements to the finite element model; essentially we are performing local remeshing. As discussed in detail in Chapter 2, when remeshing occurs care must be taken in transferring node and element quantities from the old to the new mesh. In the work of this chapter we first concerned with simulating fracture in quasi-brittle materials where the bulk material model is elastic. Therefore, we do not have internal state variables in the bulk elements that must be mapped from the integration points of one mesh to another. We also do not map cohesive zone state variables that live at the integration points of the cohesive zone element. Mesh refinement is always done before cohesive elements are inserted, and the maximum depth of refinement is set from the beginning of the simulation, so cohesive elements are never refined, therefore mapping of their state variables is unnecessary. For more complex material models or continued refinement, however, internal state variables must be transferred. Therefore, the only variable that needs to be mapped here are nodal quantities. The state of the bulk material is completely characterized by the displacement field, so in fact, we only transfer the nodal displacements when new nodes and elements are inserted.

The nodes of new element's parent element are used to interpolate the displacement field using the standard finite element shape functions. A schematic showing the newly created node and elements and their parent elements is shown in Figure 4.13. The displacement of the white node is interpolated from the nodes of either of the light grey parent elements (only one of the parent elements is needed). Linear elements in a 4k grid are used in initial work, so the new nodes are only inserted at the midpoint of edges existing element edges. Therefore interpolation of new nodal quantities reduces to a simple averaging of the nodal

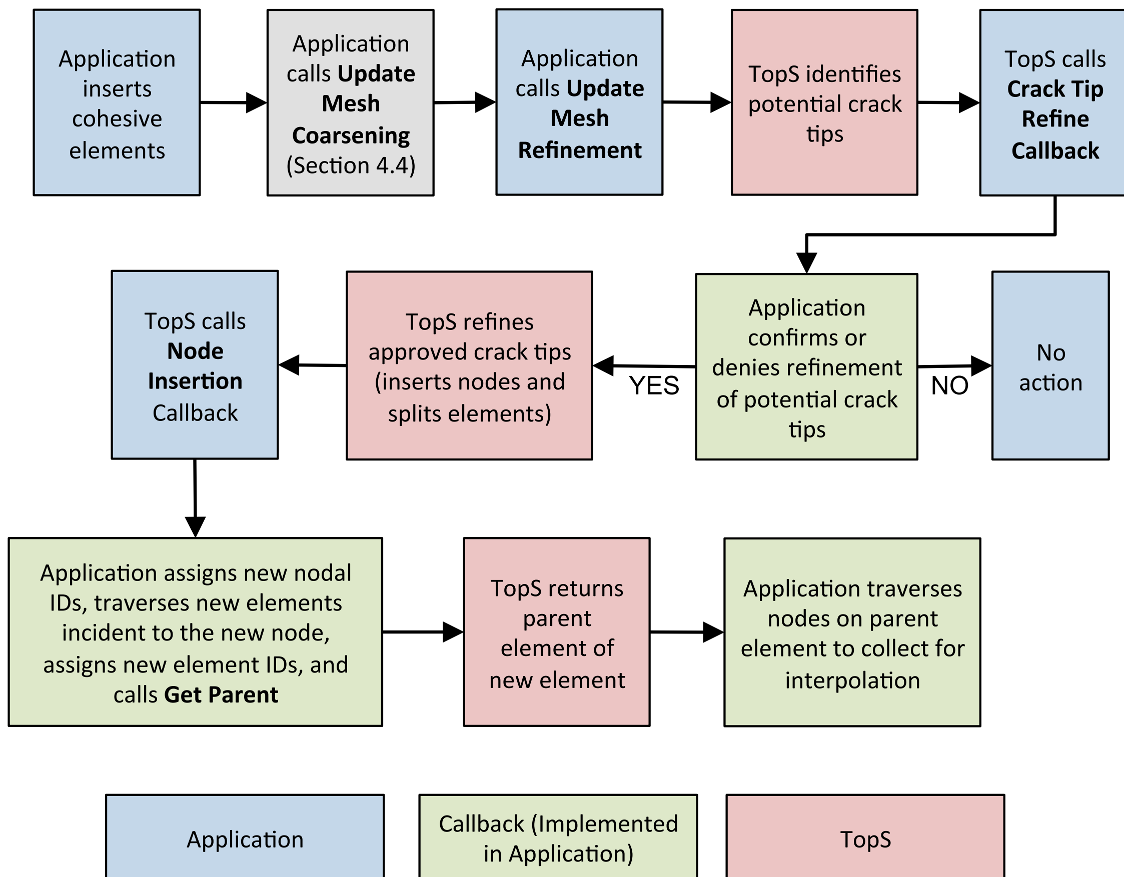


Figure 4.12: Flowchart of procedures to perform adaptive mesh refinement

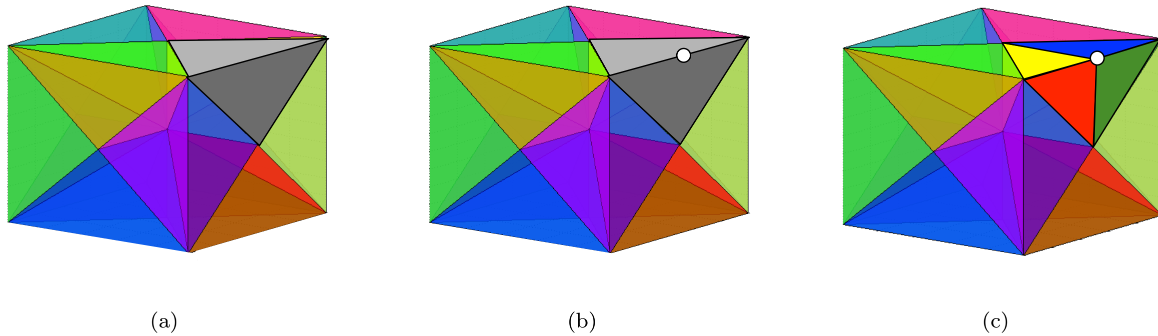


Figure 4.13: Schematic of refinement showing process, (a) two grey elements are refined by (b) insertion of new node, (c) grey (parent) elements are deleted new (child) elements are inserted

quantities at the ends of of the edge. However, we chose to keep the implementation general to allow for a straightforward extension to quadratic 4k elements or even to a non hierarchical refinement scheme.

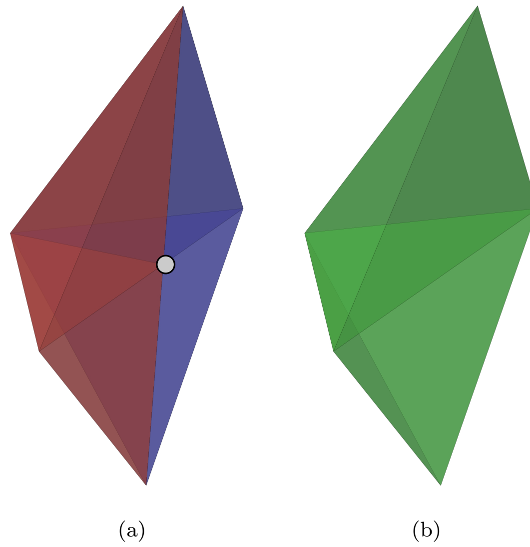
In addition to transferring nodal data, we also transfer some element level data. Certain element flags that indicate a state of an element are transferred from the parent to the child element. Boundary conditions are copied (or interpolated if appropriate) from the parent element to the child element. We also store the level of refinement of elements; the child's level of refinement is simply one level greater than the parent's level. Finally, the stiffness and mass matrices for the new elements are calculated, and new element mass matrices are used to compute the nodal masses.

4.5 Adaptive mesh coarsening

Additional computational savings are earned when adaptive mesh coarsening is included in the dynamic fracture simulation. We address the main geometric and physical aspects of adaptively coarsening the 3D 4k mesh and detail the numerical implementation into the application code.

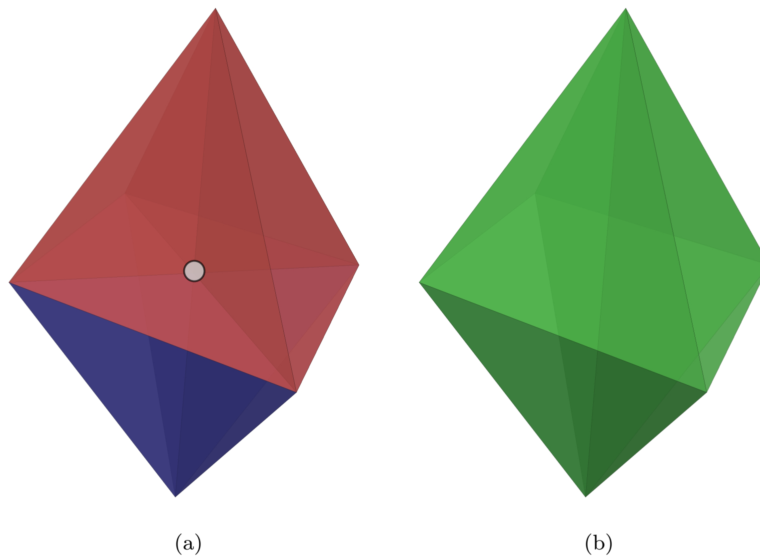
4.5.1 Geometric aspects of adaptive mesh coarsening

Once a crack tip advances, the refinement region associated with the node is destroyed and the elements associated with it are eligible for mesh coarsening. So essentially, any element that is no longer with a radius of refinement of a current crack tip is eligible for coarsening. Just as refinement was conducted in a level-by-level approach, coarsening is done the same way, but in reverse. Coarsening begins with elements at the highest level of refinement and moves down one level of refinement at a time. Coarsening checks will continue level-by-level until the elements reach refinement level 0 or until they no longer meet the criteria for mesh coarsening, which will be discussed in the next section. Due to stringent physical requirements on the activation of mesh coarsening, it is quite rare that a will a patch of elements actually be coarsened down to level 0. Elements are coarsened by removing nodes. Depending on the configuration of a patch of elements, a node may have 4, 8, 12, or 16 adjacent elements. Each of these configurations is shown in Figures 4.14-4.17,where airs of red and blue elements are shown and are coarsened to green elements. Coarsening a patch of elements only changes the connectivity internal to the patch, adjacent elements do not need to be updated if a node is removed.



(a) (b)

Figure 4.14: Case 1 for mesh coarsening: a patch of elements on a boundary is coarsened from (a) 4 elements to (b) 2 elements by removing the grey node. 2 pairs of red and blue elements are coarsened to 2 green elements



(a) (b)

Figure 4.15: Case 2 for mesh coarsening: a diamond shaped patch of (a) 8 elements is coarsened to (b) 4 elements by removing the internal grey node. 4 pairs of red and blue elements are coarsened to 4 green elements

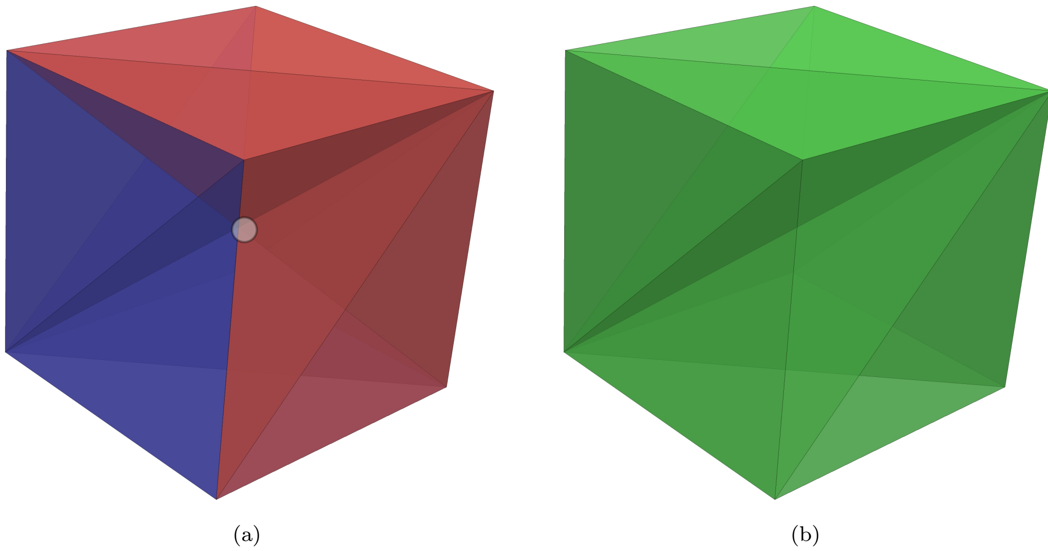


Figure 4.16: Case 3 for mesh coarsening: a hexahedron of (a) 12 elements is coarsened to (b) 6 elements by removing the internal grey node. 6 pairs of red and blue elements are coarsened to 6 green elements

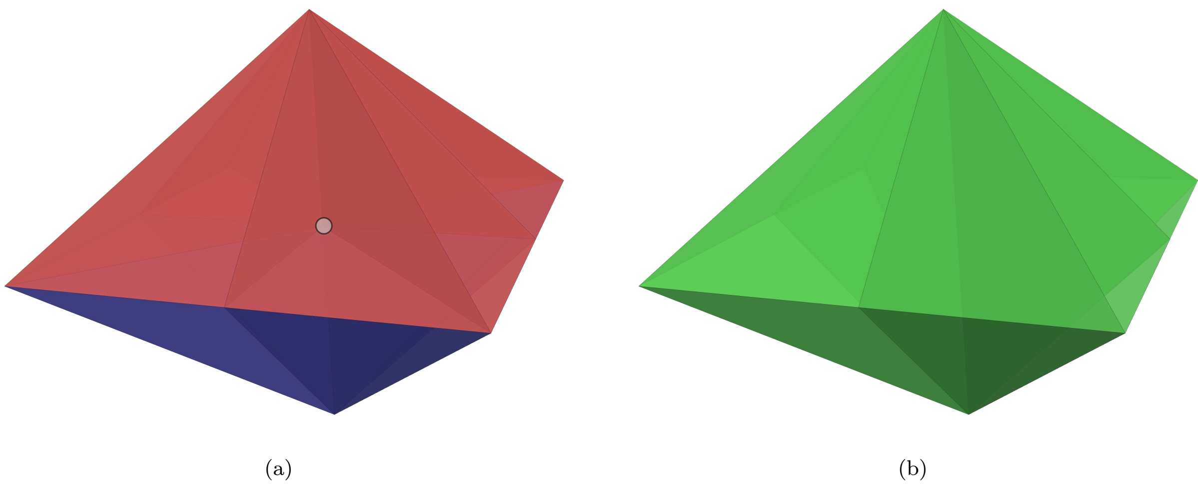


Figure 4.17: Case 4 for mesh coarsening: a diamond shaped patch of (a) 16 elements is coarsened to (b) 8 elements by removing the internal grey node. 8 pairs of red and blue elements are coarsened to 8 green elements

4.5.2 Implementation of adaptive mesh coarsening with the TopS data structure

Adaptive mesh coarsening is activated after cohesive elements are inserted in the mesh. A flow chart of the procedure indicating the work done by the application code, TopS, and the callback functions is shown in Figure 4.18. Notice that adaptive mesh coarsening takes place before adaptive mesh refinement, as indicated by the grey boxes in Figures 4.12 and 4.18, to avoid refining and coarsening the same regions many times. After inserting cohesive elements, the application calls the TopS API function `topRefinement4K3D_UpdateMeshCoarsening` to identify previous refinement regions (i.e. crack tips) and release them for coarsening. Tops calls `Callback_CanCollapseNode` from which the application notifies TopS if the region can be released for coarsening. In the current implementation, the application releases all non-crack tip regions for coarsening. However, coarsening will not actually occur unless the patch of elements meets the coarsening criteria, so there is no harm in physically changing the problem by releasing elements. Next, TopS calls `Callback_CanCollapseNode`, and passes the node that could be removed, the patch of elements that would be coarsened, and the the final coarsened elements (if the coarsening criteria were to be met). The application code notifies TopS if the node may be removed and elements coarsened using the strain criteria discussed in Section 4.5.3. Then, TopS makes the geometric and topological changes to that region of the mesh. In doing so, TopS deletes nodes and elements and calls a callback function for each operation. In `Callback_MergeElements` the client copies element data from the old refinement elements to the new coarse elements, assigns IDs to the new elements, and frees the memory associated with the deleted elements' attributes. Then in `Callback_RemoveNode` the client simply frees the memory associated with the deleted node's attribute.

4.5.3 Physical aspects of adaptive mesh coarsening

We adopt a local strain error criteria to determine if a patch should be coarsened [158, 180]. In the Can Collapse Node callback function, discussed in the previous section, the application computes the strain on the patch of elements comprised of the refined elements and the coarse elements. To maintain some computational efficiency, the strain is only computed on the patch and does not take in to account the entire mesh or larger encompassing region. The error is simply the norm of the difference of the strains, i.e.,

$$e_{\text{patch}} = \left\| \varepsilon_{\text{refined}} - \varepsilon_{\text{coarsened}} \right\| . \tag{4.1}$$

If the error is less than a user-prescribed tolerance, then the node is removed and the elements are coarsened. Since the coarsening scheme removes nodes from the model, there is a reduction in the finite element space and the coarse element representation will not be able to exactly represent the fine field. Thus, the tolerance level must be chosen carefully, such that too much information is not lost. Parametric studies to determine appropriate tolerance levels are performed on the numerical examples in Section 4.6.2. In the interest of gain computational efficiency, we accept some of loss in accuracy due to coarsening, but we ensure it is reasonable with a carefully selected tolerance.

Since we use linear tetrahedra is this work, it is not necessary to transfer any nodal variables from the fine patch to the coarse patch, we simply remove the node if the strain error meets the tolerance criteria.

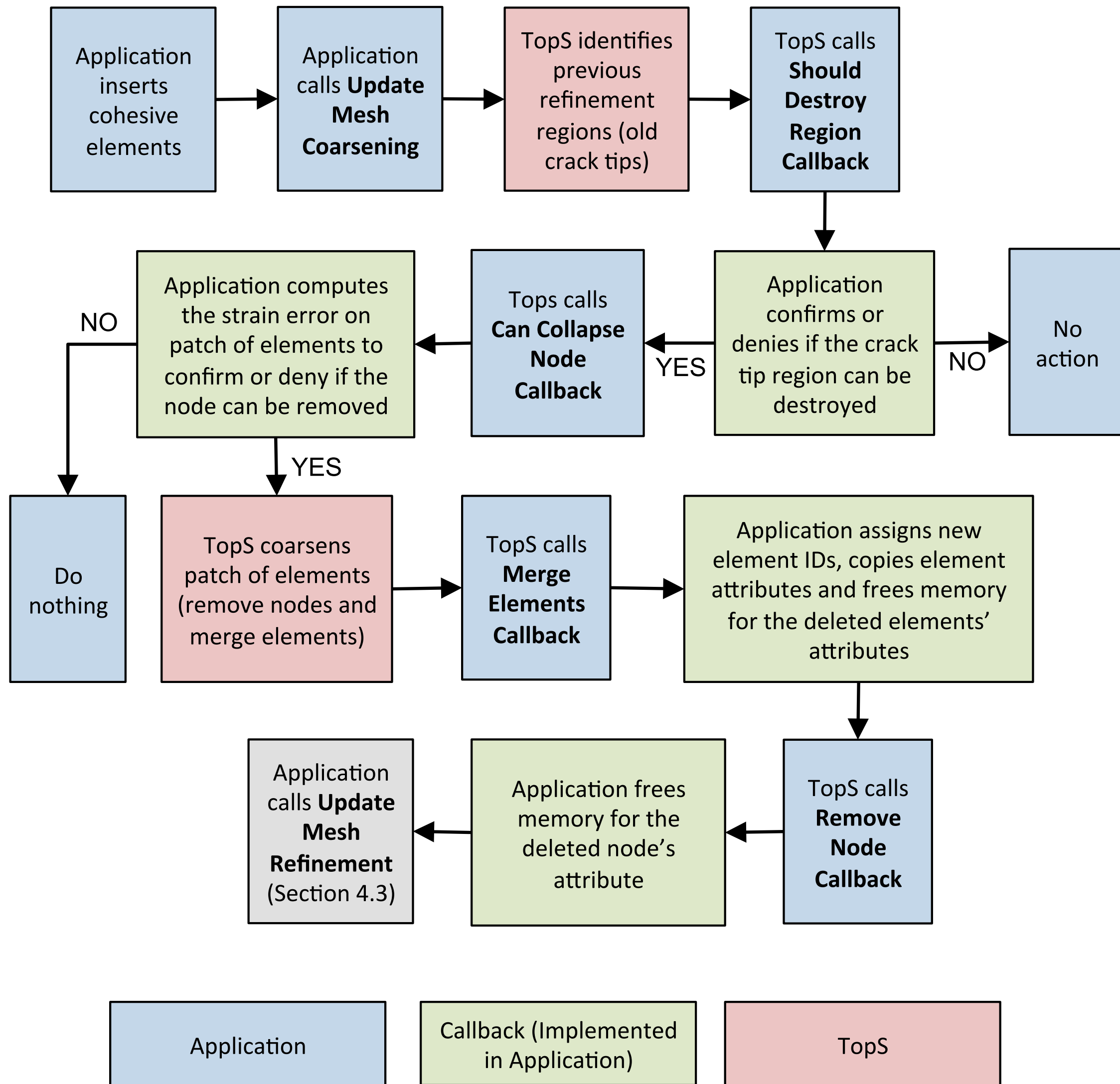


Figure 4.18: Flowchart of procedures to perform adaptive mesh coarsening

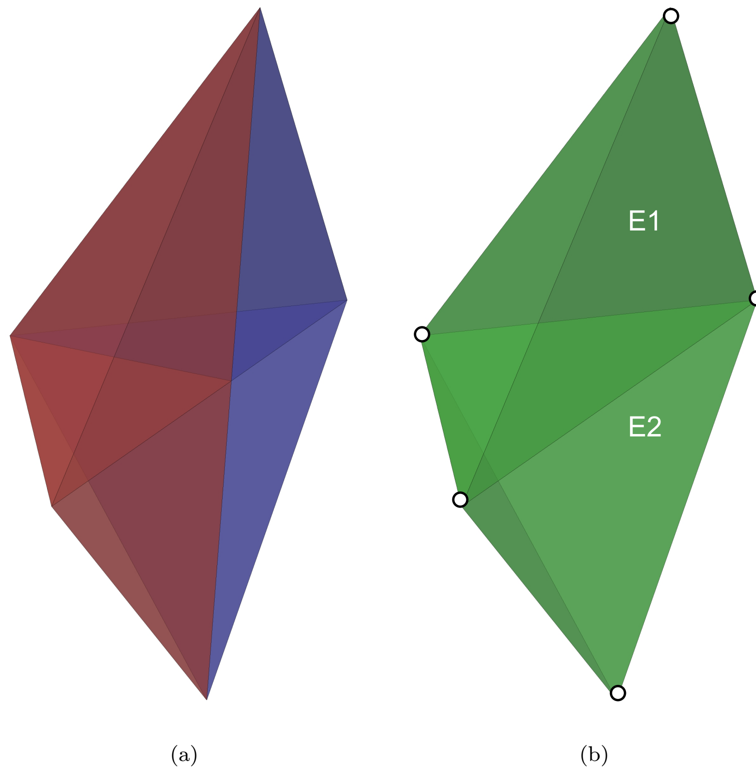


Figure 4.19: Mesh coarsening results in removal of nodes and replacement of elements. (a) Original, refined patch of 4 elements, e1, e2, e3, e4 contains 6 nodes (b) Coarsened patch contains 2 elements, E1 and E2, and 5 nodes

4.6 Numerical simulations

Next, we investigate two benchmark examples to gain some insight into the 3D adaptive mesh refinement and coarsening schemes developed in this chapter. Verification of the implementation on problems with known experimental or numerical results will give confidence that our fracture simulation framework could be used in a predictive setting. All examples in this chapter were simulated on the Trestles super computer, which is part of the fleet of XSEDE high performance computing machines. Each compute node contains four sockets, each with a 8-core 2.4 GHz AMD Magny-Cours processor, for a total of 32 cores per node and 10,368 total cores for the system. Each node has 64 GB of DDR3 RAM, with a theoretical memory bandwidth of 171 Gb/s. Before discussing the numerical results, however, we will first discuss some details related to visualization of the 3D adaptive fracture results.

4.6.1 Post-processing and visualization

Visualization of large data sets is quite cumbersome and is not easily handled by all finite element visualization software, thus it is imperative that appropriate post-processing software is selected. We use the ParaView software, an open-source multi-platform data analysis and visualization application developed by Los Alamos and Sandia National Laboratories [181]. Specifically, we use the ParaView Desktop Interface which is available for multiple operating systems and provides users the means to easily open and visualize large data sets.

ParaView is equipped with readers for an array of file types from common structural engineering software kits. Since all of the work in this dissertation is done with user-developed codes, we choose to write our output in the simple VTK (Visual Toolkit) file format. A useful resource for writing VTK files can be found in [182].

With the VTK files, we can easily visualize the deformed finite element mesh and any scalar, vector, or tensor quantity on the mesh nodes or elements. Many element types are supported, and many may be visualized at the same time. For example, we commonly plot tetrahedron (3D bulk elements) and triangles (2D interface elements) at the same time. The visualization tools including the slice, clip and threshold are used frequently to interpret results that would otherwise be impossible to visualize on a 3D geometry. For example, to visualize the crack pattern, we plot the cohesive elements with a normal separation above a certain value using the threshold tool.

Even with powerful rendering software, visualization of the mesh refinement and coarsening scheme in 3D can be difficult. To aid in understanding the geometrics and subsequent physical consequences of refining and coarsening, we 3D printed a number of tetrahedral elements at different levels of refinement, as shown in Figure 4.20. The elements needed to be represented by the STL file format in order for the 3D printer to generate the final objects. We used open-source software to generate the STL files, starting from the standard VTK. From Paraview, the files were exported to the X3D file format and opened in the 3D graphics and animation software, Blender. From there, they were converted to STL files that were sent to the 3D printer.

4.6.2 Confined crack

The first problem we examine is beam with an initial notch subjected to tensile strain loading. The insertion of cohesive elements is confined to a single plane, which corresponds to the plane of the initial notch. The two-dimensional version of this problem was also investigated in [158] for verification of adaptive topological operators. A schematic of the 3D specimen with the dimensions, loading, and boundary conditions is shown

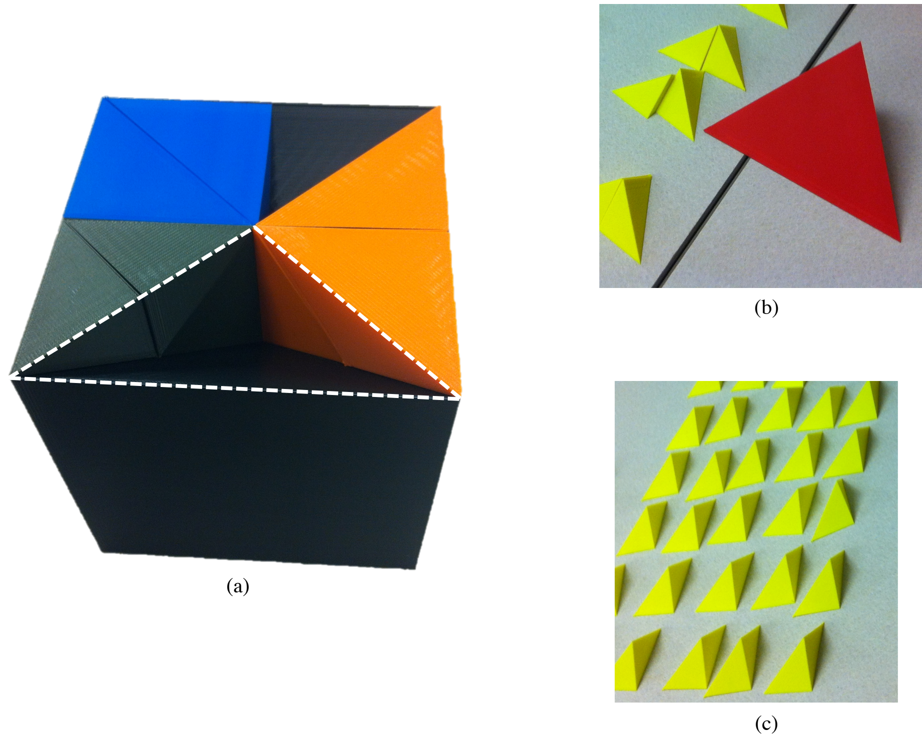


Figure 4.20: 3D printed tetrahedra: red - level 2, blue - level 3, orange - level 4, green - level 5, yellow - level 6. The black hexahedron is equivalent to 24 red, 48 blue, 96 orange, 192 green, or 384 yellow tetrahedra. (a) The black hexahedron with top face removed, the white dashed line indicates the space where one red tetrahedra would fit. The hexahedron is shown with 2 blue tetrahedra, 4 orange tetrahedra, and 4 green tetrahedra. (b) A red tetrahedra (level 2) next to pairs of yellow tetrahedra (level 6). (c) The yellow tetrahedra are oriented, and the left "sides" are pictured; the blue tetrahedra are also oriented, as evident from their position in (a)

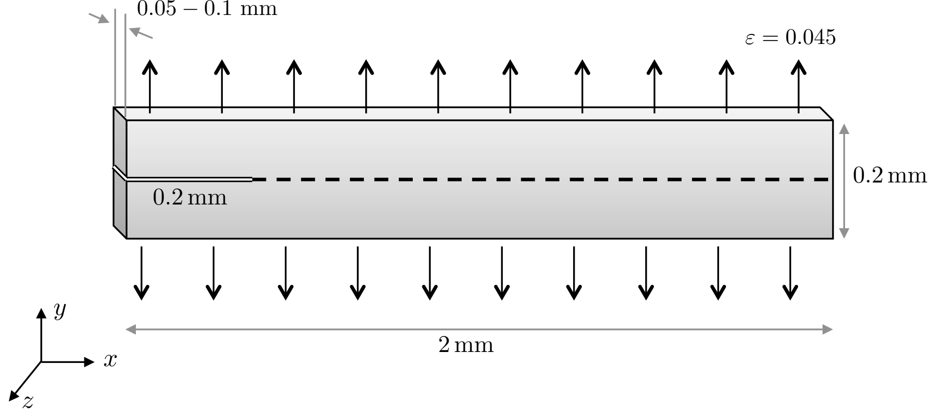


Figure 4.21: Schematic of confined notched beam subjected to strain loading

in Figure 4.21. The linear elastic bulk material has a modulus of 3.25e9 Pa, Poisson ratio of 0.3, density of 1190 $^{kg}/_{m^3}$. The fracture properties are the same in all modes: fracture energy of 352 $^{N}/_{m}$, cohesive strength of 324e6 Pa, softening shape parameter of 2, and unloading shape parameter of 1. The specimen is loaded with a strain of 0.045 in the y-direction, as shown in Figure 4.21, and out of plane deformation (i.e. z-direction) is restricted.

This problem will give us insight into several aspects of the 3D adaptive framework, namely it will allow for

- verification of the accuracy of the implementation
- comparison of the AMR and AMR+C to a case of fully uniform refinement
- investigation into the effects of transitioning from a coarse to fine mesh through parameterization of the initial element size
- examination of 3D effects through parameterization of the specimen thickness
- examination of the accumulation of error in mesh coarsening through parameterization of coarsening tolerance

The cases we investigated are listed in Table 4.1. On the thick specimen with the coarse initial grid, the domain is comprised of $40 \times 4 \times 2$ hexahedron, each of which is at Level 0 refinement meaning that it contains 6 tetrahedron. Therefore, the initial coarse grid contains 1920 tetrahedral elements of maximum edge size 50 μm. In the fully refined case all elements are refined to Level 11 resulting in a fine grid of $320 \times 32 \times 16$ hexahedron, each of which contains 24 tetrahedron of maximum edge size 6.25 μm. The fully refined mesh contains 3,932,160 elements. In the adaptive cases the initial coarse grid is utilized then a region of radius 50 μm is refined to Level 11, which results in 95,408 tetrahedral elements. The finer initial grid contains $80 \times 8 \times 4$ hexahedron with a region of radius 50 μm refined to Level 8 resulting in 112,064 tetrahedral elements of maximum edge size 25 μm. The finest level of initial refinement contains an initial grid of $160 \times 16 \times 8$ hexahedron with a region of radius 50 μm refined to Level 5 resulting in 216,768 tetrahedral elements of maximum edge size 12.5 μm. The thinner specimens are constructed in a similar manner. In all cases, the number of elements will increase as cohesive elements are inserted. However, only in the AMR and AMR+C cases will the number of bulk elements increase as the simulation progresses. The initial meshes on

thick specimens with a coarser initial mesh are shown in Figure 4.22 for illustrative purposes, and the mesh statistics for all cases investigated are shown in Table 4.1. The progression of the propagation using AMR and AMR+C are shown on the thick specimen in Figure 4.23.

Ideally, the numerical results with adaptivity would be identical to those of a uniform refinement. Of course this is not possible from a numerical perspective, so we evaluate the differences and examine the costs and benefits of AMR and AMR+C. We first check the velocity of the crack tip on the thicker specimen and compare the uniform refinement, where all elements have a long edge length of 6.25μm to AMR with an initial coarse mesh of edge size 12.5μm, 25μm and 50μm. As shown in Figure 4.24, the coarsest level of refinement results in a crack tip velocity that is significantly slower than the finer cases, ~7.8e4 m/s for the coarser case compared to ~25 m/s for the finer cases. Based on these results, we see that care must be taken to create an initial discretization that is sufficiently fine to accurately capture the wave speed of the materials. We note that specimen thickness had little difference on the crack tip velocity, but load is applied uniformly through the thickness so this is expected.

Now, given a sufficiently fine initial mesh, we compare the crack tip position versus time for the uniform, AMR and AMR+C cases. As shown in Figure 4.25, the addition of mesh coarsening does not have an impact on the velocity, no matter what how stringent the coarsening tolerance.

We also examine the energy evolution of the AMR and AMR+C. We observe some in loss total energy of the AMR and AMR+C when compared to the uniform refinement, but this diminishes when a finer initial mesh is used and the transition between the coarsest elements and finest elements is small. Greater loss when a more relaxed tolerance is used on the AMR+C, as expected, so a decision between computational efficiency and numerical accuracy must be made by the user, we recommend a value in strain error between 0.001 and 0.0001 to maintain a reasonable level of accuracy from an energy perspective.

For visualization purposes, we also compare the stress on the uniform, AMR and AMR+C mesh part way through the simulation in Figure 4.27. Near the crack tip the resolution of all three cases is similar, the main difference is slightly away from the crack tip. The AMR and AMR+C cases do not resolve the stress in quite as much detail, however we are willing to sacrifice this for the gains in computational time, which we will discuss next.

The AMR and AMR+C provide a great savings in computational time, however there is a tradeoff in where the computational time is spent. When adaptivity is employed, there are fewer element and node calculations throughout the simulation. When we employ AMR, the rate of the wall time slows as the simulation progresses because we are continuously adding elements without removing them. Note, that we are discussing the rate of the wall time, not the simulation time. The velocity of the crack is not changed as discussed previously, but the time to run simulation is impacted by the adaptivity. The AMR+C alleviates the rate of slowing of the AMR because elements and nodes are deleted as the simulation progress, so not as many calculations are needed as in the AMR case. However, not all of that time is gained because with the AMR+C we perform more computations to check on strain error on patches of elements. If a stringent tolerance is used, then many of patches will not be coarsened. Thus, we only add time to the AMR case by performing strain error comparison calculations without making the problem size smaller. This is demonstrated in Figure 4.28, which shows the crack tip position versus wall time. The total time for the uniform case is significantly higher than any of the adaptive cases and the rate of insertion is close to linear. However, in the AMR case or AMR+C case with a stringent tolerance, the rate at which cohesive elements are inserted with respect to the wall time decreases as more bulk elements are inserted and not removed. The rate of insertion remains closer to constant with a looser tolerance on the coarsening. Notice than when the

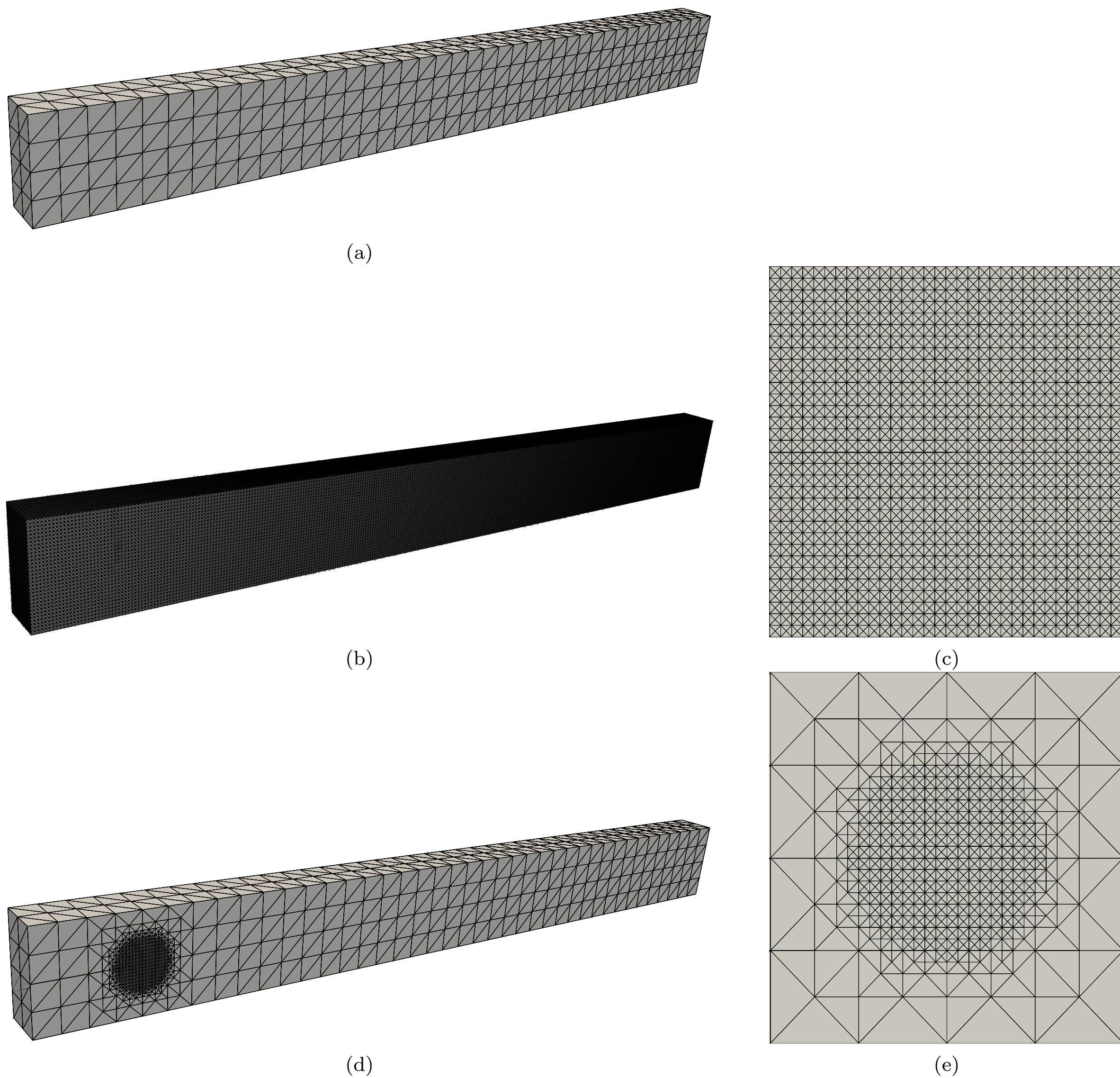
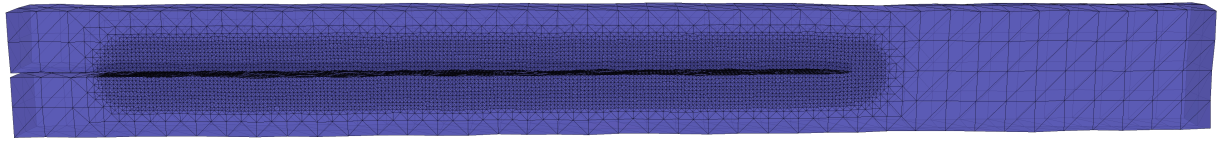
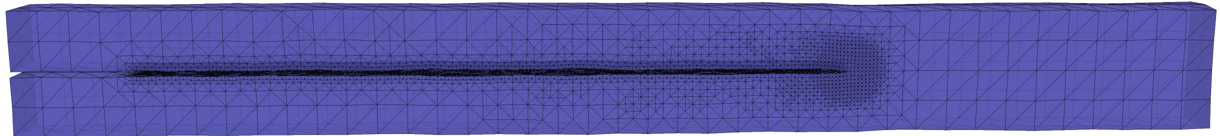


Figure 4.22: Initial meshes for the 0.1mm thick notched beam specimen. (a) Coarse grid which is the basis for each case investigated (b) Fully refined case where all elements of the coarse grid are refined to Level 11 (c) Zoom in around crack tip of fully refined case (d) AMR and AMR+C case where elements in the region of the notch tip are refined to Level 11 (e) Zoom in around crack tip of AMR and AMR+C case



(a)



(b)

Figure 4.23: Finite element mesh of the 0.1 mm thick specimen at 210 nanoseconds (7000 steps) on the (a) AMR with coarse initial mesh (b) AMR+C with initial mesh of resolution 50 μm and coarsening tolerance of 0.01

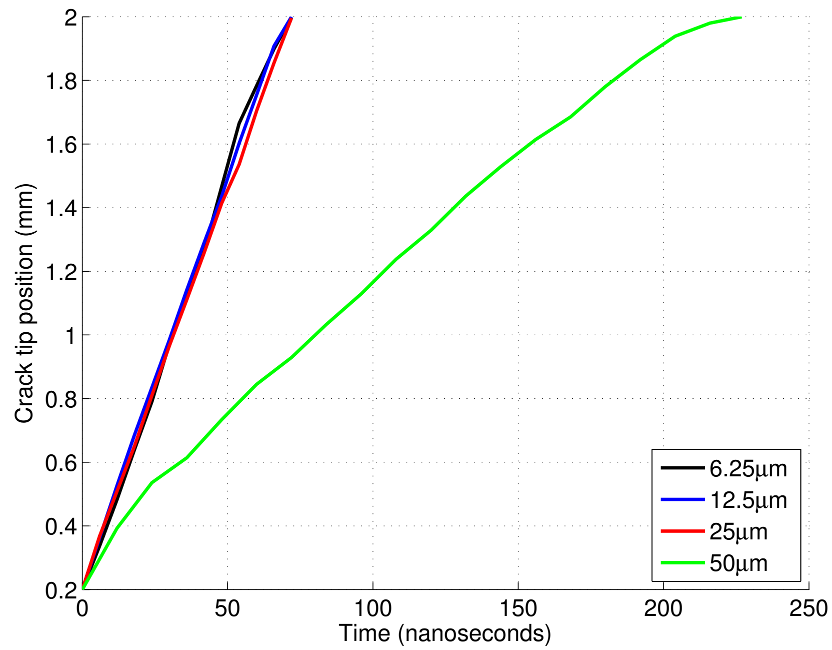


Figure 4.24: Comparison of velocity on 0.1mm thick specimen for various levels of coarse level refinement

Table 4.1: Summary of computational resource usage in confined crack problem

Specimen Thickness	Adaptivity Type	Coarse Mesh Resolution	Coarsening Tolerance	Nodes		Elements		Run time* (HH:MM)	Maximum memory usage**
				Initial	Final	Initial	Final		
0.1 mm	Uniform	6.25 μm	n/a	852,369	861,890	3,932,160	3,969,024	43:37	5.11 GB
0.1 mm	AMR	12.5 μm	n/a	44,840	448,893	216,768	2,105,060	10:59	2.69 GB
0.1 mm	AMR+C	12.5 μm	0.001	44,840	200,823	216,768	1,056,034	24:50	1.37 GB
0.1 mm	AMR	25 μm	n/a	24,279	440,581	112,064	2,065,808	11:44	2.64 GB
0.1 mm	AMR+C	25 μm	0.01	24,279	71,886	112,064	364,734	6:21	513.7 MB
0.1 mm	AMR+C	25 μm	0.005	24,279	96,218	112,064	495,260	7:47	676.2 MB
0.1 mm	AMR+C	25 μm	0.001	24,279	182,951	112,064	970,866	12:34	1.26 GB
0.1 mm	AMR+C	25 μm	0.0005	24,279	251,674	112,064	1,302,602	15:01	1.67 GB
0.1 mm	AMR+C	25 μm	0.0001	24,279	391,999	112,064	1,888,808	19:05	2.39 GB
0.1 mm	AMR	50 μm	n/a	20,565	439,556	95,408	2,062,110	36:53	1.22 GB
0.1 mm	AMR+C	50 μm	0.001	20,565	104,487	95,408	536,428	23:35	515.4 MB
0.05 mm	Uniform	6.25 μm	n/a	436,617	441,522	1,966,080	1,984,512	18:58	2.56 GB
0.05 mm	AMR	25 μm	n/a	12,787	225,524	56,032	1,030,664	5:49	1.32 GB
0.05 mm	AMR+C	25 μm	0.01	12,787	35,420	56,032	171,786	2:42	388.7 MB
0.05 mm	AMR+C	25 μm	0.005	12,787	45,636	56,032	225,062	3:58	432.2 MB
0.05 mm	AMR+C	25 μm	0.001	12,787	86,884	56,032	442,292	6:07	667.3 MB
0.05 mm	AMR+C	25 μm	0.0005	12,787	114,898	56,032	581,218	7:11	755.7 MB
0.05 mm	AMR+C	25 μm	0.0001	12,787	190,442	56,032	907,164	8:56	1.16 GB
0.05 mm	AMR	50 μm	n/a	10,610	225,803	47,340	1,032,072	6:10	1.32 GB
0.05 mm	AMR+C	50 μm	0.001	10,610	84,812	47,340	431,694	6:40	567.7 MB

* Includes time to write output files

** Only includes size of nodes and elements and their attributes, does not include overhead

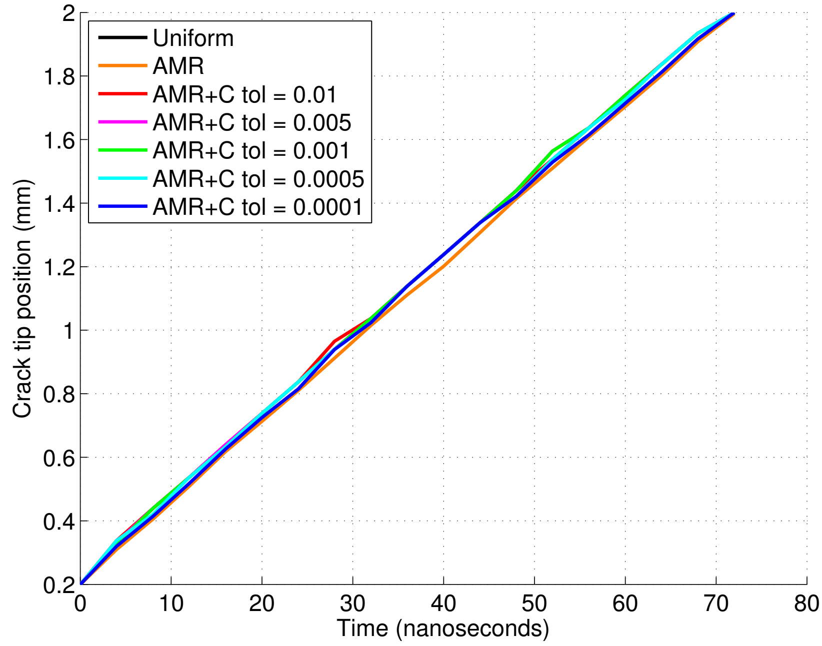


Figure 4.25: Crack tip position versus simulation time for the 0.1 mm thick specimen

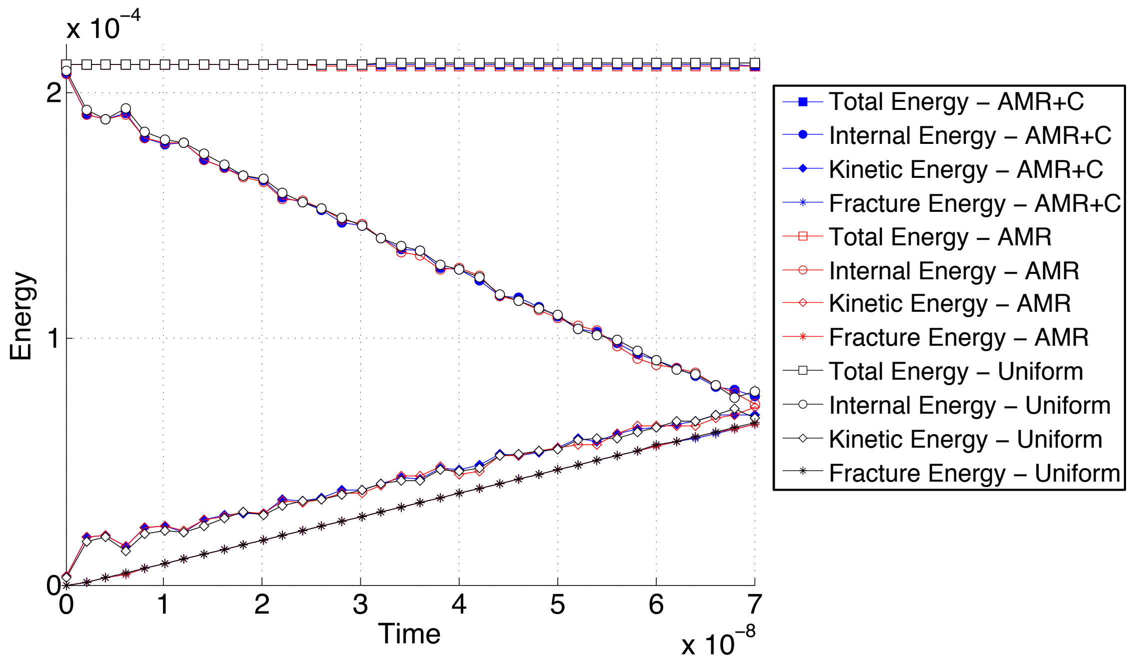


Figure 4.26: Energy evolution for the confined crack problem on the 0.1 mm thick specimen with an initial coarse mesh resolution of 12.5 μm. The coarsening tolerance for the AMR+C case is 0.001. (Add figure of finer initial mesh with AMR and AMR+C)

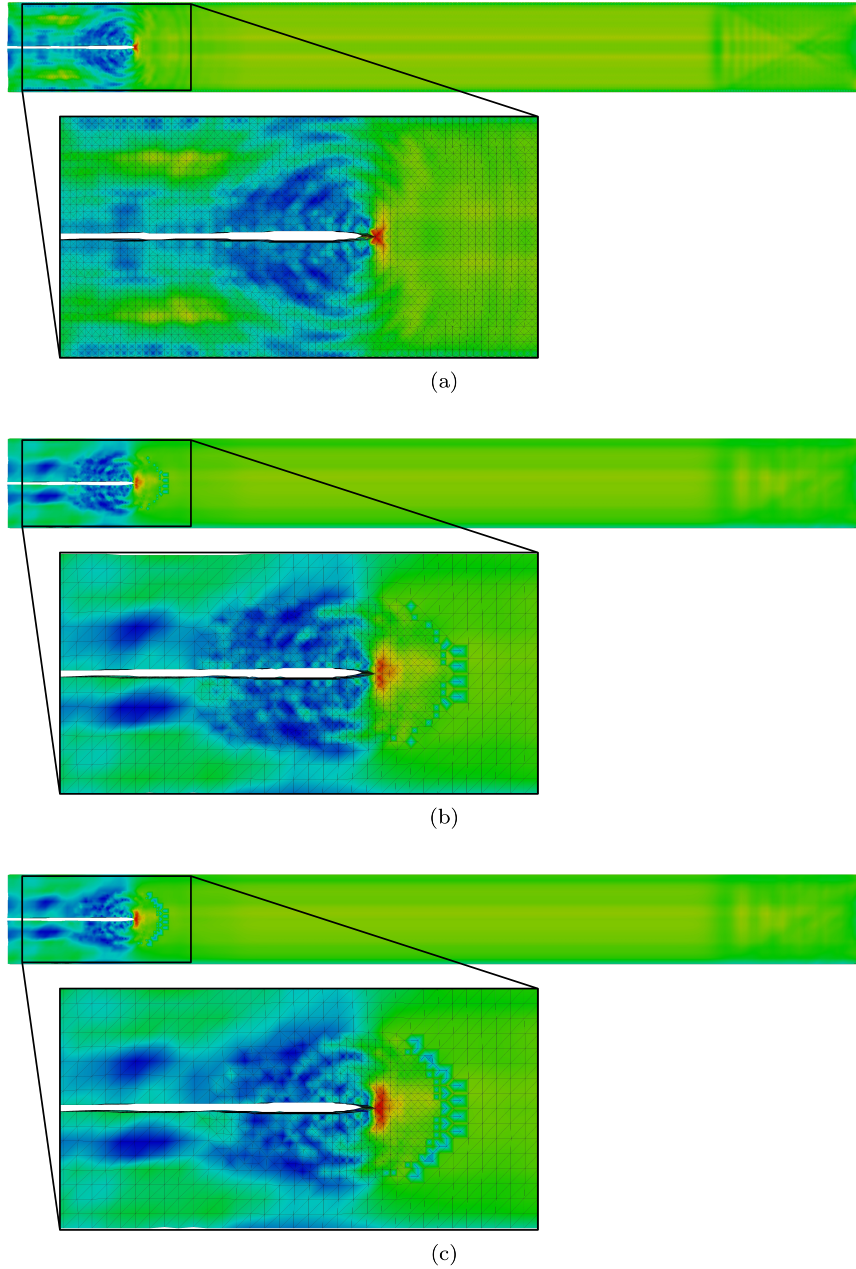


Figure 4.27: Stress σ_{yy} at 15 nanoseconds (500 steps) on the 0.1 mm thick specimen with a (a) uniform mesh (b) AMR with fine mesh and (c) AMR+C with initial mesh of resolution 12.5 μm and a coarsening tolerance of 0.01, shown on the x-y face of 3D meshes

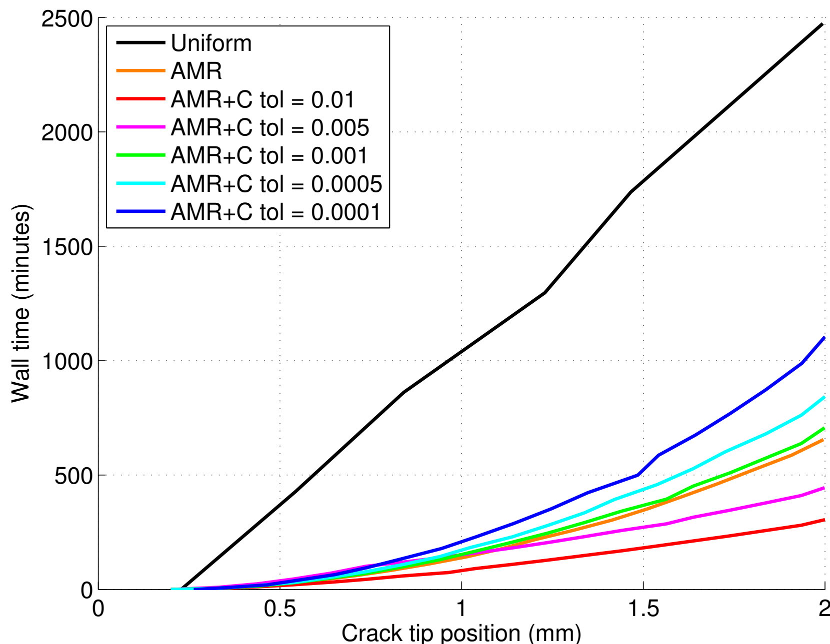


Figure 4.28: Crack tip position versus wall time for the 0.1 mm thick specimen with AMR and AMR+C with the loosest coarsening tolerance of 0.01. (Just for demonstrative purposes)

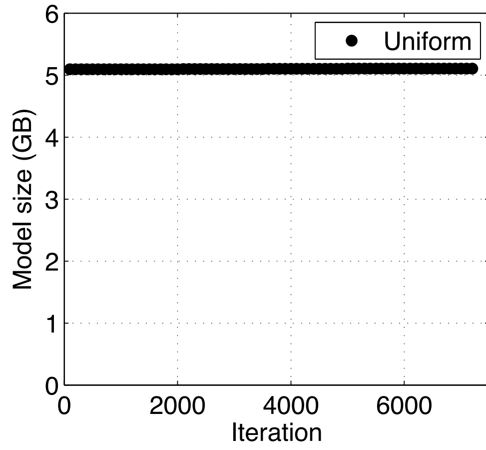
tolerance is tighter than 0.001 the total wall time is longer and rate of insertion slower than even the AMR case even though the final numbers of elements and nodes are lower (see Table 4.1 for number of elements and nodes).

Lastly, we examine the evolution of memory usage as the crack propagates. For the uniform case, the memory usage stays mostly constant except for the minimal insertion of cohesive elements. The AMR+C with a relaxed tolerance is the most memory efficient as the memory associated with elements and nodes (and their attributes) is freed often. Memory for the AMR and AMR+C with a tight tolerance continually increases throughout the simulation since elements and nodes are either not deleted at all or not deleted often. This is an important practical consideration when allocating memory for AMR problems; it is necessary to estimate the final model size and memory demands to avoid exceeding memory allowances.

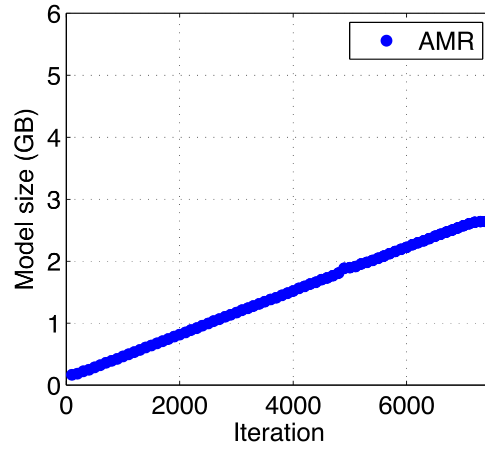
The results with the uniform refinement, AMR, and AMR+C on the confined crack problem give confidence that the implementation is working properly and can be extended to a fully unconstrained, mixed-mode problem. The main challenges of three-dimensional adaptivity were addressed in this problem. Briefly, those challenges are

- tracking multiple crack tips to manage regions of refinement,
- transferring the displacement field from the coarse regions to the fine regions, and
- evaluating error in coarsening from fine regions to coarse regions.

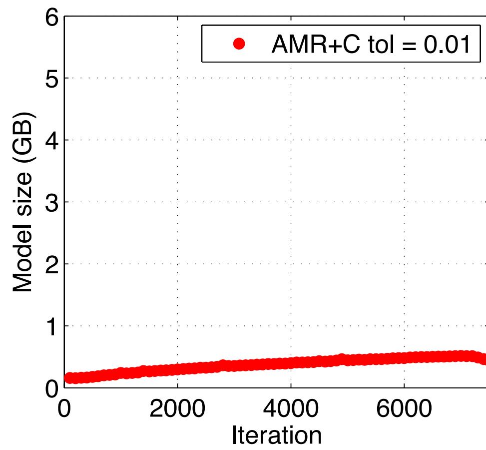
Even though the crack is planar, each node at the crack front is a crack tip that has an associated region of refinement. The number of crack tips and therefore refinement regions is the number of nodes that span the width of the beam in this example. Therefore, the crack tip tracking methodology utilized for the problem



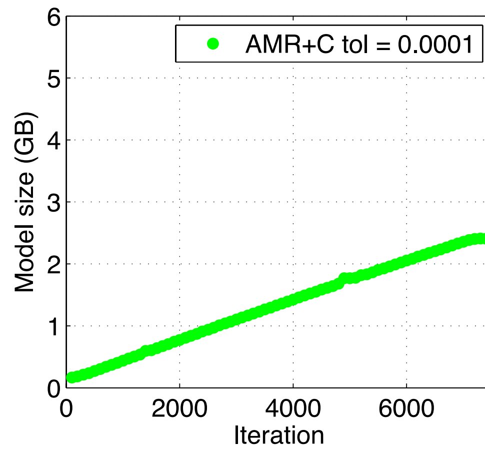
(a)



(b)



(c)



(d)

Figure 4.29: Model size versus time for the 0.1 mm thick specimen with (a) uniform refinement (b) AMR (c) AMR+C with coarsening tolerance of 0.01 (d) AMR+C with coarsening tolerance of 0.0001

is suitable when the crack is unconstrained and multiple crack fronts are present. Moreover, the transfer of data between discretizations is completely independent of the mode of fracture or confinement of cracking, therefore, the same methodology can be utilized in more complicated problems.

4.6.3 Three point bend notched beam

The second example we investigate is that three point bend (TPB) problem. A schematic of the problem and expected fracture behavior is shown in Figure 4.30. The rectangular domain has an off-center notch through the thickness at the bottom of the specimen and a velocity load at the top center. Previous studies have investigated the degree of mode mixity that occurs as the notch location varies [153, 183]. Beyond a certain distance from center the, crack does not initiate from the notch and instead initiates below the application of the load as a primarily mode I crack. When close enough, the crack initiates from the notch and propagates up and across towards the load application locations, i.e. mixed mode.

Different types of mesh discretization were also investigated in [153]. Structured and unstructured meshes were investigated and, as expected, the unstructured meshes performed better at capturing the true fracture pattern. In the present work, we are concerned with the structured 4k mesh because it allows for efficient mesh refinement and coarsening. Thus, we realize that the resulting fracture pattern on the structured mesh will not be as realistic as one on an unstructured mesh, however since we are most concerned with computational speedup, we accept some loss in accuracy of the crack pattern.

In previous investigations of this problem, uniform discretization are employed where a finer discretization is utilized in the region of the expected fracture path. This is in contrast to the approach taken in this work where we initially discretize the domain with a coarse mesh and use adaptive refinement to achieve the necessary level of refinement where needed. Based on the recommendation in [153] and an extensive parametric study conducted for this work, we selected the smallest element size as 2mm. We investigated using smaller levels of refinement, however due to limitations of computational resources, smaller elements were not feasible. If we were to use a uniform level of refinement, a smaller element size would have required more RAM than was available with the given resources, which could have been overcome by the AMR+C because significantly less RAM is required. For example, if the mesh were to be discretized with 224x72x24 4k patches or 9,805,824 elements the entire model, including overhead, would have required approximately 47 GB of RAM, but the AMR+C version would have required only 610 MB of RAM at the start of the simulation. Unfortunately, even with AMR+C the finer model was not possible to execute because the time step restriction made the over all wall time greater than 48 hours, which is the run time limit of the super computing resources available for this study. Therefore, given these limitations, we choose a coarser mesh so that we could evaluate the AMR+C scheme in a mixed mode setting.

The specimen is discretized into two different initial meshes with the same level of fine discretization at the crack tips:

- Coarse far-field mesh: $28 \times 9 \times 3$ with refinement to level 8 (2mm element size) in a radius of 6mm around a crack tip
- Fine far-field mesh: $56 \times 18 \times 6$ with refinement to level 5 (2mm element size) in a radius of 6mm around a crack tip

In total, we investigate four cases: coarse far-field mesh with AMR and AMR+C and the fine far-field mesh with AMR and AMR+C. The displacement field and final meshes for all cases are shown in Figure 4.31.

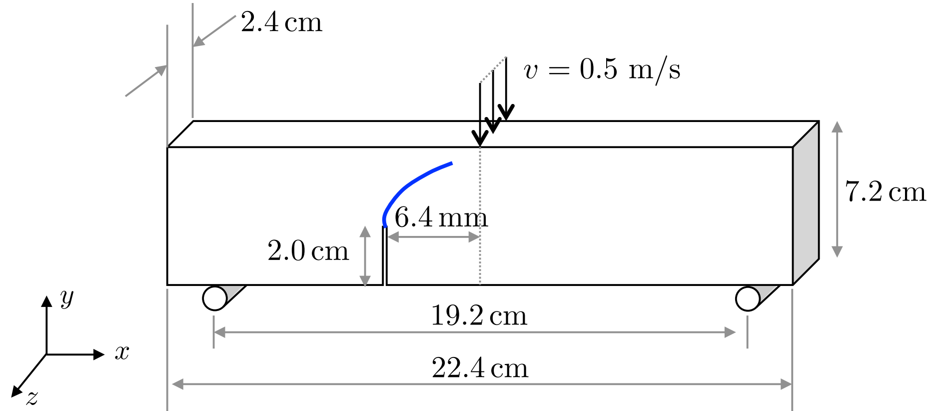


Figure 4.30: Three-point-bend beam (TPB) specimen schematic with expected fracture pattern shown in blue

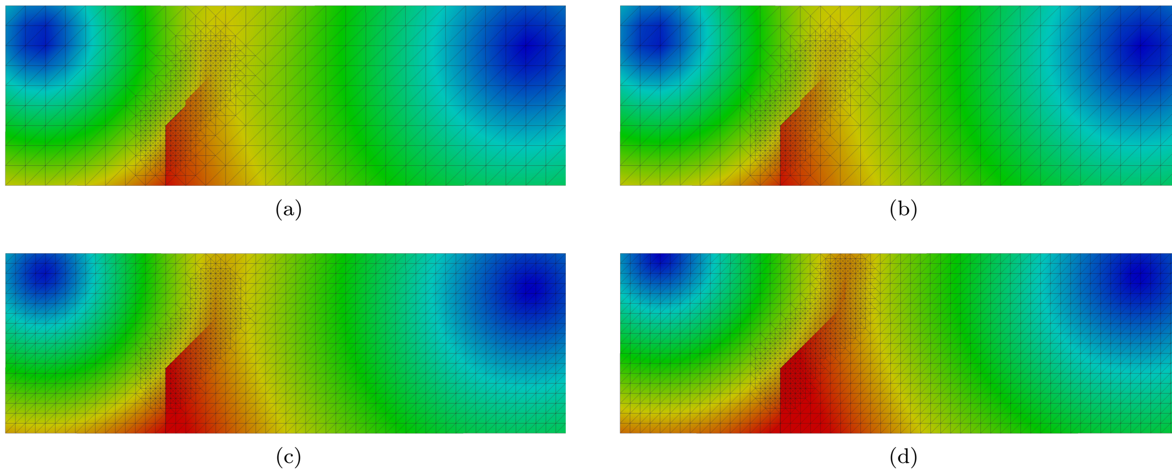


Figure 4.31: Displacement field at 0.0072 sec on coarse initial mesh with (a) AMR (b) AMR+C and at 0.0071 sec on fine initial mesh with (c) AMR (d) AMR+C

The displacement fields and crack patterns for each case are similar as evident by the refinement pattern and displacement discontinuity across the crack pattern. The crack kinks from mode I to mixed mode at the same location in all cases, but in the finer cases, the second kink occurs later than it does for the coarser cases.

To visualize the crack pattern, we plotted the fully opened cohesive elements and cohesive elements that are partially opened in Figure 4.32. The difference in crack patterns between the fine and coarse far-field meshes are evident here. Cohesive elements are considered fully open if the opening in the tangential direction at one of the facet's integration points is greater than the final normal or tangential opening, respectively. The percentage of opening for those elements that are not past the maximum final opening is also plotted in Figure 4.32. The percentage of damage is simple the ratio of opening of the element to the final opening. This quantity could be thought of as a damage parameter; aggregating the damage of all cohesive elements could indicate the damage of the entire specimen. Clearly, many more elements are inserted than are completely separated, these inserted elements are in the process of dissipating energy and represent micro-branching that limits the crack speed.

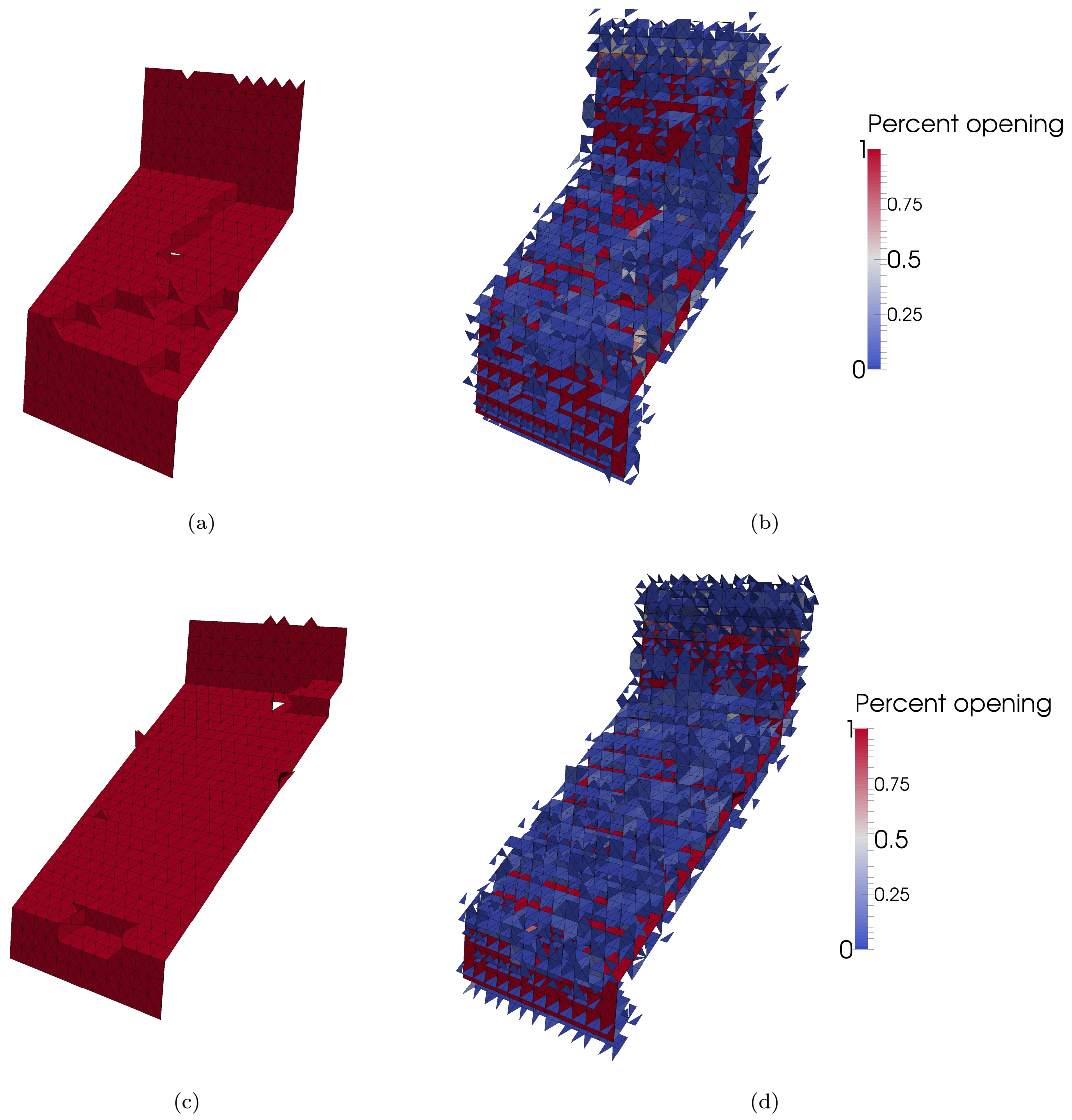


Figure 4.32: Damage of cohesive elements for the TPB specimen (a) fully open elements and (c) all cohesive elements of coarse initial mesh and (c) fully open elements and (d) all cohesive elements of fine initial mesh

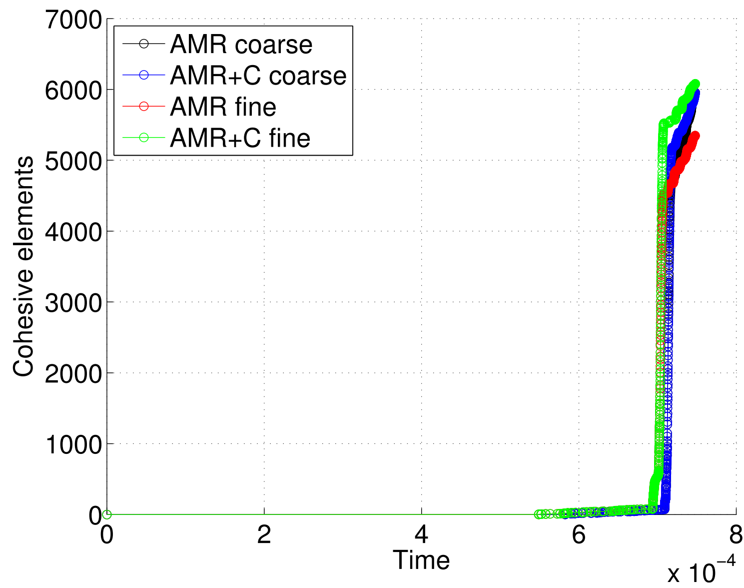


Figure 4.33: Insertion of cohesive elements versus time for the AMR and AMR+C of the TPB specimen with the coarse far field mesh and fine far field mesh

In general, it takes less time on the fine far-field mesh for the stress to build to a sufficient level to initiate cracking. The insertion of cohesive elements versus time is shown in Figure 4.33 for all cases. Notice that the overall behavior is the same for all cases; the majority of the simulation is spent building up stress at the notch tip, then once fracture initiates, the crack propagates relatively quickly.

Chapter 5

Massively parallel adaptive mesh refinement and coarsening for dynamic fracture simulations

We aim to develop the computational tools to enable large scale simulation and in this chapter, we do so through processing on GPUs, which can give massive speedup over even parallel CPU implementations. In the previous chapters of this dissertation, the failure and fracture simulations were performed on CPU systems. While we have access to large amounts of memory and hard disk space on a CPU system, we are ultimately limited by the computational speed of the processor. In the large deformation simulations of Chapter 2 we utilize perform some of the computations in parallel, but only have access to a limited number of processors, so the speedup is not great. For the case of the fracture simulations of Chapters 3 and 4 we have only a single a processor. Now, we focus on porting the adaptive simulations we have discussed in the previous two chapters to the massively parallel Graphical Processing Unit (GPU). Since computational programming on the GPU poses several challenges that are not present on a CPU system, this work begins with 2D adaptive simulations and saves the extension to 3D adaptive systems for a future investigation (see Section 6.2.6 for further discussion).

While the GPU can perform operations very quickly (e.g. floating point operations such as those associated with dynamic fracture simulation), the size of the system is extremely limited and does not immediately lend itself to large systems. The fracture problems we are investigating here are inherently large, so we develop an adaptive refinement and coarsening scheme (AMR+C) suitable for the GPU architecture to ensure only the most important information is stored during the simulation. The AMR+C will allow for analysis of much larger systems than that of an equivalent uniform mesh as we only use the finest level or refinement where necessary. Performing adaptivity on the GPU is not a straightforward task though, as a new mesh representation data structure, finite element calculation framework, and refinement and coarsening algorithms are necessary.

Before going into the details of adaptive mesh refinement and coarsening for dynamic fracture applications, it is useful to first provide a background of the GPU architecture. In Section 5.1 we discuss the unique features of the GPU that make it a challenging platform on which to develop finite element software, yet allows for massive performance when careful consideration is taken. Next we provide an overview of related work in the archival literature in Section 5.2, much of which provided a foundation and inspiration for the work developed here. Our methodology for conducting adaptive mesh modification for dynamic fracture simulation, including the physical basis for the work and computational implementation, is discussed in great detail in the Section 5.3. Equipped with a computational framework to conduct large scale fracture simulation quickly, we explore some numerical investigations in Section 5.4. We simulate benchmark problems with accepted results in the literature, but investigate features of the GPU implementation that have yet to be explored. Future directions for this work can be found in the final chapter of this dissertation.

5.1 GPU architecture

In order to discuss the GPU architecture, it is useful to compare it to a CPU system. CPUs are a multi-core system, which are optimized for serial programming with sophisticated control logic and access to ample cache memory. However, neither control logic nor access to memory improve the peak calculation speed, and by about 2003 software developers were making more advances than could be supported with the existing hardware. The speed of an individual CPU is ultimately limited by energy consumption and heat-dissipation issues, leading developers moved to many-core environment of the GPU. The many-core approach focuses on execution throughput of parallel applications. The GPU is comprised of a large number of small cores, so computationally intensive parts of a code can be moved to the GPU where it can be divided. Moreover, many-core systems have up to 10 times higher memory bandwidth than multi-core systems, making it possible to move data in and out of its memory much faster. Much of the GPU development has been motivated by the gaming industry in which massive numbers of floating point operations are required. In order to accommodate this demand, developers optimize codes for execution throughput of a massive number of threads; more chip area is given for floating point operations rather than to memory to increase throughput. To summarize, GPUs are well-suited for problems involving a large number of floating point calculations, thus they are not universally faster than CPU systems as certain applications are simply not suited for them [184]. In general the developer must take great care to ensure that the benefits of the GPU outweigh the constraints for a particular application, or the performance can suffer greatly.

The work here is performed on a GPU using NVIDIA's CUDA (Compute Unified Device Architecture) framework. A CUDA capable GPU is organized into an array of highly threaded streaming multiprocessors (SMs). Two or more SMs form a building block, the specific number depends on the architecture of the system, Each SM in a building block is comprised of streaming processors (SPs) that share a set of instructions. Each SP is massively threaded, so it can run thousands of threads per application.

The CUDA programming system consists of the host, which is the CPU, and the device, which is the GPU. The program is organized such that some of the application is executed on the host, while the parallel capable code is executed on the device. The device code is activated through functions, called kernels, which generate a large number of threads to perform the parallel computations. All threads execute the same code, which is an instance of single-program, multiple-data (SPMD) parallel programming style. The threads generated in a particular kernel call are organized as follows: the first level is an array of blocks, where each block is identified by its coordinate. Moving one level lower, each block contains a two-dimensional or three-dimensional array threads, which are identified by their thread ID. All thread blocks have the same number of threads. Each block may contain up to 512 or 1024 threads depending on the GPU compute capability and hardware, and the entire grid may contain thousands or millions of threads. A challenge in GPU programming is to create enough threads to fully utilize the hardware. A schematic of the grid-block-thread organization is shown in Figure 5.1.

Another challenge associated with GPU computing is related to memory access. In addition to the device memory, the host also has memory; communication between the host and device memory is critical. A summary of the different types of GPU memory are:

- *Global memory* is located off-chip and all device threads can read/write from/to it. The amount of space in global memory is considerable, however access to it is relatively slow. The host can transfer data to and from global memory.
- Like global memory, *constant memory* allows read only access to the device threads, but the host has

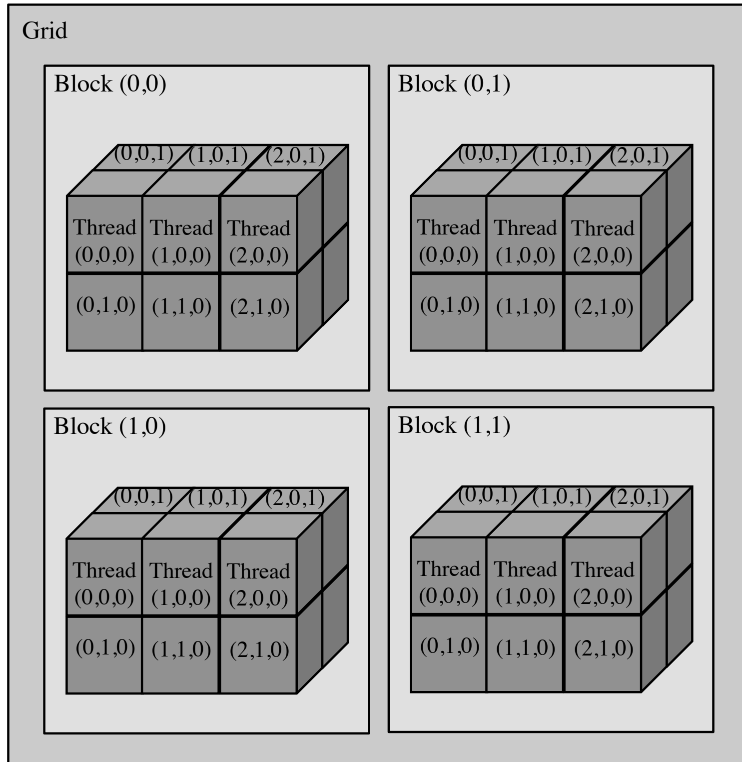


Figure 5.1: Schematic of grid with 2×2 blocks of $3 \times 2 \times 2$ threads each

both read and write access.

- All threads on a block have read and write access to the *shared memory*. Threads from a different block do not have access to shared memory of other blocks. Similarly, the host does not have access to shared memory.
- Each thread has *register memory* which it can read and write to. Other threads do not have access to another thread's register memory. Similarly, the host does not have access to register memory.

Memory is allocated on the host with the standard C `malloc()` command and in a similar way on the device with `cudaMalloc()`. Device space is allocated in global memory, then `cudaMemcpy()` can be used to move the data in one of four ways: host to host, host to device, device to host, device to device.

5.2 Review of related work

In the past decade, researchers have been porting multi-core applications to many-core systems for increased performance. See [185] for an overview of the state of the practice and trends an GPU computing. In the field of finite elements, recent efforts have been focused on linear system assembly, storage, and solution. Researchers in [186] propose an iterative technique to generate space matrices on several GPUs in order to overcome limited storage space. Memory issues are also addressed in [187], where techniques for assembling finite element matrices are explored. Solution of linear systems is also non-trivial, especially on GPUs. To address this, the authors of [188] proposes a massively parallel Cholesky factorization technique for use on GPUs.

Multi-core dynamic fracture simulations have been studied by several authors in the past. For example, researchers [161] investigate cohesive dynamic fracture in a parallel CPU environment ParFUM framework [189]. The cohesive elements require an external activation criteria and do not feature the initial elastic slope present in the intrinsic model, the interface elements are present at all facets before the start of the simulation. Nodes at all facets are duplicated from the beginning, but the traction separation relationship is not activated until the external stress-based criteria is met. In this way, the work is more similar to an intrinsic implementation because the mesh connectivity does not change as the problem evolves.

In a related work, authors in [71] also pre-insert externally active cohesive elements into the domain. They use a discontinuous Galerkin approach, and because the mesh topology does not change, they are able to attain scalability of the parallel implementation.

More recently, fully extrinsic dynamic fracture simulation was achieved in [190]. The work is based on the ParTopS data structure, the parallel version of TopS discussed in the previous chapters, and supports insertion of extrinsic cohesive elements on-the-fly.

Fracture simulation using the many-core GPU environment and adaptive finite element mesh operations are not well documented in the literature. To the best knowledge of the author, adaptive dynamic fracture simulation on GPUs has not been investigated. Adaptivity has been explored in other fields, for example cartesian meshes are generated on the GPU for computational fluid dynamics applications resulting in speedup of up to 36 for large meshes [191]. Dynamic fracture with the extrinsic cohesive zone model on a uniform mesh is investigated and implemented in [192], and serves as the motivation and foundation on which the proposed adaptive GPU fracture is based.

5.3 Adaptive mesh modification on Graphical Processing Units

Mesh adaptivity is important in the context of finite element applications, as it enables larger simulations to be performed in less computational time. Of course, adaptivity has been widely utilized in the context of material fracture and failure modeling, which has been demonstrated throughout the course of this thesis. However, mesh adaptivity and hierarchical schemes are also utilized in other fields such as modeling electronic chip packages [193], large-eddy simulations [194], and astrophysical thermonuclear flash [195], just to name a few. We continue the focus of this work on dynamic fracture simulation and develop the data structure and algorithm for adaptive modification of the finite element mesh, namely mesh refinement and coarsening. First, we present the data structure followed by the methodology for computing finite element calculations on the GPU. The next two subsections detail the algorithms for adaptive insertion of cohesive elements and adaptive mesh refinement and coarsening, respectively.

5.3.1 Data structure for 4k adaptive finite element mesh representation

An efficient data structure is critical in adaptive fracture applications. From a data representation perspective, the implications of inserting a cohesive element or refining or coarsening a region of the mesh involves dynamically changing the size of the node/element representation and updating adjacency relations. If this is not done in an efficient way with respect to computational processing time, then the cost of solving even a small problem could be dominated by upkeep of the mesh representation instead of on the structural mechanics computations [196–198].

In parallel computations, especially on the GPU, it is common that a data structure is tailored very specifically for the application of interest [199, 200]. Several data structures have been devised including

those for dynamic fracture with adaptive insertion of cohesive elements and adaptive topological operators on a serial CPU platform [162], dynamic fracture with adaptive insertion of cohesive elements on a parallel CPU platform [190], and dynamic fracture with adaptive insertion of cohesive elements on a massively parallel GPU platform [192]. However, none of these previous approaches are appropriate for the present work of adaptivity on a GPU platform for a variety of reasons. First, data structures for serial platforms are not equipped to handle issues of concurrency which is critical in parallel simulations. Secondly, the CPU data structures (for serial or parallel platforms) do not have nearly the space restrictions as that of a GPU; we need to store far fewer entities explicitly on the GPU and instead derive them each time they are accessed. Finally, problems in which the bulk mesh remains constant throughout the simulation does not pose the challenges of constant insertion and deletion of variables in the data structure, as we have in the adaptive simulations proposed here.

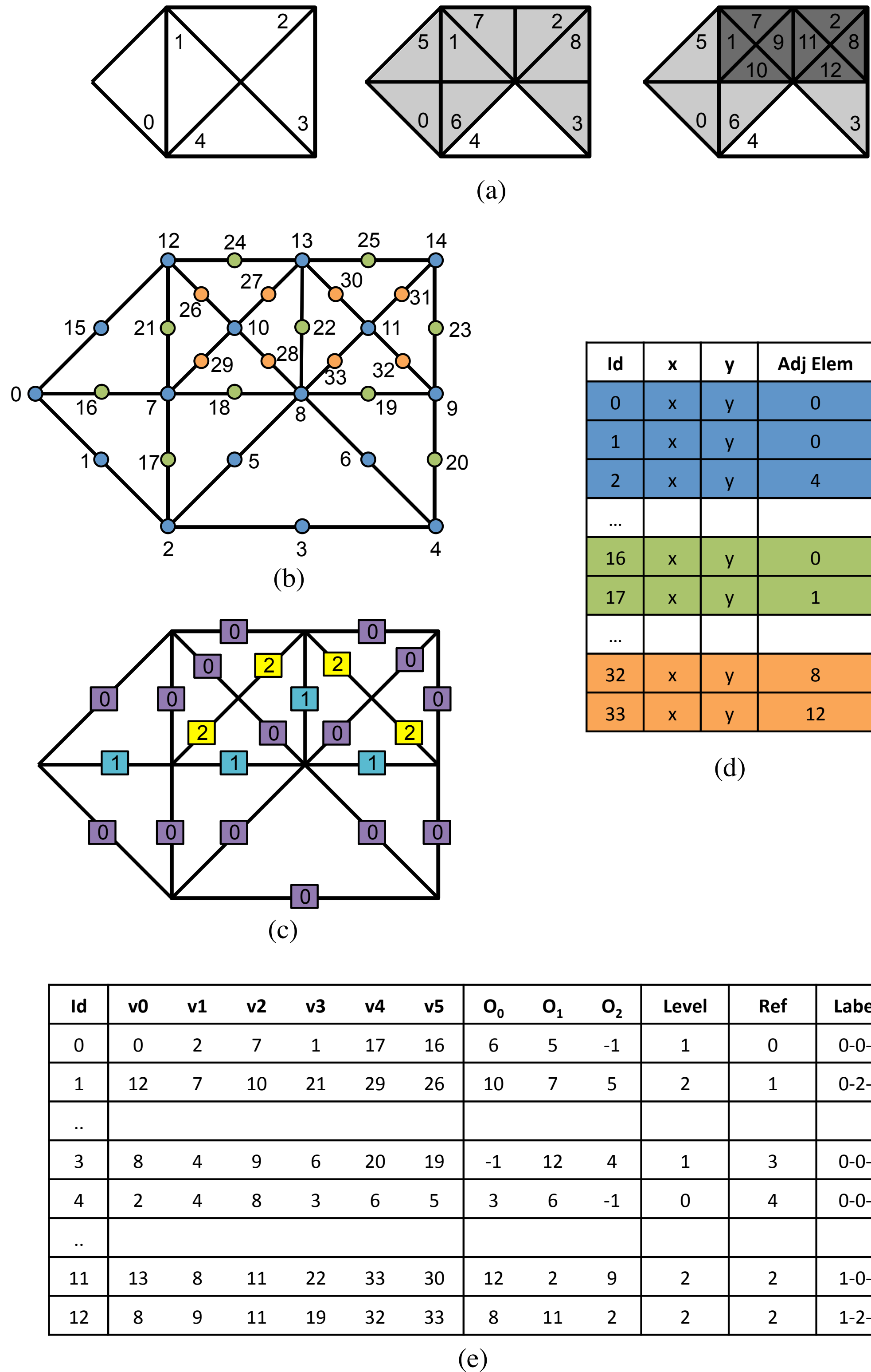
Therefore, given the requirements of the adaptive simulation and limitations imposed by the GPU architecture, we propose a simple and inexpensive data structure for representation of an evolving 4k structured finite element mesh. The data structure consists of node and element tables with some basic adjacencies and information necessary for the hierarchical refinement and coarsening scheme. A schematic of a sample mesh and corresponding data structure representations is shown in Figure 5.2; we will refer back to this figure several times in the next few section to aid discussion.

The sample coarse mesh is partially refined in three steps, as shown in Figure 5.2(a) and the corresponding nodal number of the final mesh is shown in Figure 5.2(b). All of the data necessary to store this adaptive mesh is contained in the node table (Figure 5.2(d)) and element table (5.2(e)). Notice that each row of the node table contains x-y coordinate and an adjacent bulk element of the node. The bulk element associated with each node is used for traversal of adjacent entities to gather adjacency not explicitly stored on an as needed basis. For bulk elements, each row of the element table contains the nodal connectivity for the quadratic triangular elements and the ids of elements opposite each vertex, which are used in a similar fashion as the adjacent element in the nodal table, e.g. to construct adjacency information on the fly. Additional information necessary for mesh refinement and coarsening is also stored in the element table. We showed the nodal and element IDs as entities of the node and element tables in the schematic, however this is only for demonstrative purposes, the IDs are implied by the index of the array. Cohesive elements are also stored in the element table (although none are shown in Figure 5.2). Their rows contain only the list of six nodes that define the element, and adjacency information. We do not refine or coarsen cohesive elements, so the additional information is not necessary.

As bulk elements are subdivided, they are assigned a new level of refinement; elements at level 0 are shown in white, elements at level 1 are shown in light grey, and elements at level 2 are shown in dark grey. These levels are stored in the element table as shown in Figure 5.2(e).

In addition to the element level information, we also adopt a facet labeling technique in which new facets resulting from mesh refinement are assigned a label with a number equal to one more than its adjacent facets. For example, when the simulation begins, the facets of each element are labeled 0, then the new facets resulting from one level of refinement are labeled 1, next new facets resulting from a second level of refinement are labeled 2, and so on. The facet labels are shown on the schematic in Figure 5.2(c) and as a column of the element table in Figure 5.2(e).

Finally, we must keep a history of the coarse element from which finer elements came so that we can revert back to coarse elements when needed. To accomplish this, we store the ID of the coarse element (parent) from which two elements (children) emerge during refinement from level n to level $n + 1$; the reference



Id	x	y	Adj Elem
0	x	y	0
1	x	y	0
2	x	y	4
…			
16	x	y	0
17	x	y	1
…			
32	x	y	8
33	x	y	12

Id	v0	v1	v2	v3	v4	v5	O_0	O_1	O_2	Level	Ref	Labels
0	0	2	7	1	17	16	6	5	-1	1	0	0-0-1
1	12	7	10	21	29	26	10	7	5	2	1	0-2-0
..												
3	8	4	9	6	20	19	-1	12	4	1	3	0-0-1
4	2	4	8	3	6	5	3	6	-1	0	4	0-0-0
..												
11	13	8	11	22	33	30	12	2	9	2	2	1-0-2
12	8	9	11	19	32	33	8	11	2	2	2	1-2-0

Figure 5.2: Schematic of GPU data structure for adaptive 4k mesh (a) progression of mesh refinement and element labeling, (b) node numbering on refined mesh, (c) facets labels indicating order of refinement on refined mesh, (d) node table showing node ids, coordinates, and adjacent element, (e) element table showing element id, nodal connectivity, adjacent elements, reference level, level of refinement, and facet labels

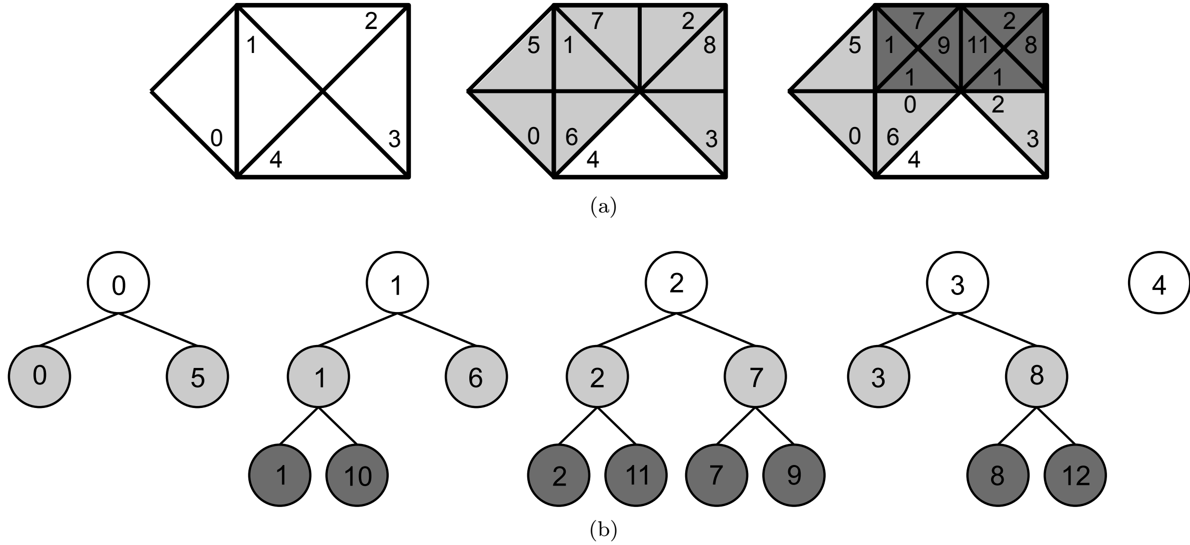


Figure 5.3: (a) Refinement of elements 1, 2 and 3 from level 0 (white) to level 2 (dark grey), (b) Binary tree representation of refinement

element is stored in the element table shown in Figure 5.2(e). To move back from a refined element back up to its parent element we also the history of mesh refinement in a binary tree, shown in Figure 5.3. When an element is split, the resulting elements are children of the original element in the tree. Coarsening involves moving back up the tree.

When coarsening the mesh, elements and nodes are removed from the tables, which results in “holes” in the data structure. Rather than collapsing the data structure by renumbering the mesh entities (we explored this approach but concluded that it was too computationally expensive for the present application), we opt to fill the holes the next time a node or element is inserted into the mesh. Thus we need to keep track of unused node and element IDs, which we do through a node and element stack. When new nodes/elements are added to the mesh, we first insert them at the indices stored in the respective stack. Then once the stack is empty we add nodes/elements by creating new entities in the node/element tables, respectively. We avoid coarsening of cohesive elements, therefore there is no need to store additional information for adaptivity.

5.3.2 Node and element calculations on GPU

In parallel finite element simulations, it is critical to avoid concurrently writing to the same location in memory. For example, node quantities (e.g. displacements) are composed of contributions from adjacent elements. Writing conflicts will arise if multiple elements are updating a node at the same time. Typically, this issue is handled with graph coloring schemes, such as that proposed in [201], whereby the color groups are visited in a serial fashion, and one thread is launched per element in that color group. This is usually done once at the start and absorbed in the overhead cost as it is a relatively expensive operation for arbitrary meshes. In the adaptive simulation, however, the number of bulk elements and their connectivity will change. Therefore the coloring algorithm would need to be executed every time a bulk element was removed or inserted, which would be computationally inefficient even on the GPU.

Our solution for the adaptive simulation is to sweep the nodes (one thread per node) and gather information from its adjacent elements, rather than sweeping elements (one thread per element) and updating its

adjacent nodes. The data structure discussed in Section 5.3.1 allows for quick access the elements adjacent to the nodes. Using the node-based calculations, elements will be visited concurrently, as nodes within a close vicinity will share adjacent elements. However, since data will only be read from the element entities, the issue of concurrent writing is not present. The efficiency of this approach is similar to that of the element sweep approach. When mesh refinement and coarsening are enabled, this approach leads to some variation in final crack patterns and results from one simulation to another. While this variation is not incorrect it warrants some investigation, which is conducted via numerical experiment in Section 5.4.

5.3.3 Adaptive insertion of cohesive elements

The extrinsic cohesive model employed in this work requires an external criteria to activate (i.e. insert) the traction-separation relation associated with the cohesive element. A number of approaches have been utilized to activate the elements including strength/stress, strain, velocity, numerical instability, etc. We will assume that elements have been activated for the following discussion. When cohesive elements are inserted into the mesh, the nodes along the insertion facet are duplicated. The cohesive element is defined by the original nodes along the facet (three in this case of quadratic triangular elements) plus the additional nodes resulting from duplication.

As mentioned in Section 5.3.2, we do not utilize a graph coloring scheme in this work. One of the implications of our node-based approach is that we cannot use previous strategies, such as those adopted in [190, 192], to insert cohesive elements at bulk element facets. Instead we utilize a two-step node sweep strategy, which is analogous to the update scheme discussed in Section 5.3.2.

In the first step, we launch one thread per cohesive element to be inserted. The cohesive elements are added to the element table but the nodes are not yet duplicated or added to the node table. Once all of the elements are inserted we begin the second step.

One thread per node belonging to a cohesive element is launched to determine if it needs to be duplicated. All nodes defining the cohesive element are duplicated except those at the end of a series of cohesive elements [163], hence non-duplicated nodes are used to define the crack tip. This algorithm does not cause issues of concurrent writing because each thread is responsible for duplicating its own node.

5.3.4 Adaptive mesh refinement and coarsening

One of the main contributions of this work is the development and implementation of the 4k refinement and coarsening scheme on the GPU. The criteria for which is similar to that of [158] and is brevity reviewed here before describing the GPU implementation.

Multiple crack tips may emerge as the quasi-brittle fracture simulation evolves in time (recall that the definition of a crack tip is the non-duplicated node of a cohesive element). These crack tips are necessary to perform adaptive mesh refinement, as we use the *a priori* assumption that regions around crack tips (i.e. high gradient of the displacement field) must be the finest in the simulation. Given a crack tip, elements that fall within a user-specified radius are refined according to the hierarchical 4k scheme to a user-defined level. To avoid complicated transfer of internal state variables associated with the nonlinear cohesive elements, this refinement strategy prohibits cohesive elements from being refined or coarsened. Thus, cohesive elements may only be inserted at elements that are already refined to the highest level. This assumption is generally acceptable, as we expect cracks to initiate from areas that are fully refined, e.g. initial defects, notch, or crack tip.

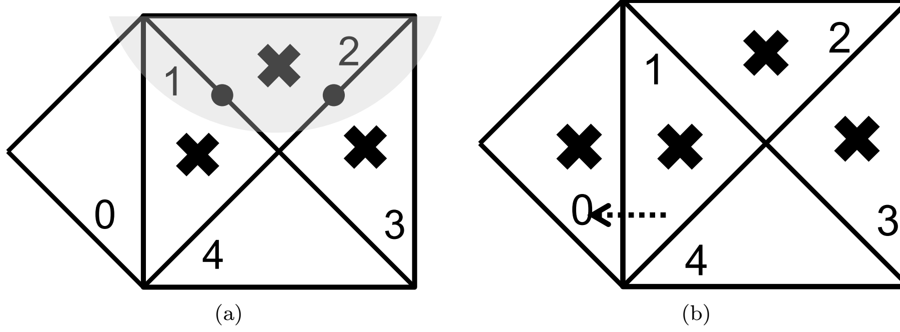


Figure 5.4: Marked opposite elements (a) elements with at least one node inside the refinement region are marked, (b) elements adjacent a marked element's hypotenuse are also marked

For adaptive mesh coarsening, we can deduce that in regions far away from the crack tip, a coarser mesh is sufficient. However, unlike refinement, the coarsening criteria is not based on an element's geometric position relative to the crack tip. Rather, it is based on convergence of the norm of the strain of the coarsened mesh to that of the refined mesh. The error between the norm of the strain in a patch of the refined 4k element is compared to the norm of the strain in the same patch but with coarse elements. If the error is less than a certain threshold, 2% in this study, then the patch is refined. Since the finite element space is becoming less rich, energy conservation is not expected, however the loss is minimal and justified by the gain in memory and processing time.

The algorithm to refine bulk elements in a certain region of a 4k mesh is a multi-step procedure described as follows and corresponds to Algorithm 2. Before proceeding, we will clarify the notation we use in Algorithm 2; `<<< x >>>` indicates is kernel call is being made where x indicates the number of threads launched. First groups of cells (4k patches) are visited to determine if they overlap a current refinement region, if so, the cell is marked (line 2 in Algorithm 2). Next, we move to the element and start with a kernel call where to visit each bulk element by launching one thread per element. If the midpoint of at least one facet of the element lies inside the refinement region, then the element is marked for refinement (lines 4-5 in Algorithm 2). In the next kernel call, for each marked element we also mark the element adjacent to the hypotenuse of the originally marked element (lines 7-8 in Algorithm 2). This is illustrated in Figure 5.4, where the facets between elements 2 and 1 and elements 2 and 3 have at least one node that falls within the radius of refinement shown as the light grey semi-circle, so all three elements are marked. Element 0 is adjacent to the hypotenuse of marked element 1, so it is also marked.

In the next kernel call we launch one thread per marked element and split it according to the 4k hierarchical refinement strategy, i.e. split the element along its longest edge [202]. It is useful to note that the first two nodes in a row of the element table are the corner nodes that define the hypotenuse of the element (see Figure 5.2(e)), thus the longest facet of an element is directly accessible and does not require additional calculations. New nodes/elements are created in this step by either adding them to the node/element tables or by reusing node/element IDs from their respective stacks (line 10 in Algorithm 2). The last kernel updates adjacency of the newly added elements in the element table (line 11 in Algorithm 2). This procedure is continued until all elements inside the refinement region reach the level prescribed by the user.

The refinement scheme is demonstrated from a topological perspective in Figure 5.5. The domain contains an initial notch which is refined. In the next step cohesive elements are inserted (notice that they are also inserted along facets of fully refined elements) and the crack tip nodes are updated. The new regions of

Algorithm 2 Kernel based algorithm to perform adaptive mesh refinement

```
1  RefineCUDA ()
2    MarkRefinedRegionCells <<<numCells>>>
3    do {
4      MarkElements<<<numElements>>>
5      numMarkedElements = ScanMarkedElements<<<numElements>>>
6      do {
7        MarkNeighbors<<< numMarkedElements >>>
8        numMarkedNeighbors = ScanMarkedNeighbors<<<numElements>>>
9      } while numMarkedNeighbors != 0
10     SplitFacets<<<totalNumMarkedElements>>>
11     Update adjacency<<< totalNumMarkedElements>>>
12   } while there are marked elements
13 end RefineCUDA
```

refinement are shown with orange circles around the new crack tips. The elements that fall within the refinement radius any of the crack tips is marked for refinement. Elements adjacent to the hypotenuse of a marked element are also marked. Finally marked elements are refined until the desired level of refinement is reached and elements just outside the refinement region are refined to an intermediate level to ensure compatibility with the coarser elements.

The parallel coarsening algorithm is essentially the reverse of the refinement, however the implementation on the GPU must be done in such as way to ensure concurrency is avoided. As with the refinement algorithm, we first mark cells that are outside current refinement regions (line 2 in Algorithm 3). Then, one thread per bulk element is launched. The bulk elements are marked if the mid points of all of its facets are outside of an existing refinement region (lines 4-5 in Algorithm 3). After bulk elements are marked, then the nodes are visited through a kernel call launching one thread per node. The node is marked for coarsening if (1) four of its adjacent bulk elements were marked as being outside a refinement region, and (2) two of the facets emanating from it are labeled with values greater than than that of any other facets on adjacent elements (lines 9-10 in Algorithm 3).

Once a node is marked for coarsening, we check that it meets the coarsening criteria. If so, then a kernel call is used to update the adjacent elements' reference element (line 11 in Algorithm 3). Finally the element is coarsened, which involves updating the adjacent element to the node in the node table, the nodes defining the adjacent elements, the elements opposite to the corner nodes of the adjacent elements in the element table, the level of refinement of the adjacent elements (line 12 in Algorithm 3).

5.4 Numerical investigations

The adaptive mesh refinement and coarsening scheme implemented here is applicable for many types of fracture problems, however the benefits of the approach are most realized for problems dominated by few cracks. Consider the contrary example of pervasive fracture problems [172]. All or most of domain needs high levels of mesh refinement to capture the fracture behavior, thus an adaptive refinement scheme would not have any effect. The following investigations are intended to examine implications of the GPU implementation on the physics of the problem, push the limits of the GPU to determine the maximum problem size that can be simulated, and to explore the response of systems through parametric studies in an efficient manner.

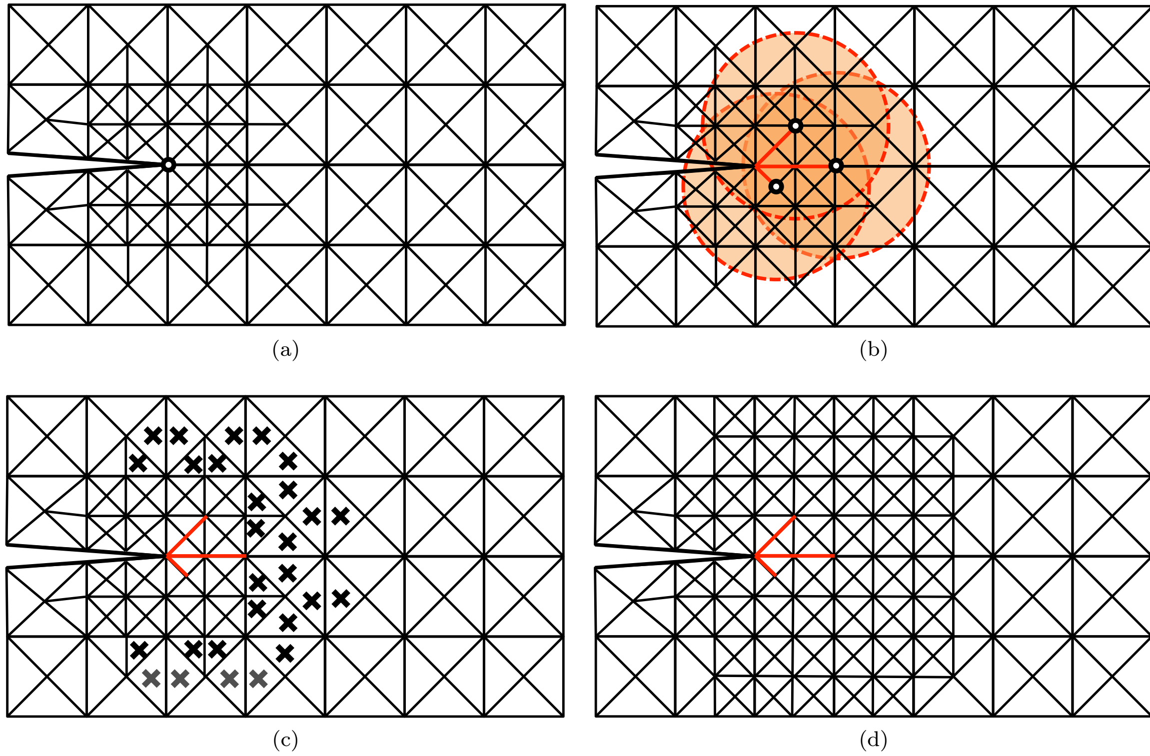


Figure 5.5: 4k refinement scheme (a) Mesh is initially refined around the notch tip. (b) Cohesive elements are inserted along facets of fully refined elements, new crack tips are identified and new refinement regions associated with each crack tip are created. Elements to be refined for all crack tips are collected simultaneously, as opposed to one crack tip at a time (c) Elements within the refinement region are marked (black 'x') and elements adjacent to the hypotenuse of a marked element are marked (grey 'x') (d) Marked elements are refined to the full level and transition region refined to ensure element compatibility

Algorithm 3 Kernel based algorithm to perform adaptive mesh coarsening

```
1  CoarsenCUDA ()
2    MarkCoarsenRegionCells <<<numCells>>>
3    do {
4      MarkElements <<<numElements>>>
5      numMarkedElements = ScanMarkedElements <<<numElements>>>
6      if numMarkedElements == 0 then {
7        break
8      }
9      MarkNodes <<<numNodes>>>
10     numMarkedNodes = ScanMarkedNodes <<<numNodes>>>
11     UpdateReferenceTable <<<numMarkedNodes>>>
12     Coarsen <<<numMarkedNodes>>>
13   } while there are marked elements
14 end CoarsenCUDA
```

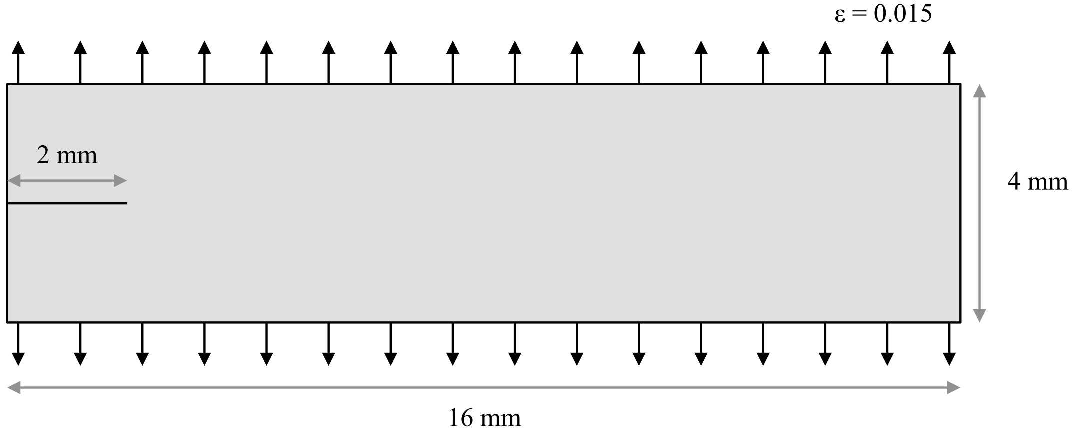


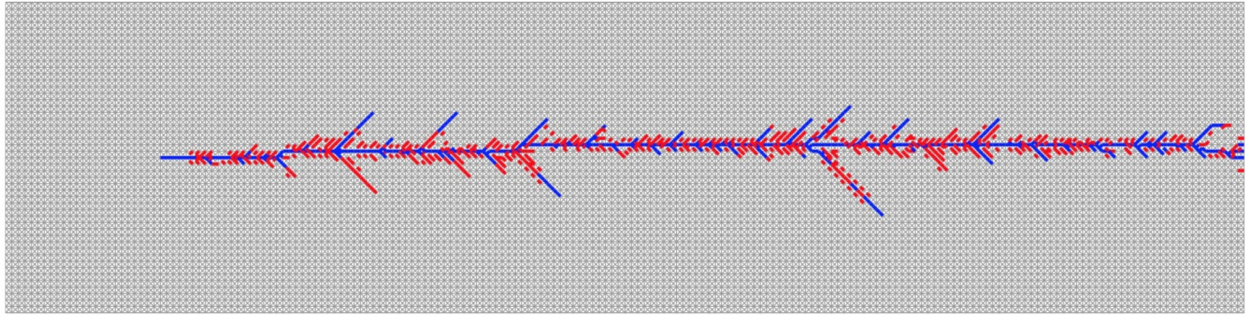
Figure 5.6: Reduced scale micro-branching problem geometry, loading conditions, and material properties

5.4.1 Reduced-scale micro-branching specimen

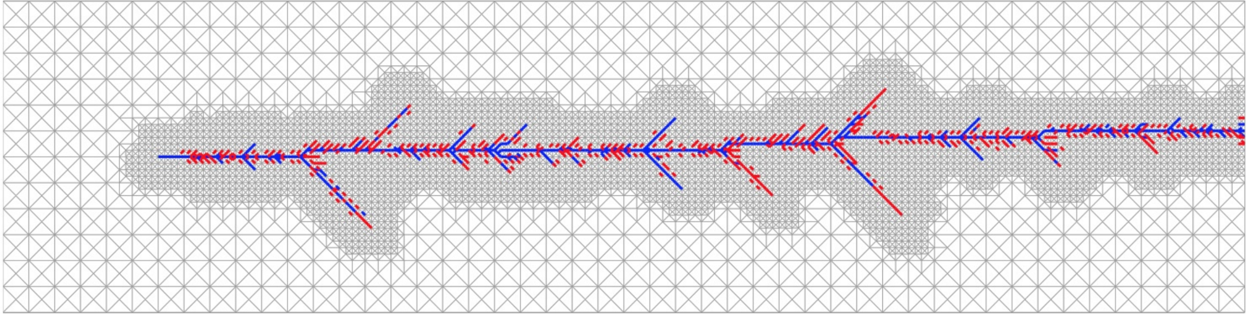
We verify the implementation of the adaptive scheme on the GPU through a series of numerical investigations on a well-known micro-branching problem. This model problem is inspired by the experimental work of [203] and has been simulated by many authors [88, 89, 153, 190]. Similar to the previous investigations, we utilize a reduced scale model for direct comparison purposes. Later we will address the issue of the full scale model. The problem features few major cracks, which makes it a good candidate for the adaptive scheme, and several minor cracks that results in a complex fracture pattern. The simple geometry and loading conditions of the reduced scale model are shown in Figure 5.6. For the reduced scale model, we use the material parameters suggested in [153]. Due to the reduction of the model size and known issues related to representing an experimental system on a numerical model, we adopt the following: Young's Modulus of 3.24e9 Pa, density of 1190 $^{kg}/_{m^3}$, and a Poisson ratio of 0.3 for the bulk elements and a fracture energy of 352.3 $^{N}/_{m}$ and cohesive strength of 129.6e6 Pa for the cohesive elements. The shape of the softening curve is linear, as given by the PPR shape parameter of 2 in each opening direction. Unloading is assumed to occur linearly back to the origin, i.e. permanent deformation is not sustained. To prevent interpenetration of materials, a penalty stiffness is applied if cohesive tractions become negative.

First, it is useful to compare the results using AMR and AMR+C to that of an equivalent uniformly refined mesh. For the reduced scale model, the uniform mesh is comprised of 192×48 4k patches, or 36,864 T6 elements. The AMR and AMR+C enabled meshes are initially discretized into 48×12 4k patches, or 2,304 T6 elements, then adaptively refined to a level 4 in the region of the crack tips. Elements are removed in the AMR+C case in regions aways from the crack tips when the root mean error in the strain on a patch of elements falls below the user defined threshold of 0.01.

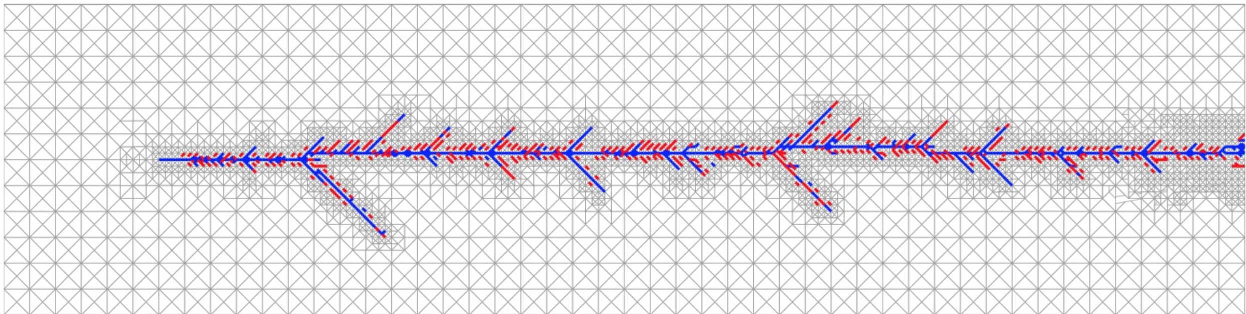
The final crack patterns for each case are shown in Figure 5.7. The finite element meshes are visible and various levels of refinement are clear in the AMR and AMR+C cases. Cohesive elements that are open greater than a certain threshold of the critical opening distance in either the normal or tangential direction are are plotted in blue, and elements that are present in the model but not open greater than the threshold in either direction are plotted in red. The threshold by which a cohesive element is considered open is important when quantifying the fracture pattern. For visualization purposes, we show the fracture pattern with the relatively low threshold of 10%, but in the quantifications reported later in this section we also examine a higher threshold.



(a)



(b)



(c)

Figure 5.7: Final crack pattern for the reduced scale micro-branching problem for (a) uniform mesh (b) AMR enabled mesh (c) AMR+C enabled mesh. Cohesive elements opened greater than 10% of the normal or tangential critical opening distance are shown in blue, other cohesive elements are shown in red

Table 5.1: Comparison of final quantities between Uniform, AMR and AMR+C simulations

Tol	**Mesh type**	**Elements**	**Nodes**	**Crack tip velocity**	**Total crack length**	**Num Branches**	**Avg. Brach Length**
0.1	Uniform	36,864	76,268	777.5 m/s	0.034 m	2	2.2e-4
0.1	AMR*	13,277	28,506	754.3 m/s	0.036 m	1	5.2e-4
0.1	AMR+C*	8,303	18,296	755.5 m/s	0.039 m	2	5.1e-4
0.75	Uniform	36,864	76,268	777.5 m/s	0.019 m	14	4.6e-4
0.75	AMR*	13,277	28,506	754.3 m/s	0.021 m	19	4.1e-4 m
0.75	AMR+C*	8,303	18,296	755.5 m/s	0.021 m	20	4.7e-4 m

* The AMR and AMR+C quantities are averaged over 20 simulations

Table 5.1 shows the final number of elements and nodes (after adaptivity and insertion of cohesive elements) and quantitative differences between the simulations, namely the crack tip velocity, total crack length, and number of branches off the main crack. The crack tip velocity is computed by performing a linear regression of the crack tip versus time, where the crack tip is defined at the right-most non duplicated node of a cohesive element. The crack tip velocity is quite stable throughout the simulation, so the linear regression agrees well with the raw data. Notice that the crack tip velocity is the same for both tolerances, because by our definition the crack tip for the purposes of the velocity calculation is independent of the amount of element opening. The total crack length is total distance covered by all of the cohesive elements open greater than a certain fraction of the critical normal or tangential opening length (denoted Tol in Table 5.1). We see good quantitative agreement between the uniform, AMR and AMR+C cases in terms of the crack tip velocity and total crack length.

The number and length of branches was post-processed using a simple algorithm performed on the final fracture pattern. Starting from the notch tip node, the main branch is detected by traversing cohesive elements using the adjacency information stored in the data structure. The main branch consists of the path of full open elements that reach the right end of the specimen. Once the main crack is detected, the secondary branches are found by again traversing the main crack. At every point where the crack branches, the path is followed using adjacent information until it terminates. Primary branches are those with the longest length emanating from the main branch. A shorter branch emanating from a primary branch is denoted as a secondary branch, see Figure 5.8. This algorithm excludes cohesive elements that are not connected to the main crack. We chose this approach so that the process of counting branches would be controlled and consistent between specimens. The procedure of quantifying number and length of branches is too subjective to be evaluated by a visual inspection. There is quite a difference in the number and average length of branches and between the uniform, AMR and AMR+C cases. We report this information, because when visualizing a fracture pattern, one often focuses on the number and length of branches, however this data can be misleading, because it is not an accurate representation of the crack velocity or the total crack length, which includes the main crack and the many kinks it may have, as illustrated in Figure 5.8. Instead, we choose total crack length as the important quantity on which to compare the cases because it is directly related to the total energy released during the fracture process, and this is quantity that should remain similar between different numerical representations of the same process.

The quantities shown in Table 5.1 for the AMR and AMR+C cases are average over 20 simulations. This is because the massively parallel nature of the adaptivity in GPU implementation introduces some variation into the fracture simulation. We will explain this through a simple refinement example. In Figure 5.9, the patch of four, grey elements is refined to the patch of eight, colored elements. The order in which these

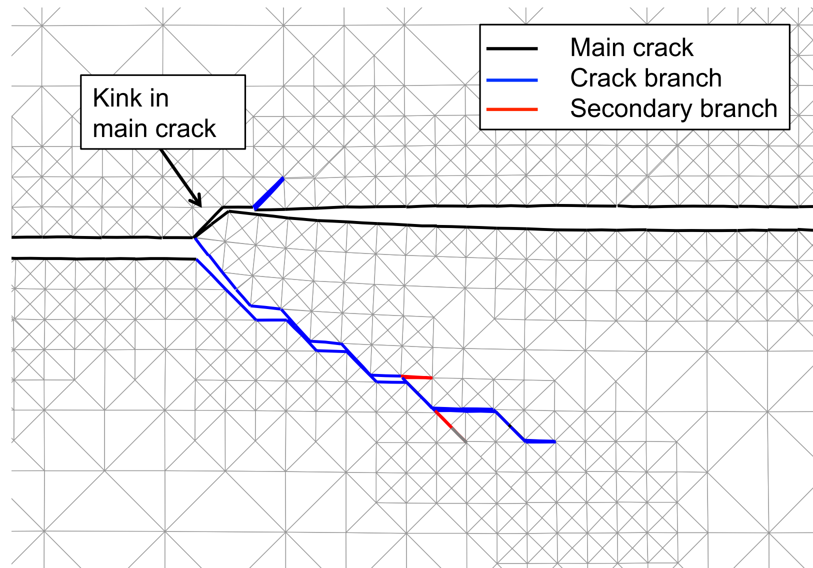


Figure 5.8: Details of crack branching including kink in the main crack, crack branches, and secondary branches

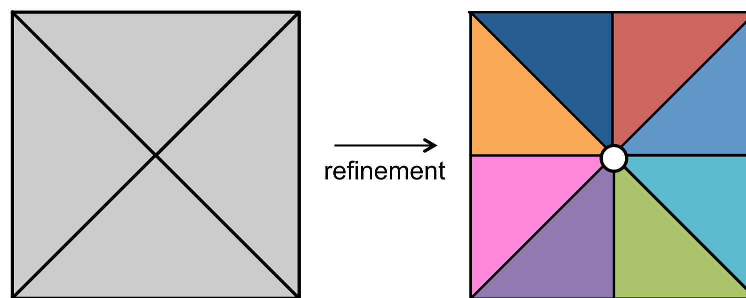


Figure 5.9: A coarse mesh patch of four grey elements is refined to a patch of 8 colored elements; the order in which the color elements contributions are added to the node will vary from one simulation to the the next

elements are inserted is random, meaning that in one simulation the green element may be inserted first, and in a second simulation the red element may be first, and so on. Recall that avoid graph coloring and concurrency issues, we traverse nodes and gather necessary data from elements as opposed to traversing elements and writing to nodes. This traversal is accomplished by first visiting the adjacent element to the white node, then by visiting the element opposite its corner node, and from the next element we visit the element opposite its corner node, and so on until we reach the first visited element. So, since the order in which the elements are inserted is not constant from one simulation to the next, the first element visited and subsequent order of traversal is also not constant. Since we only have a certain level of accuracy in floating point operations, we cannot, in general, guarantee $A + B \neq B + A$. So when computing quantities on the white node, we may pull date in one simulation from the green element, then dark blue, then orange, then cyan, then light blue, then pink, then purple, then red; and in another simulation we may pull from purple, then red, then light blue, then pink, then orange, then cyan, then green, then dark blue, etc. These variations accumulate over all of the computations, nodes, time steps, etc. and the result is a a variation in final fracture patterns.

Table 5.2: Variation in crack tip velocity, energy released, and occurrence of branching for 20 simulations of each the AMR and AMR+C enabled meshes

		Tol = 0.1		**Tol = 0.75**	
		AMR	**AMR+C**	**AMR**	**AMR+C**
Total crack length	Mean	0.036 m	0.039 m	0.021 m	0.021 m
	Standard deviation	8.8e-4 m	6.8e-4 m	9.2e-4 m	9.4e-4 m
Number of branches	Mean	17	19	1	2
	Standard deviation	3	4	1	1
Average branch length	Mean	5.2e-4	5.1e-4	4.1e-4 m	4.7e-4 m
	Standard deviation	4.2e-4	4.6e-4	4.2e-4	6.1e-4

It should be noted that we also examined an implementation in which the order of element/nodal computations is prescribed and the same from one simulation to another and verified that the results are identical. This does not imply that the implementation with no variation is correct and the one with variation is incorrect. The same randomness is present in the consistent implementation and if we chose to access the elements in a different order, we would have a similar effect as the implementation with variation. We chose to pursue the implementation that introduces randomness because it is much more computationally efficient.

Using the reduced scale micro-branching problem, we investigate the impact that the randomness has on the final result. We performed the simulation 20 times on each of an AMR and AMR+C enabled mesh, then quantified the variation in fracture patterns in Table 5.2.

As before, we notice a large difference in the number of crack branches especially for the low crack tolerance, which emphasizes the point that number of branches is not an ideal measure to by which to compare fracture patterns resulting from the same process, e.g. same geometry, material properties, and loading conditions. The variance on the total crack length is quite low, suggesting that the variation caused by the numerical implementation is low. The crack tip velocity also shows low variation amongst the 20 iterations, for the AMR and AMR+C cases the crack tip velocities are $754.3 \pm 9.8\ {}^{m}/_{s}$ and $755.6 \pm 10.1\ {}^{m}/_{s}$, respectively. Additionally, the total energy released during the fracture process is quite comparable, $75.0 \pm 2.6\ {}^{N}/_{m}$ and $77.0 \pm 2.0\ {}^{N}/_{m}$ for the AMR and AMR+C cases, respectively. The total energy released considers all cohesive elements, regardless of their amount of opening, thus this quantity is also independent of the threshold.

We observe some other additional fracture pattern characteristics. The branch spacing is fairly regular among all simulations and the main cracks kinks about 3-6 times during the simulation. Most of the branches are 1-3 elements in length, then the frequency drops significantly, as shown in Figure 5.10. Secondary branches occurred in about half of the adaptive. Thus we concluded that the variation caused by the massively parallel GPU implementation is not significant.

The variation caused by the GPU could alternatively be viewed as a way to induce randomness into the numerical model, which in other similar studies was achieved by perturbing a structured mesh [88] or by using a completely random mesh [89]. The adaptive GPU implementation allows the use of structured mesh with variability that would be expected of a random mesh.

Finally, we compare the computational time of the proposed scheme with other platforms (serial CPU, single GPU) and different types of implementation (adaptive vs. non-adaptive). The serial CPU versions were run on a .3 GHz Intel Core i5 processor and the GPU version implemented here was done on a GeForce GTX TITAN with 2688 CUDA cores and 6Gb memory. To the best of the authors' knowledge there has been no other implementation of an adaptive mesh refinement and coarsening scheme done on a parallel platform. Table 5.3 shows the run times and speedups over the serial implementation without adaptivity.

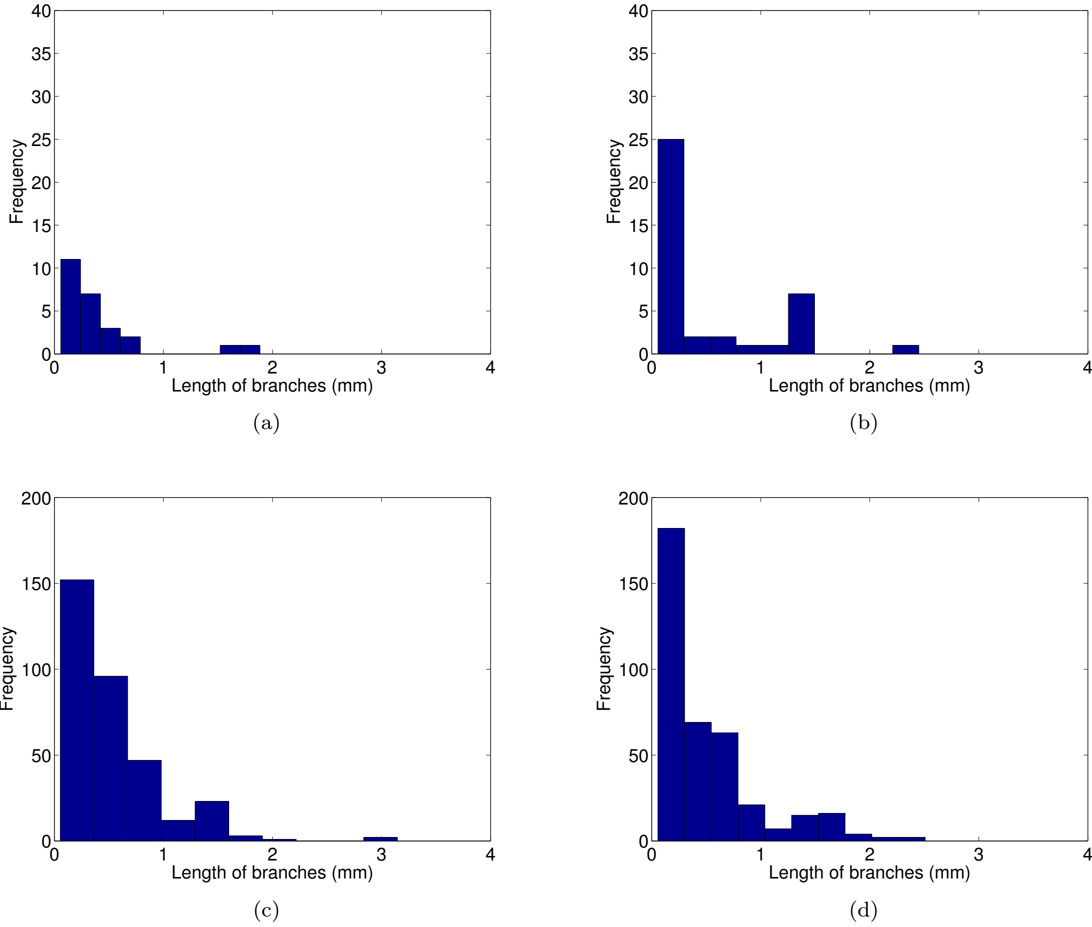


Figure 5.10: Histogram of branch lengths over 20 simulations for the (a) AMR enabled meshes with an open crack tolerance of 75% of critical normal opening, (b) AMR+C enabled meshes with an open crack tolerance of 75% of critical normal opening, (c) AMR enabled meshes with an open crack tolerance of 10% of critical normal opening and (d) AMR+C enabled meshes with an open crack tolerance of 10% of critical normal opening

Table 5.3: Comparison of wall time of the reduced scale micro-branching problem on different platforms (The speed up factor is shown with respect to the no adaptivity case on the serial CPU)

Platform	**Implementation**	**Wall time**	**Speed Up Factor**
Serial CPU	No adaptivity	1,196 sec	–
Serial CPU	AMR	83 sec	14
Serial CPU	AMR+C	57 sec	21
Single GPU	No adaptivity	12 sec	100
Single GPU	AMR	18 sec	66
Single GPU	AMR+C	20 sec	60

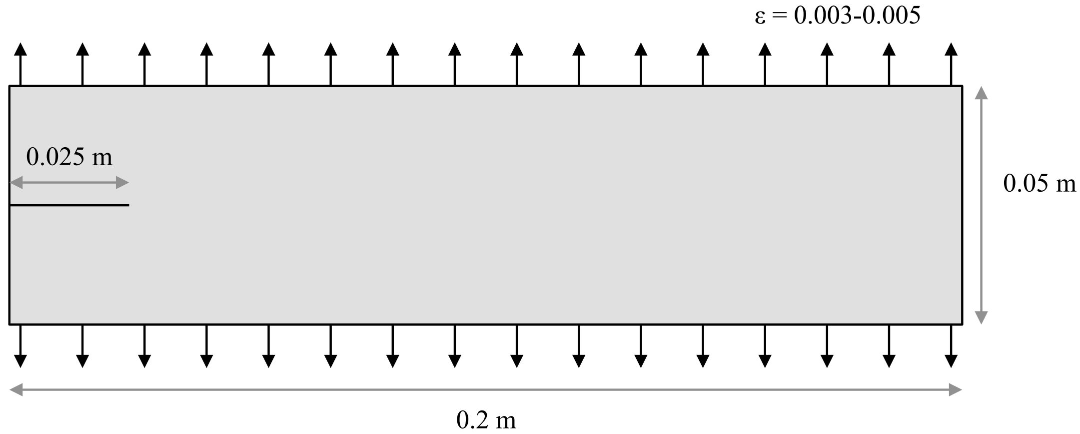


Figure 5.11: Geometry of full scale micro-branching problem

Of course, the GPU is much faster than the serial CPU, thus adaptivity also performs faster on the GPU than the CPU. It is interesting to note that the cases of adaptivity on the GPU actually take longer than the uniform case. This is because for this small problem, the percentage of time spent on updates related to adaptive mesh refinement and coarsening on the GPU is greater than that spent on the finite element calculations. When the problem is larger on the GPU, then we begin to see a difference in wall time between the uniform and adaptive simulations. More important, however, is that the size of the problem is severely limited for the uniform case on the GPU; this limitation is alleviated by adaptivity, which makes large problems feasible because we store much less information than we would on a uniform mesh. So, we may not achieve a large speedup between the uniform and adaptive cases on the GPU, but adaptivity gives the ability to examine problems that we could not be able to simulate otherwise.

5.4.2 Full scale micro-branching specimen

Next, we are interested in comparing the fracture pattern from the reduced scale model with that of the actual experimental set-up proposed in [203]. The full scale problem size has dimensions 50×200 mm (12.5 times larger than the reduced scale case from the previous section), as shown in Figure 5.11. Previous numerical simulations of this work using the inter-element cohesive zone model have only simulated reduced scale problems due to limitations of computation resources and sophisticated algorithms [158, 190, 204]. The adaptivity algorithm implemented on the GPU architecture makes simulation of this full scale problem possible. We should note, that even with the GPU and adaptive mesh refinement, computational times for the following simulations were very high: up to 14 hours for each of the results shown here.

In scaling up the problem, not only does the geometry of the specimen change, but also the applied load and material properties [82]. For the full scale model, the goal was to keep the numerical representation as close to the experiment. Thus specimen dimensions are those of the experiment, and the material properties are those of PMMA, the material used in the experiment. The Young's Modulus is 3.24e9 Pa, the density is 1190 $^{kg}/_{m^3}$, and the Poisson ratio is 0.3 for the bulk elements, while and a fracture energy of 352.3 $^{N}/_{m}$ and cohesive strength of 62.1e6 Pa is used for the cohesive elements. As before, linear softening, linear unloading back to the origin, and a penalty stiffness to prevent interpenetration are utilized. We examined a range of externally applied loads: a low strain of 0.003, mid strain of 0.004, and a high strain of 0.005, which are similar to the loads applied in the experiment.

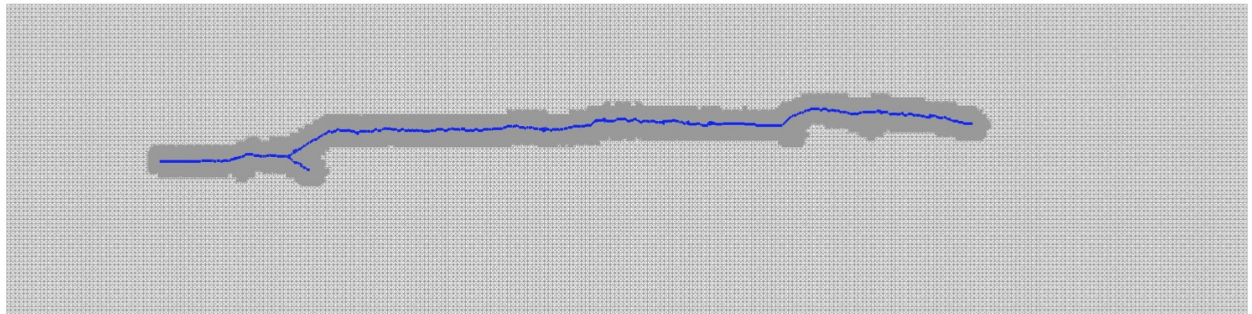
The difference between the full scale and the reduced scale model is the applied load and the cohesive strength. For the reduced scale model, the loading was increased such that the strain energy per unit length felt by the specimen would match that of the experiment. The adjustment of the material properties for the reduced scale model, namely the cohesive strength, is not as straightforward as has been demonstrated by other authors [153, 204]. A cohesive strength that is too large means that fracture never initiates, while a low strength results in the insertion of an excessive number of cohesive elements, which is not physically realistic. Thus, we used the value recommended in [153] and [190]. However, for the full scale problem, we do not adjust the material properties, and use the experimentally obtained cohesive strength of PMMA.

The model is initially discretized with 300×75 4k mesh patches, or 90,000 elements. We use the AMR to sufficiently reduce the element size at the notch tip. Note that a uniform mesh of comparable size would contain 1,440,000 elements, which is well beyond the size capacity of the GPU, thus the adaptivity is essential. The fracture patterns for three different strains are shown in Figure 5.12. Here we plot cohesive elements that have opened more the 75% of the critical opening distance. The numerical results obtained here agree well with those shown in the original experiment [203]. At lower strains the fracture surface is smoother and features one predominate crack. As the load increases, branches appear and the fractured surface becomes rougher. Finally, at the highest strain, many branches are present and are increased in length. The velocities of the three cases also increase with increased applied strain. In the lowest strain case, the velocity is relatively stable and lower than the higher strain cases. As the strain increases so do the velocity and the oscillation of the crack tip velocity.

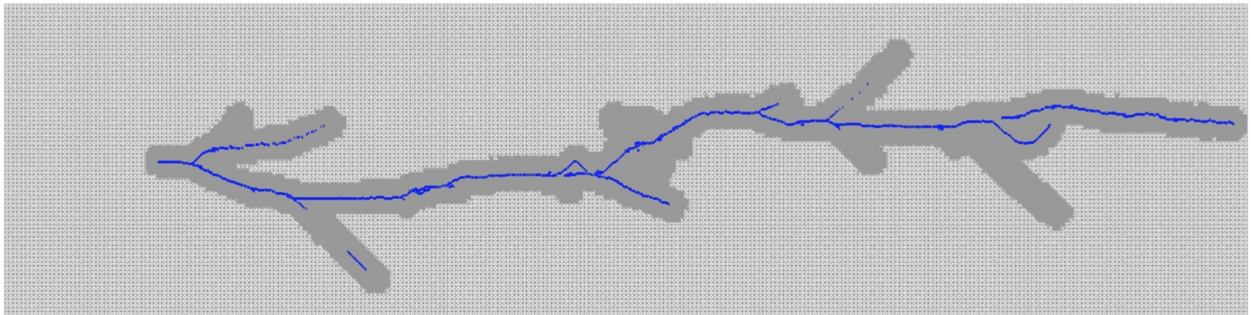
The details of the crack pattern and the adaptive mesh refinement scheme are shown in Figure 5.13 for the low strain case. Elements that are open less than 75% of the critical opening distance are shown in the zoom-in view in red. Notice that in relation to the crack branch, the branches comprised of partially open elements are quite small. The details of the refinement scheme are clear, elements within the user defined radius of a crack tip are refined. The radius of refinement is sufficiently large such that new cohesive elements will be inserted within the bounds of the refined elements.

When comparing the reduced scale model results and the full scale model results, we notice some qualitative similarities, but the details are not evident in the smaller model. Thus, whenever possible, it is recommended to use a numerical model that closely resembles the actual experiment. However, in many cases, that is not entirely feasible due to lack of access to powerful and sophisticated computational resources.

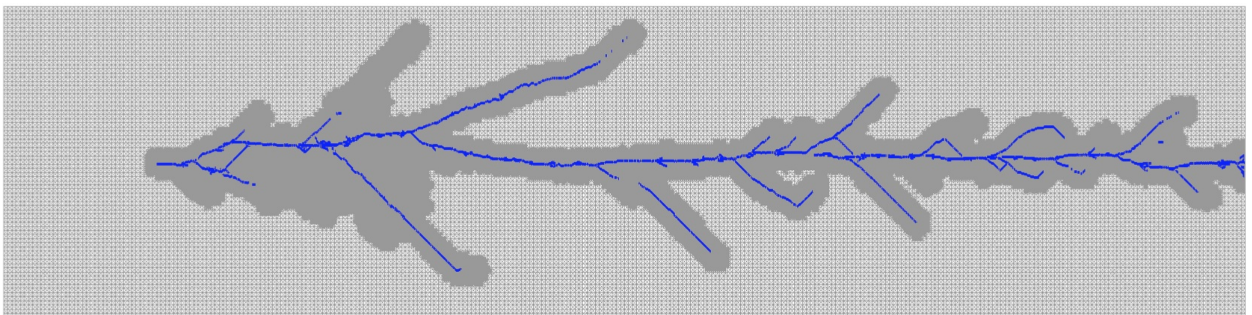
The numerical studies performed here were on homogenous domains, which made implementation on the GPU simpler because a single set of material properties could be stored for the entire model. It is, however, possible to change the implementation such that each node has independent material properties. An investigation of this nature is under development for a domain containing a weak interface, inspired by the experimental study in [205].



(a)



(b)



(c)

Figure 5.12: Final fracture patterns for full scale micro-branching problem with an externally applied strain of (a) 0.003, (b) 0.004, and (c) 0.005

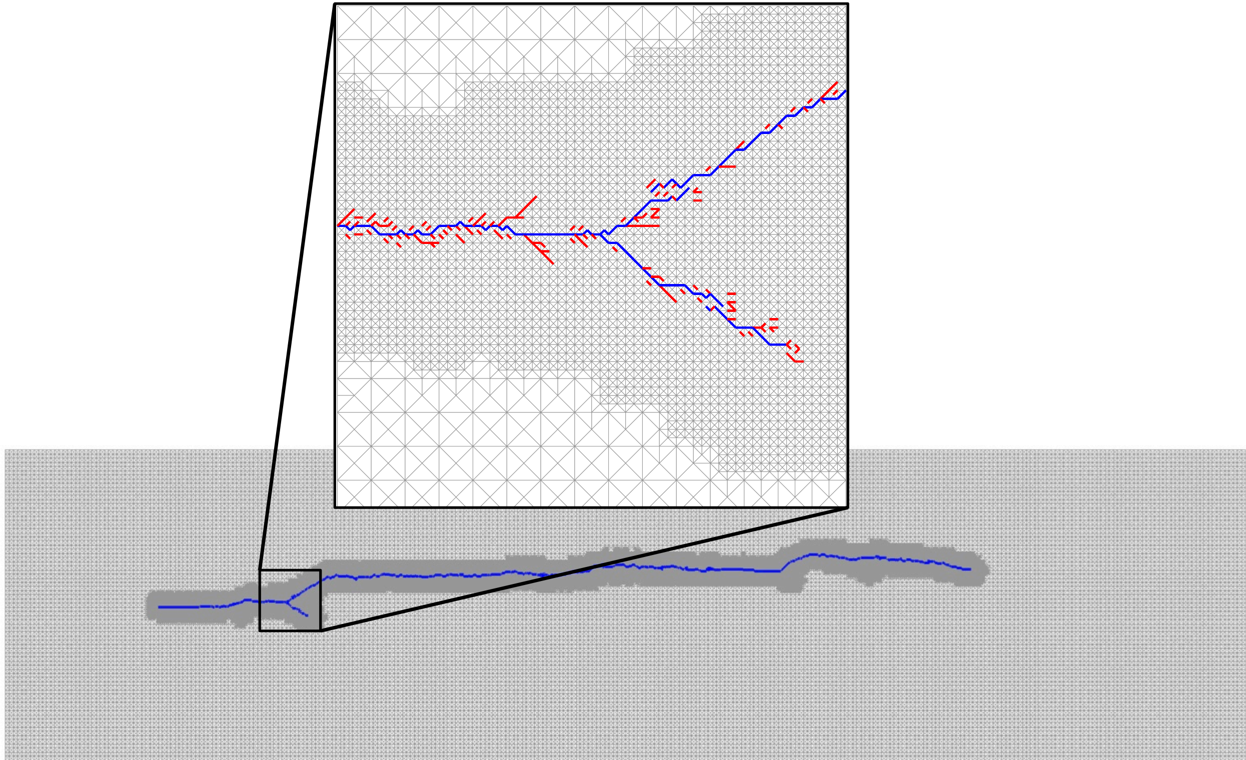


Figure 5.13: Detailed view of fracture pattern for the full scale micro-branching problem with an externally applied strain of 0.003

Chapter 6

Conclusion

Numerical simulation of fracture and failure is well established field in which a host of researchers and practitioners have contributed. Despite the decades of work on the topic, there still exist substantial challenges in computational modeling of practical systems. The contributions of this work are partially in the area of ductile failure and partially brittle fracture. In the former, failure is characterized by extremely large deformations before the creation of new surfaces. The difficulty in modeling such systems is that numerical instability is encountered before failure actually occurs. To this effect, we investigate an interpolation scheme that allows for effective transfer of internal state variables after remeshing is performed as needed in the simulation. The majority of this dissertation focuses on brittle failure, which is defined as creation of new surfaces in the domain. We perform our studies and investigations of dynamic fracture using the inter-element extrinsic cohesive model approach and the PPR potential-based cohesive model to represent crack propagation [164]. Again, mesh adaptivity is at the crux of the contributions. However, unlike the work on ductile failure where we investigate the means to physically handle mesh adaptivity without focusing on the topological changes to the mesh, in the work on brittle fracture, we actually develop the techniques to perform mesh adaptivity from the topological and physical perspectives. Efficient mesh representation is essential when the mesh changes on the fly, so this work utilizes the TopS data structure or variations upon to perform changes to the mesh as the simulation is progressing [153, 162]. This thesis places heavy focus on the computational aspects of mesh adaptation (i.e. adaptive mesh refinement and coarsening in three dimensions and GPU-based adaptive mesh refinement and coarsening), which is a critical component to enabling large scale failure simulation. In the remainder of this chapter, we summarize the main contributions of this dissertation, then we elaborate on potential future research directions.

6.1 Summary of contributions

As stated in the introduction, this work is the product of a number of fruitful collaborations. The individual contributions from the author were detailed in Section 1.3, and here the main contributions of this thesis as a whole for the simulation of fracture and failure are detailed:

- Mesh adaptivity improves the fidelity of the simulation results in modeling either failure by means of excessive deformation in ductile systems or by creation of new surfaces in brittle systems. In many cases, we have shown that adaptivity is necessary to perform the simulation.
- The Lie-group interpolation and variational recovery scheme is investigated in great detail on a model problem. We use severe cases to test the proposed algorithm and find only minor loss due to numerical diffusion. This suggests that the approach will be favorable when applied used as intended for practical problems.

- Studies on adaptive splitting of polygonal elements reveals that geometric considerations are not sufficient to determine if a mesh and element type perform well in dynamic fracture simulations. Through numerical simulations, we show that a tradeoff between geometric and computational considerations is necessary. We concluded that the adaptive restricted splitting of CVT polygonal element meshes scheme are best suited for 2D dynamic fracture problems.
- Development of 3D adaptive mesh refinement and coarsening scheme yields the ability to simulate problems that would otherwise be untenable because of limitations of computational resources. Improvement of the TopS data structure and specialized development for 3D adaptivity on 4k meshes leads to an efficient and clean implementation of refinement and coarsening that could be utilized in a variety of applications.
- Investigation of adaptive mesh refinement and coarsening schemes for dynamic fracture simulation on the massively parallel GPU architecture reveals insight into the numerical simulation that otherwise have not been investigated. While slight variation during floating point operations leads to fracture patterns that appear very different, we showed that they are statistically similar and give confidence in this and other similar implementations.

6.2 Future research directions

Several research directions are possible given the developments made in this dissertation. The proposals detailed in this section build on the work established in this document.

6.2.1 Future investigations and applications of Lie-group interpolation and variational recovery scheme

The fundamental investigations presented in Chapter 2 on mapping internal state variables using the Lie-group interpolation and variational recovery scheme gave us confidence that the approach is numerically sound and is applicable for a larger range of applications. A first extension of the work would be to examine how long the load can be applied before errors propagate and lead to a lack of numerical convergence. Hexahedra elements are utilized in this work, however, remeshing complicated domains with these elements is not straightforward. Thus there is a push to use tetrahedral elements instead.

Some initial work was done on the tetrahedral elements to evaluate their effectiveness in large deformations. The first investigation was to project a linear field from five integration points of quadratic tetrahedral elements to the ten nodes. However, five integration points cannot be consistently projected because the projection matrix cannot be integrated, 11 integration points are needed. Thus resulted in oscillatory fields, which is evident in Figure 6.1(a). Next we utilized a linear projection and constant pressure to avoid volumetric locking. The stress field became more smooth, as shown in Figure 6.1(b), but the results are still not ideal. Next, a different type of tetrahedral element was examined, the composite tetrahedral element that contains 10 nodes and utilizes multilinear shape functions and constant pressure to avoid locking. Figure 6.1(c) shows the nodal stress projected to the nodes using the composite tetrahedral elements. The composite tetrahedral results in the field that is qualitatively closest to what is expected, however, more development on the element formulation is still needed. For the composite tetrahedron to be utilized in the interpolation and recovery scheme, the nodal fields would be interpolated by the composite tetrahedral shape functions, and the element fields would be projected by the tet4 shape functions.

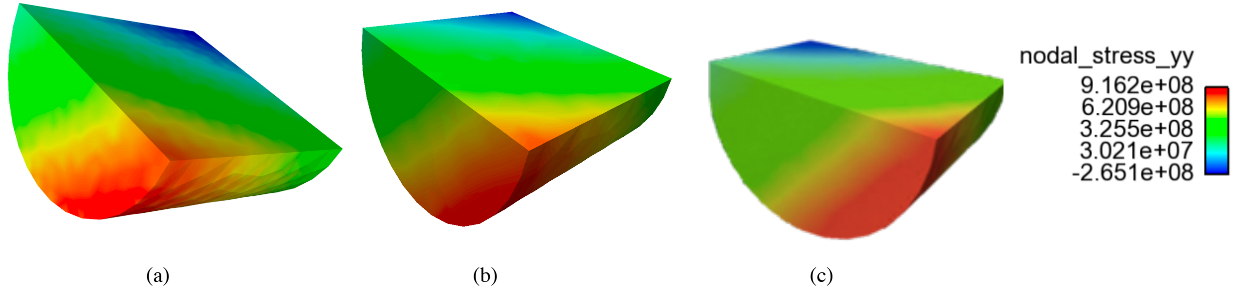


Figure 6.1: Stress field after projection from integration points to nodes of Tet10 elements using (a) quadratic projection (b) linear projection (c) composite linear projection

Once the composite tetrahedral elements are fully developed and the tensile bar specimen simulated, the attention can move to the large application of interest, which is that of a laser weld in tough metals. In tensile testing of a laser butt weld, failure is characterized by necking first and creation of new surfaces are a secondary effect. The experimental results shown in Figure 6.2 demonstrate that massive deformations occur before any new surfaces are created. This behavior is hypothesized to be due to the presence of voids in the material; their size and distribution is thought to contribute to the behavior. First attempts to model the butt weld with voids present were not successful because massive deformations in the finite elements resulted in lack of convergence, as seen in Figure 6.3. Thus, the remesh and mapping scheme with tetrahedral elements could be utilized to model the behavior of the laser weld as an attempt to capture the experimental results.

6.2.2 Investigation of activation criteria for extrinsic cohesive elements

In chapters 3-5, extrinsic cohesive zone elements are activated when the averaged tractions along an facet are greater than the material cohesive strength. To summarize, we first compute stresses at the integration points, then extrapolate to the nodes. Normal and tangential components of the tractions along the facet are computed. They are computed by averaging the contribution of each node. If the normal or tangential traction is greater than the normal or tangential cohesive strength, respectively, then the cohesive element is inserted. This approach has been used in several studies [88, 89, 101, 153, 158], and similar strength-based approaches have also been used with success [151, 159–161].

As a future investigation, one could explore different criteria to insert extrinsic cohesive elements along bulk element boundaries. First, rather than only using a single stress as an indicator and combination of stresses could be assessed as in the Mohr-Coulomb failure criterion, given by

$$\tau = \sigma \tan(\phi) + c$$

This creates a failure envelope where τ is the shear strength, σ is the normal strength, c is the intercept with the τ axis, and ϕ is the slope. While still strength-based, this type of failure criterion takes into account the interplay between the normal and tangential strength, rather than just evaluating them separately.

Loss of ellipticity is another flavor of activation criteria that has been widely studied in the analysis of material instability [206, 207]. Similarly, in dynamic fracture simulation loss of hyperbolicity has been an

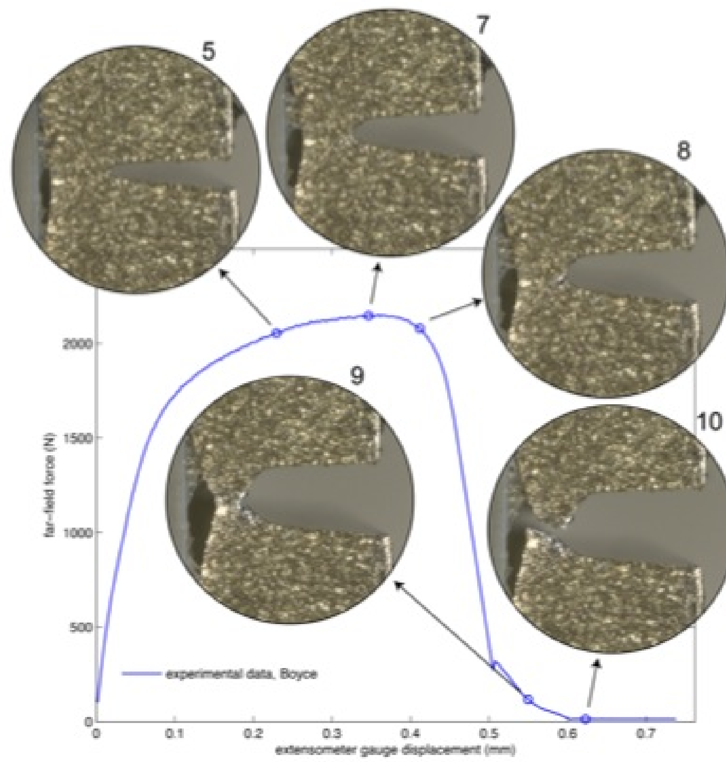


Figure 6.2: Experimental load versus displacement curve of 304L butt weld

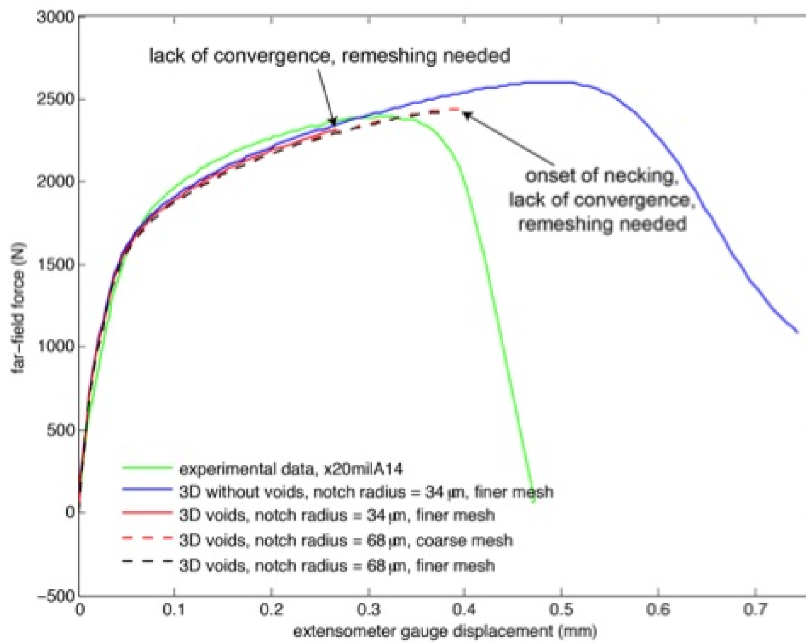


Figure 6.3: Experimental load versus displacement curve of 304L butt weld with simulated results overlaid. The voids in the material impact the overall behavior, however, current modeling attempts have been unsuccessful due to large distortions in the finite elements

indicator [48, 84]. Th ellipticity condition is summarized as

$$(\mathbf{m} \otimes \mathbf{n}) : \mathbf{C} : (\mathbf{m} \otimes \mathbf{n}) \geq 0, \qquad \forall\, \mathbf{m},\, \mathbf{n} \in \mathbb{R}^3 \tag{6.1}$$

where $\mathbf{C}$ is the elasticity tensor and $\mathbf{m}$ and $\mathbf{n}$ are unit vectors. We can define

$$\mathbf{A} = \mathbf{n} \cdot \mathbf{C} \cdot \mathbf{n} \tag{6.2}$$

then the ellipticity condition becomes

$$\mathbf{m} \cdot \mathbf{A} \cdot \mathbf{m} > 0, \quad \text{where } \mathbf{m} \in \mathbb{R}^3,\ \|\mathbf{m}\| = 1 \tag{6.3}$$

which is satisfied when

$$\det \mathbf{A} > 0. \tag{6.4}$$

The challenge lies in determining the direction of the loss of ellipticity, for which the vector $\mathbf{n}$ is parameterized. Then $\det \mathbf{A}$ becomes a function of the parameters of $\mathbf{n}$, and the ellipticity condition is a minimization problem that can be solved with standard techniques. In the case of interfacial cohesive fracture, there are a finite number of possible directions for fracture initiation, which are given by the facets in the finite element mesh. Therefore the ellipticity condition in 6.4 does not need to be parameterized, instead all of the possible fracture directions could be evaluated.

In the previous discussion the activation of cohesive elements was assumed to be the same at all points in the simulation (i.e. during crack initiation, propagation, and branching). However, one could explore different criteria for crack initiation than for crack branching.It is known that the crack tip velocity is limited by the wave speed of the material; in experimental systems micro-branching off the main crack serves as a mechanism to limit the speed [203]. In [53], researchers used the crack tip velocity as a criteria to initiation crack branching and ultimately limit the speed of the main crack. A weakness of the method employed in [53] is that it allowed for only y-type branches. A more sophisticated method could be developed in which the crack velocity activates branching, but the direction and length of the branches could be calculated using other means.

6.2.3 Automatic remeshing around crack tips

While the scope of the work presented in this dissertation was broad, a common theme running through all of the chapters was the notion of mesh adaptivity. We have seen the great computational benefit that adaptive mesh refinement can have on simulations of any size, and how local modification of polygonal element meshes results in improved fracture patterns over structured meshes. Thus, a natural extension and coalescence of the work presented in this chapter is adaptive remeshing of unstructured meshes. Straightforward subdivision of the polygonal element meshes with adaptive element splitting has already been investigated in [89]. So, a further extension we propose here is to completely remesh the crack tip, rather than hierarchically refining the polygons or the 4k patches.

A meshing technique of this nature has recently been developed [208] in which a planar curved domain is meshed with triangular elements by transforming a background mesh to conform the the domain boundary. This method is also appropriate for evolving domains in which the same background mesh (the so-called "universal mesh") is utilized to triangulate many instances of the domain. Quasi-static curvilinear crack

propagation in two dimensions has recently been simulated using this meshing approach [209].

A similar technique has been developed by [210] (collaborators on the work presented in this dissertation) for quadrilateral and triangular element meshes. In their approach, the mesh can be generated around an arbitrary line (e.g. crack or notch) in the mesh. This is demonstrated in Figure 6.4 on a simply geometry with a random and complex fracture pattern around which the mesh is generated. After the nodes are moved, the quality of mesh is likely degraded, so a smoothing algorithm is run to improve the modified and adjacent elements. Finally, the mesh may be converted to a quadrilateral mesh using the clustering algorithm proposed by [211].

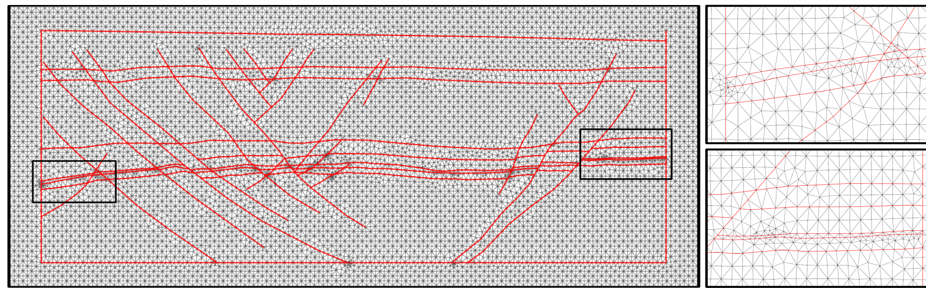
The approach requires an initial triangular mesh of the problem domain without the constraints. Araujo and Celes [210] investigated a triangular Delaunay mesh and a 4k mesh as a starting point although any initial triangular is applicable. Next, the constraints are inserted one at a time and the mesh adjusted for one constraint before the next is inserted. First the end points of the constraint polyline are inserted and the nodes of the initial mesh are adjusted such that they are coincident. Next, an iterative procedure modifies the mesh such that the polyline is represented by mesh nodes to a tolerance.

While the meshing capability is present, the approach has not been utilized in an actual fracture application. In order to make this approach viable, a few assumptions related to the crack propagation criteria would need to be modified. First, the approach used throughout this dissertation assumes that the crack can only propagate to a fixed number of points through element facets. If arbitrary remeshing is possible, then the actual crack direction can be realized. Using a stress based approach for brittle materials, the direction and magnitude of the principle stress at the crack tip could be determined. If this stress is greater than the cohesive strength, then the area would be remeshed around the constraint of the new crack increment. Using the current extrinsic cohesive zone model approach, the cohesive element would be inserted by duplicating the nodes on the newly added constraint facet.

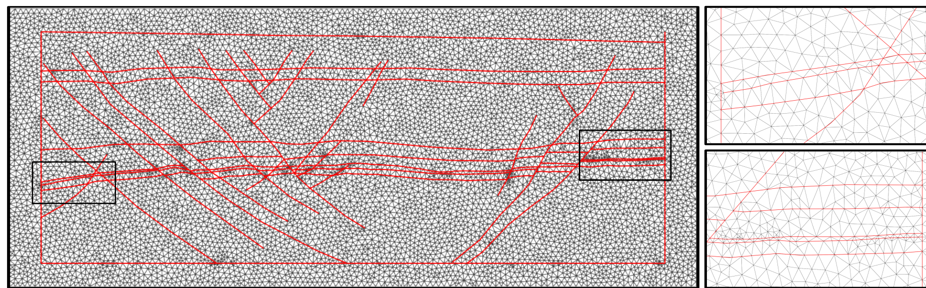
The work under development could also be extended to polygonal element meshes. Ideally, the new criteria for crack propagation, it is clear that the necessity for element splitting no longer exists because the crack direction would be determined independent of the mesh orientation ahead of the crack tip. However, it may not be computationally feasible to remesh the regions ahead of crack tips or globally throughout the domain every time the crack advances. Thus, we can imagine a hybrid approach in which a region ahead of the crack tip is remeshed at a certain number of time step intervals, for example, then the element splitting could still be utilized to capture the change in direction of the crack propagation in between remeshing steps.

The proposed remeshing algorithm is applied to the entire mesh (e.g. the entire mesh is regenerated every time the crack advances). However, to improve computational costs, the mesh should only be changed locally in regions near crack tips or where a greater mesh resolution is necessary. Thus, a additional component of this proposed work would be to develop an algorithm to perform the non hierarchical remeshing on subregions of the mesh while maintaining nodal compatibility with the regions of the mesh that are not modified.

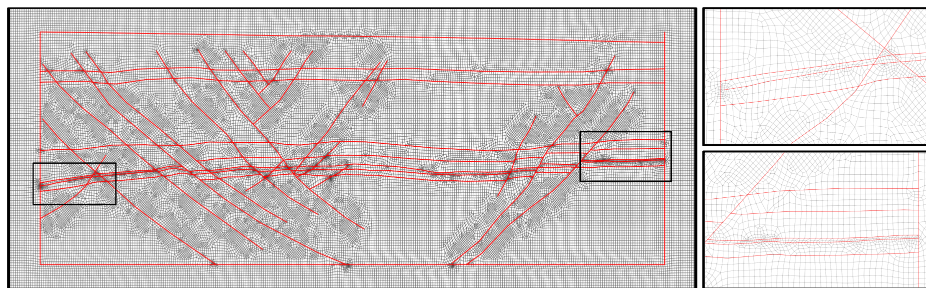
Whether the remeshing is performed on a local or global basis, the user needs to specify the target element size. In the work presented in this thesis, the element sizes were determined based on a a-priori assumptions and parametric studies of mesh refinement. However, to make the simulations more robust and predictive, the size of the elements should be calculated based on information of the state of the system. Error estimators may be used to achieve this target element size. In previous works, researchers used error indicators based on interpolation error estimates to adaptively refine and coarsen a mesh to capture anisotropic damage [212]. Khoei and coworkers have modeled three-dimensional crack propagation in quasi-static problems. Error indicators in the stress field indicate the target mesh size, then adaptive remeshing is



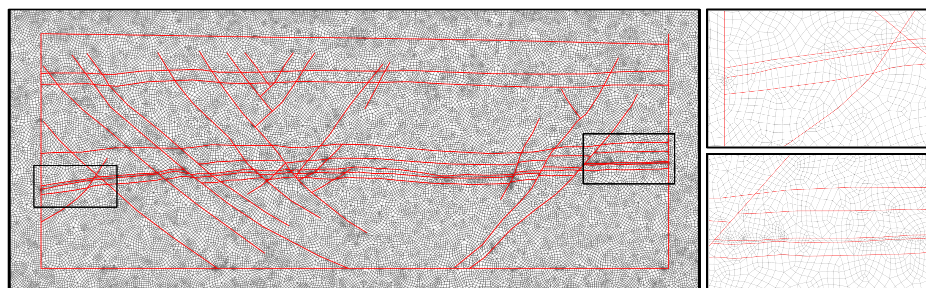
(a)



(b)



(c)



(d)

Figure 6.4: Polygonal remeshing around the constraint of an arbitrary crack pattern shown in red (a) 4k based triangular mesh (b) Delaunay based triangular mesh (c) 4k based quadrilateral mesh (d) Delaunay based quadrilateral mesh

performed before the crack propagates [127, 213–215]. A hierarchical adaptivity approach is taken in [216] to drive crack propagation based on *a posteriori* error estimations in two dimensions. The previous works featured h-adaptivity, whereby the mesh is locally refined or coarsened, i.e., the number of elements and nodes change. Alternatively, r-adaptivity could also be used in fracture simulation. In this approach, the mesh is moved but no elements or nodes are added. For example, brittle fracture is simulated in [217] by adaptively aligning the element facets with the crack propagation direction.

6.2.4 Polyhedra finite elements for dynamic fracture simulation

While the adaptive splitting of polygonal elements in two dimensions presented in Chapter 3 is not conducive to three-dimensional fracture on polyhedral elements, we indicate some directions to be pursued to achieve that goal. For instance, recently, the Wachspress shape functions, together with their gradients, have been derived in 3D [218] and either those or other related interpolants [219] could serve as a starting point. The geometrical treatment regard- ing three-dimensional splitting and element redefinition may incorporate some of the computational geometry ideas developed at Sandia National Laboratory [220] and in reference [221]. Alternatively, new interpolants may open avenues for the three-dimensional treatment of fracture surfaces on poly- topes. In this regard, the virtual element method (VEM) may be explored. Recently, the VEM has been implemented to treat elastic problems without cracks [222], and thus, one could build upon this work to incorporate new crack surfaces where and when needed, which could take advantage of the adaptive features offered by the VEM [223, 224].

6.2.5 Adaptive time stepping for explicit dynamic fracture

A critical issue in explicit dynamic simulations, like the ones performed in Chapters 3-5 of this dissertation, is that of the critical time step. It is well known that the time step for the dynamic simulation is limited by the CFL condition [225]. Intuitively, the CFL condition states that the distance traveled by a wave in one time step must be less than the spatial time step, or in other words, the numerical wave speed must be at least as fast as the physical wave speed. In the simulation of brittle fracture, we have seen that we need quite a very small element size around the crack tip and other areas of high gradient in the domain. However, with the use of adaptive mesh refinement and coarsening techniques, the mesh is coarse in regions away from the crack tip. Since we are using a uniform time step throughout the spatial domain, the time step associated with the largest elements is much smaller than is necessary according to the CFL condition. This leads to the idea to add an additional level of adaptivity to the simulation: vary the time step spatially throughout the mesh based on the minimum element size of the region. Techniques of temporal adaptivity, which the domain is subdivided and different time steps applied at different regions of the mesh have been proposed in the literature, but the extension to the the extrinsic fracture simulation has yet to be done to the best knowledge of the author.

The previous work of [5, 226] proposed methods for adaptive temporal and spatial grid spacing. For a simple case of one spatial dimension, the idea is illustrated in Figure 6.5. The mesh is spatially refined in the middle of mesh. In the approach used through this dissertation, the time stepping was done in accordance with Figure 6.5(a) where the time step (Δt_A) is the same everywhere throughout the spatial mesh. However, the proposed approach would follow that of Figure 6.5(b) where the small time step is only used at the fine elements (Δt_B) and a larger time step is utilized for the coarser spatial mesh (Δt_A). Of course, we need to take care to transfer information between the coarse and fine time steps. Displacements are calculated at

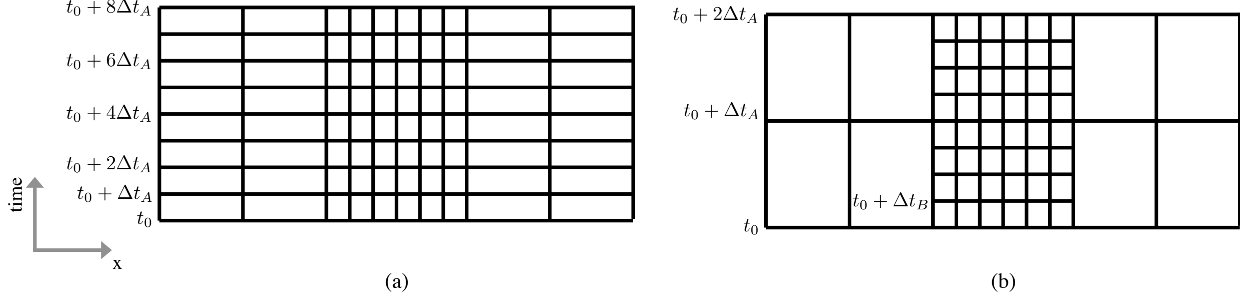


Figure 6.5: Temporal adaptivity illustrated on a one-dimensional spatial mesh (a) a uniform time step is used throughout the domain (b) a smaller time step is used in areas of mesh refinement (Δt_B) and a larger time step is used in the other areas (Δt_A). Figure adapted from [5]

auxiliary points that exist between the fine time steps and coarse time steps.

Other new approaches may be of use in developing an adaptive time stepping scheme. Prakash and Hjelmstad [227] for example, have developed a multi-time-step coupling method in which the entire domain is divided into smaller subdomains. Essentially, each subdomain is treated and solved separately, meaning that each one may be integrated using a different critical time step based on its smallest element size. In order to assemble the domain after the subdomains are computed, internal forces are computed then subdomains are updated to account for the internal forces coming from neighboring subdomains such that continuity of velocity at interfaces is achieved. Combescure and Gravouil [228] demonstrated good agreement between a multi-time-step coupling scheme and a standard time stepping scheme, however the approach has not been incorporated into a problem in which the spatial mesh also changes with time. While significant development would be necessary, an adaptive time stepping approach based on spatial discretization would provide greater flexibility and computational efficiency in the context of the adaptive spatial mesh refinement and coarsening algorithms.

6.2.6 Three dimensional adaptive mesh refinement and coarsening on graphical processing units

Based on the mesh adaptivity on the GPU work done in Chapter 5, some extensions are clear. First the work was conducted in two dimensions, so an extension to three dimensions is obvious. However, given the current implementation on a single GPU, the size of the problem is greatly limited due to the GPU architecture and access to memory. See Section 5.1 for a discussion of the GPU architecture and resulting memory access issues. Also, recall that the reason AMR+C was performed on the GPU in the first place was not because the simulation time was too great with a uniform mesh, but because the space requirements were so great that a large scale problem could not actually be represented. Thus, in order to perform adaptivity in 3D, the implementation would need to be transferred to multiple GPUs so that there is access to more memory.

A natural first step would be to start with a uniform mesh and adaptive insertion of cohesive elements in 3D on multiple GPUs. In this way, different parts of the model would be simulated on different GPUs. Thus, the simulations parameters that take the most memory (e.g. stiffness and mass matrix) are stored separately for each part of the model on its respectively GPU. A uniform mesh would likely utilize a graph coloring scheme, which would need to be handled first on a CPU. Since the CPU memory space is not an issue as it is on the GPU, we can simply gather this computational time as overhead before the simulation begins. This extension is currently underway in the coauthors of the work presented in Chapter 5. Next,

adaptivity would be incorporated with the 3D simulation on multiple GPUs. As was the case for the 2D adaptivity, the coloring scheme would not be appropriate on multiple GPUs, thus a nodal scheme would need to be developed as it was in Chapter 5.

Chapter 7

Bibliography

[1] C. Pelties, J. de la Puente, J.-P. Ampuero, G. B. Brietzke, and M. Käser, "Three-dimensional dynamic rupture simulation with a high-order discontinuous Galerkin method on unstructured tetrahedral meshes," *Journal of Geophysical Research*, vol. 117, no. 2, p. B02309, 2012.

[2] G. H. van Lenthe and R. Müller, "CT-based visualization and quantification of bone microstructure in vivo," *IBMS BoneKEy*, vol. 5, no. 11, pp. 410–425, 2008.

[3] B. R. Lawn, *Fracture of brittle solids.* Cambridge: CAmbridge University Press, 1993.

[4] K. D. Papoulia, S. A. Vavasis, and P. Ganguly, "Spatial convergence of crack nucleation using a cohesive finite-element model on a pinwheel-based mesh," *International Journal for Numerical Methods in Engineering*, vol. 67, no. 1, pp. 1–16, 2006.

[5] Z. Wanxie, X. Zhuang, and J. Zhu, "A self-adaptive time integration algorithm for solving partial differential equations," *Applied Mathematics and Computation*, vol. 89, no. 1-3, pp. 295–312, 1998.

[6] S. Hao, "I-35W Bridge collapse," *Journal of Bridge Engineering*, vol. 15, no. 5, pp. 608–614, 2009.

[7] H. M. Salem and H. M. Helmy, "Engineering Structures," *Engineering Structures*, vol. 59, pp. 635–645, 2014.

[8] Z. P. Bažant and M. Verdure, "Mechanics of progressive collapse: Learning from World Trade Center and building demolitions," *Journal of Engineering Mechanics*, vol. 133, no. 3, pp. 308–319, 2007.

[9] F. W. Flocker and L. R. Dharani, "Low velocity impact resistance of laminated architectural glass," *Journal of architectural engineering*, vol. 4, no. 1, pp. 12–17, 1998.

[10] S. Zhu, D. Levinson, H. X. Liu, and K. Harder, "The traffic and behavioral effects of the I-35W Mississippi River bridge collapse," *Transportation research part A: policy and practice*, vol. 44, no. 10, pp. 771–784, 2010.

[11] L. A. Dalguer, "Numerical algorithms for earthquake rupture dynamic modeling ," in *The mechanics of faulting: From laboratory to earthquakes* (A. Bizzarri and H. S. Bhat, eds.), Research Signpost, 2012.

[12] Y. Ben-Zion, "Dynamic ruptures in recent models of earthquake faults," *Journal of the Mechanics and Physics of Solids*, vol. 49, no. 9, pp. 2209–2244, 2001.

[13] Y. Ben-Zion and Z. Shi, "Dynamic rupture on a material interface with spontaneous generation of plastic strain in the bulk," *Earth and Planetary Science Letters*, vol. 236, no. 1-2, pp. 486–496, 2005.

[14] X. Lu, N. Lapusta, and A. J. Rosakis, "Pulse-like and crack-like dynamic shear ruptures on frictional interfaces: experimental evidence, numerical modeling, and implications," *International Journal of Fracture*, vol. 163, no. 1-2, pp. 27–39, 2010.

[15] S. Tanimura, K. Mimura, T. Nonaka, and W. Zhu, "Dynamic failure of structures due to the great Hanshin-Awaji earthquake," *International journal of impact engineering*, vol. 24, no. 6, pp. 583–596, 2000.

[16] D. G. Lignos, Y. Chung, T. Nagae, and M. Nakashima, "Numerical and experimental evaluation of seismic capacity of high-rise steel buildings subjected to long duration earthquakes ," *Computers and Structures*, vol. 89, no. 11-12, pp. 959–967, 2011.

[17] B. Mobasher, M. S. Mamlouk, and H.-M. Lin, "Evaluation of crack propagation properties of asphalt mixtures," *Journal of Transportation Engineering*, vol. 123, no. 5, pp. 405–413, 1997.

[18] M. N. Guddati, Z. Feng, and Y. R. Kim, "Toward a micromechanics-based procedure to characterize fatigue performance of asphalt concrete," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1789, no. 1, pp. 121–128, 2002.

[19] M. P. Wagoner, W. G. Buttlar, G. H. Paulino, and P. Blankenship, "Investigation of the fracture resistance of hot-mix asphalt concrete using a disk-shaped compact tension test," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1929, no. 1, pp. 183–192, 2005.

[20] H. Kim, M. P. Wagoner, and W. G. Buttlar, "Simulation of fracture behavior in asphalt concrete using a heterogeneous cohesive zone discrete element model," *Journal of Materials in Civil Engineering*, vol. 20, no. 8, pp. 552–563, 2008.

[21] E. V. Dave, S. Leon, and K. Park, "Thermal Cracking Prediction Model and Software for Asphalt Pavements," *T&DI Congress 2011: Integrated Transportation and Development for a Better Tomorrow*, pp. 667–676, 2011.

[22] E. Dave, G. H. Paulino, and W. G. Buttlar, "Viscoelastic Functionally Graded Finite Element Method Using Correspondence Principle," *ASCE: Journal of Materials in Civil Engineering*, vol. 23, no. 1, 2011.

[23] D. Hersman, R. Sumwalt, C. Hart, M. R. Rosekind, and E. F. Weener, "Southwest Airlines, Flight 812," Tech. Rep. DCA11MA039, 2013.

[24] J. L. Beuth and J. W. Hutchinson, "Fracture analysis of multi-site cracking in fuselage lap joints," *Computational Mechanics*, vol. 13, no. 5, pp. 315–331, 1994.

[25] H. T. Budiman, K. F. Henault, and P. A. Lagace, "Effects of Longitudinal and Hoop Stiffeners on Damage Propagation in Pressurized Composite Cylinders," *AIAA Journal*, vol. 35, no. 1, pp. 145–151, 1997.

[26] I. Scheider and W. Brocks, "Cohesive elements for thin-walled structures," *Computational Materials Science*, vol. 37, no. 1-2, pp. 101–109, 2006.

[27] R. Zimmermann, H. Klein, and A. Kling, "Buckling and postbuckling of stringer stiffened fibre composite curved panels – Tests and computations," *COMPOSITE STRUCTURES*, vol. 73, no. 2, pp. 150–161, 2006.

[28] W. Duan and S. Joshi, "Structural behavior of large-scale triangular and trapezoidal threaded steel tie rods in assembly using finite element analysis ," *Engineering Failure Analysis*, vol. 34, no. C, pp. 150–165, 2013.

[29] M. Fulland, M. Sander, G. Kullmer, and H. A. Richard, "Analysis of fatigue crack propagation in the frame of a hydraulic press," *Engineering Fracture Mechanics*, vol. 75, no. 3-4, pp. 892–900, 2008.

[30] A. Sekhar and B. S. Prabhu, "Transient analysis of a cracked rotor passing through critical speed," *Journal of Sound and Vibration*, vol. 173, no. 3, pp. 451–421, 1994.

[31] K. K. Ray, N. Narasaiah, and R. Sivakumar, "Studies on short fatigue crack growth behavior of a plain carbon steel using a new specimen configuration," *Materials Science and Engineering: A*, vol. 372, no. 1-2, pp. 81–90, 2004.

[32] R. O. Ritchie, "The conflicts between strength and toughness," *Nature Publishing Group*, vol. 10, no. 11, pp. 817–822, 2011.

[33] J. J. Lewandowski, W. H. Wang, and A. L. Greer, "Intrinsic plasticity or brittleness of metallic glasses," *Philosophical Magazine Letters*, vol. 85, no. 2, pp. 77–87, 2005.

[34] D. C. Hofmann, J.-Y. Suh, A. Wiest, G. Duan, M.-L. Lind, M. D. Demetriou, and W. L. Johnson, "Designing metallic glass matrix composites with high toughness and tensile ductility," *Nature*, vol. 451, no. 7182, pp. 1085–1089, 2008.

[35] I. V. Yannas and J. F. Burke, "Design of an artificial skin. I. Basic design principles," *Journal of biomedical materials research*, vol. 14, no. 1, pp. 65–81, 1980.

[36] N. J. Glassmaker, A. Jagota, C.-Y. Hui, W. L. Noderer, and M. K. Chaudhury, "Biologically inspired crack trapping for enhanced adhesion," *Proceedings of the National Academy of Sciences*, vol. 104, no. 26, pp. 10786–10791, 2007.

[37] T. Kokubo, H.-M. Kim, and M. Kawashita, "Novel bioactive materials with different mechanical properties," *Biomaterials*, vol. 24, pp. 2161–2175, June 2003.

[38] I. Carol, C. M. L o pez, and O. Roa, "Micromechanical analysis of quasi-brittle materials using fracture-based interface elements," *International Journal for Numerical Methods in Engineering*, vol. 52, no. 12, pp. 193–215, 2001.

[39] H. Gao, "Application of fracture mechanics concepts to hierarchical biomechanics of bone and bone-like materials," *International Journal of Fracture*, vol. 138, no. 1-4, pp. 101–137, 2006.

[40] D. Garcia, P. K. Zysset, M. Charlebois, and A. Curnier, "A three-dimensional elastic plastic damage constitutive law for bone tissue," *Biomechanics and Modeling in Mechanobiology*, vol. 8, no. 2, pp. 149–165, 2008.

[41] R. Hambli and S. Allaoui, "A Robust 3D Finite Element Simulation of Human Proximal Femur Progressive Fracture Under Stance Load with Experimental Validation," *Annals of Biomedical Engineering*, vol. 41, no. 12, pp. 2515–2527, 2013.

[42] A. Mota, W. S. Klug, M. Ortiz, and A. Pandolfi, "Finite-element simulation of firearm injury to the human cranium," *Computational Mechanics*, vol. 31, pp. 115–121, May 2003.

[43] D. O. Halwani, P. G. Anderson, B. C. Brott, A. S. Anayiotos, and J. E. Lemons, "The role of vascular calcification in inducing fatigue and fracture of coronary stents," *Journal of Biomedical Materials Research Part B: Applied Biomaterials*, vol. 100B, pp. 292–304, Sept. 2011.

[44] T. L. Anderson, *Fracture mechanics: fundamentals and applications*. CRC Press, 3 ed., 2005.

[45] J. R. Rice, "The elastic-plastic mechanics of crack extension," *International Journal of Fracture Mechanics*, vol. 4, no. 1, pp. 41–47, 1968.

[46] Z.-H. Jin, G. H. Paulino, and R. H. Dodds, Jr., "Cohesive fracture modeling of elastic–plastic crack growth in functionally graded materials," *Engineering Fracture Mechanics*, vol. 70, pp. 1885–1912, Sept. 2003.

[47] A. Agwai, I. Guven, and E. Madenci, "Predicting crack initiation and propagation using XFEM, CZM and peridynamics: A comparative study," *2010 Proceedings 60th Electronic Components and Technology Conference (ECTC)*, pp. 1178–1185, 2010.

[48] T. Belytschko, H. Chen, J. Xu, and G. Zi, "Dynamic crack propagation based on loss of hyperbolicity and a new discontinuous enrichment," *International Journal for Numerical Methods in Engineering*, vol. 58, no. 12, pp. 1873–1905, 2003.

[49] S. Li and S. Ghosh, "Extended Voronoi cell finite element model for multiple cohesive crack propagation in brittle materials," *International Journal for Numerical Methods in Engineering*, vol. 65, no. 7, pp. 1028–1067, 2006.

[50] J. L. Asferg, P. N. Poulsen, and L. O. Nielsen, "A consistent partly cracked XFEM element for cohesive crack growth," *International Journal for Numerical Methods in Engineering*, vol. 72, no. 4, pp. 464–485, 2007.

[51] J. F. Mougaard, P. N. Poulsen, and L. O. Nielsen, "An enhanced cohesive crack element for XFEM using a double enriched displacement field," in *6th International Conference on Fracture Mechanics of Concrete and Concrete Structures*, pp. 139–146, 2007.

[52] T. Menouillard, J. Rethore, N. Moes, A. Combescure, and H. Bung, "Mass lumping strategies for X-FEM explicit dynamics: Application to crack propagation," *International Journal for Numerical Methods in Engineering*, vol. 74, no. 3, pp. 447–474, 2008.

[53] F. Armero and C. Linder, "Numerical simulation of dynamic fracture using finite elements with embedded discontinuities," *International Journal of Fracture*, vol. 160, no. 2, pp. 119–141, 2009.

[54] J. Rethore, S. Roux, and F. Hild, "Hybrid analytical and extended finite element method (HAX-FEM): A new enrichment procedure for cracked solids," *International Journal for Numerical Methods in Engineering*, vol. 81, no. 3, pp. 269–285, 2009.

[55] R. Radulovic, O. T. Bruhns, and J. Mosler, "Effective 3D failure simulations by combining the advantages of embedded Strong Discontinuity Approaches and classical interface elements," *Engineering Fracture Mechanics*, vol. 78, no. 12, pp. 2470–2485, 2011.

[56] C. L. Richardson, J. Hegemann, E. Sifakis, J. Hellrung, and J. M. Teran, "An XFEM method for modeling geometrically elaborate crack propagation in brittle materials," *International Journal for Numerical Methods in Engineering*, vol. 88, no. 10, pp. 1042–1065, 2011.

[57] S. Loehnert, C. Prange, and P. Wriggers, "Error controlled adaptive multiscale XFEM simulation of cracks," *International Journal of Fracture*, vol. 178, no. 1-2, pp. 147–156, 2012.

[58] V. Gupta, D.-J. Kim, and C. A. Duarte, "Analysis and improvements of global–local enrichments for the Generalized Finite Element Method ," *Computer Methods in Applied Mechanics and Engineering*, vol. 245-246, pp. 47–62, 2012.

[59] J. Garzon, P. O'Hara, C. A. Duarte, and W. G. Buttlar, "Improvements of explicit crack surface representation and update within the generalized finite element method with application to three-dimensional crack coalescence," *International Journal for Numerical Methods in Engineering*, vol. 97, no. 4, pp. 231–273, 2013.

[60] N. Moes and T. Belytschko, "Extended finite element method for cohesive crack growth," *Engineering Fracture Mechanics*, vol. 69, no. 7, pp. 813–833, 2002.

[61] M. Jirásek, "Comparative study on finite elements with embedded discontinuities," *Computer Methods in Applied Mechanics and Engineering*, vol. 188, no. 1, pp. 307–330, 2000.

[62] M. Ortiz, Y. Leroy, and A. Needleman, "A finite element method for localized failure analysis," *Computer Methods in Applied Mechanics and Engineering*, vol. 61, no. 2, pp. 189–214, 1987.

[63] T. Belytschko, J. Fish, and B. E. Engelmann, "A finite element with embedded localization zones," *Computer Methods in Applied Mechanics and Engineering*, vol. 70, no. 1, pp. 59–89, 1988.

[64] J. Fish and T. Belytschko, "Elements with embedded localization zones for large deformation problems," *Computers and Structures*, vol. 30, no. 1, pp. 247–256, 1988.

[65] E. N. Dvorkin, A. M. Cuitiño, and G. Gioia, "Finite elements with displacement interpolated embedded localization lines insensitive to mesh size and distortions," *International Journal for Numerical Methods in Engineering*, vol. 30, no. 3, pp. 541–564, 1990.

[66] E. N. Dvorkin and A. P. Assanelli, "2D finite elements with displacement interpolated embedded localization lines: the analysis of fracture in frictional materials," *Computer Methods in Applied Mechanics and Engineering*, vol. 90, no. 1, pp. 829–844, 1991.

[67] Y. D. Ha and F. Bobaru, "Studies of dynamic crack propagation and crack branching with peridynamics," *International Journal of Fracture*, vol. 162, no. 1-2, pp. 229–244, 2010.

[68] F. Bobaru and Y. D. Ha, "Adaptive refinement and multiscale modeling in 2D peridynamics," *International Journal for Multiscale Computational Engineering*, vol. 9, no. 6, pp. 635–659, 2011.

[69] W. Liu and J.-W. Hong, "Discretized peridynamics for brittle and ductile solids," *International Journal for Numerical Methods in Engineering*, vol. 89, no. 8, pp. 1028–1046, 2011.

[70] A. Seagraves and R. Radovitzky, "Advances in Cohesive Zone Modeling of Dynamic Fracture," in *Dynamic Failure of Materials and Structures* (A. Shukla, G. Ravichandran, and Y. D. S. Rajapakse, eds.), Boston, MA: Springer US, 2010.

[71] R. Radovitzky, A. Seagraves, M. Tupek, and L. Noels, "A scalable 3D fracture and fragmentation algorithm based on a hybrid, discontinuous Galerkin, cohesive element method," *Computer Methods in Applied Mechanics and Engineering*, vol. 200, no. 1-4, pp. 326–344, 2011.

[72] R. Abedi, S.-H. Chung, M. A. Hawker, J. Palaniappan, and R. B. Haber, "Modeling Evolving Discontinuities with Spacetime Discontinuous Galerkin Methods," in *IUTAM Symposium on Discretization Methods for Evolving Discontinuities* (A. Combescure, R. de borst, and T. Belytschko, eds.), pp. 59–87, 2007.

[73] R. Abedi, M. A. Hawker, R. B. Haber, and K. Matouš, "An adaptive spacetime discontinuous Galerkin method for cohesive models of elastodynamic fracture," *International Journal for Numerical Methods in Engineering*, vol. 81, no. 10, pp. 1207–1241, 2010.

[74] C. Farhat, I. Harari, and U. Hetmaniuk, "The discontinuous enrichment method for multiscale analysis," *Computer Methods in Applied Mechanics and Engineering*, vol. 192, no. 28, pp. 3195–3209, 2003.

[75] G. J. Wagner and W. K. Liu, "Coupling of atomistic and continuum simulations using a bridging scale decomposition," *Journal of Computational Physics*, vol. 190, no. 1, pp. 249–274, 2003.

[76] J. Fish and W. Chen, "Discrete-to-continuum bridging based on multigrid principles," *Computer Methods in Applied Mechanics and Engineering*, vol. 193, no. 17, pp. 1693–1711, 2004.

[77] H. B. Dhia and G. Rateau, "The Arlequin method as a flexible engineering design tool," *International Journal for Numerical Methods in Engineering*, vol. 62, no. 11, pp. 1442–1462, 2005.

[78] S. Zhang, R. Khare, Q. Lu, and T. Belytschko, "A bridging domain and strain computation method for coupled atomistic–continuum modelling of solids," *International Journal for Numerical Methods in Engineering*, vol. 70, no. 8, pp. 913–933, 2007.

[79] P. Aubertin, J. R e thor e, and R. D. Borst, "A coupled molecular dynamics and extended finite element method for dynamic crack propagation," *International Journal for Numerical Methods in Engineering*, vol. 81, no. 1, pp. 72–88, 2009.

[80] J.-H. Song, H. Wang, and T. Belytschko, "A comparative study on finite element methods for dynamic fracture," *Computational Mechanics*, vol. 42, no. 2, pp. 239–250, 2007.

[81] J. O. Hallquist, *LS-DYNA Theory Manual*. Livermore, California, 2006.

[82] P. A. Klein, J. W. Foulk, E. P. Chen, S. A. Wimmer, and H. J. Gao, "Physics-based modeling of brittle fracture: cohesive formulations and the application of meshfree methods," *Theoretical and Applied Fracture Mechanics*, vol. 37, no. 1, pp. 99–166, 2001.

[83] T. Rabczuk and T. Belytschko, “Cracking particles: a simplified meshfree method for arbitrary evolving cracks,” *International Journal for Numerical Methods in Engineering*, vol. 61, no. 13, pp. 2316–2343, 2004.

[84] H. Gao and P. Klein, “Numerical simulation of crack growth in an isotropic solid with randomized internal cohesive bonds,” *Journal of the Mechanics and Physics of Solids*, vol. 46, no. 2, pp. 187–218, 1998.

[85] K. Park, G. H. Paulino, and J. R. Roesler, “Virtual Internal Pair-Bond Model for Quasi-Brittle Materials,” *Journal of Engineering Mechanics*, vol. 134, no. 10, pp. 1–11, 2008.

[86] D. Dugdale, “Yielding of steel sheets containing slits,” *Journal of the Mechanics and Physics of Solids*, vol. 8, no. 2, pp. 100–104, 1960.

[87] G. I. Barenblatt, “The mathematical theory of equilibrium cracks in brittle fracture,” *Advances in applied mechanics*, vol. 7, no. 55-129, p. 104, 1962.

[88] G. H. Paulino, K. Park, W. Celes, and R. Espinha, “Adaptive dynamic cohesive fracture simulation using nodal perturbation and edge-swap operators,” *International Journal for Numerical Methods in Engineering*, vol. 84, no. 11, pp. 1303–1343, 2010.

[89] D. W. Spring, S. E. Leon, and G. H. Paulino, “Unstructured polygonal meshes with adaptive refinement for the numerical simulation of dynamic cohesive fracture,” vol. 189, no. 1, pp. 33–57.

[90] E. V. Dave, W. G. Buttlar, S. E. Leon, B. Behnia, and G. H. Paulino, “IlliTC – low-temperature cracking model for asphalt pavements,” *Road Materials and Pavement Design*, vol. 14, pp. 57–78, Aug. 2013.

[91] K. Park and G. H. Paulino, “Cohesive zone models: a critical review of traction-separation relationships across fracture surfaces,” *Applied Mechanics Reviews*, vol. 64, no. 6, p. 060802, 2011.

[92] M. Falk, A. Needleman, and J. Rice, “A critical evaluation of cohesive zone models of dynamic fracture,” *Journal de Physique. IV*, vol. 11, no. 5, pp. 43–50, 2001.

[93] K. D. Papoulia, C.-H. Sam, and S. A. Vavasis, “Time continuity in cohesive finite element modeling,” *International Journal for Numerical Methods in Engineering*, vol. 58, no. 5, pp. 679–701, 2003.

[94] D. V. Kubair and P. H. Geubelle, “Comparative analysis of extrinsic and intrinsic cohesive models of dynamic fracture,” *International Journal of Solids and Structures*, vol. 40, no. 15, pp. 3853–3868, 2003.

[95] M. Elices, “The cohesive zone model: advantages, limitations and challenges,” *Engineering Fracture Mechanics*, vol. 69, no. 2, pp. 137–163, 2001.

[96] X. Zeng and S. Li, “Comput. Methods Appl. Mech. Engrg.,” *Computer Methods in Applied Mechanics and Engineering*, vol. 199, no. 9-12, pp. 547–556, 2010.

[97] S. Krenk, “Energy conservation in Newmark based time integration algorithms,” *Computer Methods in Applied Mechanics and Engineering*, vol. 195, no. 44-47, pp. 6110–6124, 2006.

[98] A. Delaplace and A. Ibrahimbegovic, “Performance of time stepping schemes for discrete models in fracture dynamic analysis,” *International Journal for Numerical Methods in Engineering*, vol. 65, no. 9, pp. 1527–1544, 2006.

[99] D. Doyen, A. Ern, and S. Piperno, “Quasi-explicit time-integration schemes for dynamic fracture with set-valued cohesive zone models,” *Computational Mechanics*, vol. 52, no. 2, pp. 401–416, 2013.

[100] S. E. Leon, E. N. Lages, C. N. de Araújo, and G. H. Paulino, “On the effect of constraint parameters on the generalized displacement control method,” *Mechanics research communications*, vol. 56, pp. 123–129, Mar. 2014.

[101] S. E. Leon, D. W. spring, and G. H. Paulino, "Reduction in mesh bias for dynamic fracture using adaptive splitting of polygonal finite elements," *International Journal for Numerical Methods in Engineering*, vol. 100, no. 8, pp. 555–576, 2014.

[102] B. L. Boyce, "Preface to the Special Issue on the Sandia Fracture Challenge," *International Journal of Fracture*, vol. 186, no. 1-2, pp. 1–3, 2014.

[103] B. L. Boyce, S. L. B. Kramer, H. E. Fang, T. E. Cordova, M. K. Neilsen, K. Dion, A. K. Kaczmarowski, E. Karasz, L. Xue, A. J. Gross, A. Ghahremaninezhad, K. Ravi-Chandar, S. Lin, S. W. Chi, J. S. Chen, E. Yreux, M. Rüter, D. Qian, Z. Zhou, S. Bhamare, D. T. O'Connor, S. Tang, K. I. Elkhodary, J. Zhao, J. D. Hochhalter, A. R. Cerrone, A. R. Ingraffea, P. A. Wawrzynek, B. J. Carter, J. M. Emery, M. G. Veilleux, P. Yang, Y. Gan, X. Zhang, Z. Chen, E. Madenci, B. Kilic, T. Zhang, E. Fang, P. Liu, J. Lua, K. Nahshon, M. Miraglia, J. Cruce, R. DeFrese, E. T. Moyer, S. Brinckmann, L. Quinkert, K. Pack, M. Luo, and T. Wierzbicki, "The Sandia Fracture Challenge: blind round robin predictions of ductile tearing," *International Journal of Fracture*, vol. 186, no. 1-2, pp. 5–68, 2014.

[104] S. Brinckmann and L. Quinkert, "Ductile tearing: applicability of a modular approach using cohesive zones and damage mechanics," *International Journal of Fracture*, vol. 186, no. 1-2, pp. 141–154, 2013.

[105] A. J. Gross and K. Ravi-Chandar, "Prediction of ductile failure using a local strain-to-failure criterion," *International Journal of Fracture*, vol. 186, no. 1-2, pp. 1–23, 2013.

[106] K. Nahshon, M. Miraglia, J. Cruce, R. DeFrese, and E. T. Moyer, "Prediction of the Sandia Fracture Challenge using a shear modified porous plasticity model," *International Journal of Fracture*, vol. 186, no. 1-2, pp. 93–105, 2013.

[107] M. K. Neilsen, K. N. Dion, H. E. Fang, A. K. Kaczmarowski, and E. Karasz, "Ductile tearing predictions with Wellman's failure model," *International Journal of Fracture*, vol. 186, no. 1-2, pp. 107–115, 2013.

[108] K. Pack, M. Luo, and T. Wierzbicki, "Sandia Fracture Challenge: blind prediction and full calibration to enhance fracture predictability," *International Journal of Fracture*, vol. 186, no. 1-2, pp. 155–175, 2014.

[109] P. Yang, Y. Gan, X. Zhang, Z. Chen, W. Qi, and P. Liu, "Improved decohesion modeling with the material point method for simulating crack evolution," *International Journal of Fracture*, vol. 186, no. 1-2, pp. 177–184, 2014.

[110] Z. Zhou, S. Bhamare, and D. Qian, "Ductile fracture in thin sheet metals: a FEM study of the Sandia fracture challenge problem based on the Gurson–Tvergaard–Needleman fracture model," *International Journal of Fracture*, vol. 186, no. 1-2, pp. 185–200, 2014.

[111] T. Zhang, E. Fang, P. Liu, and J. Lua, "Modeling and simulation of 2012 Sandia fracture challenge problem: phantom paired shell for Abaqus and plane strain core approach," *International Journal of Fracture*, vol. 186, no. 1-2, pp. 117–139, 2013.

[112] A. Mota, W. Sun, J. T. Ostien, J. W. Foulk Iii, and K. N. Long, "Lie-group interpolation and variational recovery for internal variables," *Computational Mechanics*, vol. 52, no. 6, pp. 1281–1299, 2013.

[113] K. J. Bathe, *Finite Element Procedures.* Upper Saddle River, New Jersey: Prentice Hall, 1996.

[114] J. Bonet, *Nonlinear Continuum Mechanics for Finite Element Analysis.* New York, USA: Cambridge University Press, 1997.

[115] J. Donea, A. Huerta, J. P. Ponthot, and A. Rodríguez Ferran, "Arbitrary lagrangian–eulerian methods," Encyclopedia of computational mechanics, 2004.

[116] J. N. Reddy, *An Introduction to Nonlinear Finite Element Analysis.* New York: Oxford University Press, 2004.

[117] A. Srikanth and N. Zabaras, "An updated Lagrangian finite element sensitivity analysis of large deformations using quadrilateral elements," *International Journal for Numerical Methods in Engineering*, vol. 52, no. 10, pp. 1131–1163, 2001.

[118] C. A. Felippa, "Nonlinear finite element methods," *University of Colorado, Boulder, Colorado, USA*, 2001.

[119] A. R. Khoei and R. W. Lewis, "Adaptive finite element remeshing in a large deformation analysis of metal powder forming," *International Journal for Numerical Methods in Engineering*, vol. 45, no. 7, pp. 801–820, 1999.

[120] J. Braun and M. Sambridge, "Dynamical Lagrangian Remeshing (DLR): a new algorithm for solving large strain deformation problems and its application to fault-propagation folding," *Earth and Planetary Science Letters*, vol. 124, no. 1, pp. 211–220, 1994.

[121] Z. J. Zhang, G. H. Paulino, and W. Celes, "Extrinsic cohesive modelling of dynamic fracture and microbranching instability in brittle materials," *International Journal for Numerical Methods in Engineering*, vol. 72, no. 8, pp. 1017–1048, 2007.

[122] A. Rodríguez Ferran, F. Casadei, and A. Huerta, "ALE stress update for transient and quasistatic processes," *International Journal for Numerical Methods in Egineering*, vol. 43, no. 2, pp. 241–262, 1998.

[123] A. Rodríguez Ferran, A. P rez Foguet, and A. Huerta, "Arbitrary Lagrangian-Eulerian (ALE) formulation for hyperelastoplasticity," *International Journal for Numerical Methods in Engineering*, vol. 53, no. 8, pp. 1831–1851, 2002.

[124] F. Baaijems, "An U-ALE formulation of 3-D unsteady viscoelastic flow," *International Journal for Numerical Methods in Engineering*, vol. 36, no. 7, pp. 1115–1143, 1993.

[125] R. Radovitzky and M. Ortiz, "Error estimation and adaptive meshing in strongly nonlinear dynamic problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 172, no. 1, pp. 203–240, 1999.

[126] R. Boussetta, T. Coupez, and L. Fourment, "Adaptive remeshing based on a posteriori error estimation for forging simulation," *Computer Methods in Applied Mechanics and Engineering*, vol. 195, no. 48-49, pp. 6626–6645, 2006.

[127] A. R. Khoei and S. A. Gharehbaghi, "Three-dimensional data transfer operators in large plasticity deformations using modified-SPR technique," *Applied Mathematical Modelling*, vol. 33, no. 7, pp. 3269–3285, 2009.

[128] O. C. Zienkiewicz and J. Z. Zhu, "The superconvergent patch recovery and a posteriori error estimates. Part 1: The recovery technique," *International Journal for Numerical Methods in Engineering*, vol. 33, no. 7, pp. 1331–1364, 1992.

[129] D. Peric, "Transfer operators for evolving meshes in small strain elasto-placticity," *Computer Methods in Applied Mechanics and Engineering*, vol. 137, no. 3-4, pp. 331–344, 1996.

[130] X. Jiao and M. T. Heath, "Common-refinement-based data transfer between non-matching meshes in multiphysics simulations," *International Journal for Numerical Methods in Engineering*, vol. 61, no. 14, pp. 2402–2427, 2004.

[131] M. M. Rashid, "Material state remapping in computational solid mechanics," *International Journal for Numerical Methods in Engineering*, vol. 55, no. 4, pp. 431–450, 2002.

[132] M. Ortiz and J. J. Quigley, "Adaptive mesh refinement in strain localization problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 90, no. 1, pp. 781–804, 1991.

[133] A. Orlando and D. Peric, "Analysis of transfer procedures in elastoplasticity based on the error in the constitutive equations: Theory and numerical illustration," *International Journal for Numerical Methods in Engineering*, vol. 60, no. 9, pp. 1595–1631, 2004.

[134] A. Bucher, A. Meyer, U. J. Görke, and R. Kreißig, "A Comparison of Mapping Algorithms for Hierarchical Adaptive FEM in Finite Elasto-Plasticity," *Computational Mechanics*, vol. 39, no. 4, pp. 521–536, 2006.

[135] M. Ortiz, R. A. Radovitzky, and E. A. Repetto, "The computation of the exponential and logarithmic mappings and their first and second linearizations," *International Journal for Numerical Methods in Engineering*, vol. 52, no. 12, p. 1431, 2001.

[136] N. J. Higham, "The Scaling and Squaring Method for the Matrix Exponential Revisited," *SIAM Review*, vol. 51, no. 4, pp. 747–764, 2009.

[137] A. H. Al-Mohy and N. J. Higham, "Improved Inverse Scaling and Squaring Algorithms for the Matrix Logarithm," *SIAM Journal on Scientific Computing*, vol. 34, no. 4, pp. C153–C169, 2012.

[138] MATLAB, *version 8.0.0.783 (R2012b)*. Natick, Massachusetts: The MathWorks Inc., 2012.

[139] SIERRA Solid Mechanics Team, "Adagio 4.18 User's Guide," Tech. Rep. SAND2010-6313, 2010.

[140] M. Heroux, R. Bartlett, V. H. R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, and A. Salinger, "An overview of Trilinos," Tech. Rep. SAND2003-2927, 2003.

[141] Sandia Corporation, *Cubit User Doumentation*, 14.1 ed., 2014.

[142] L. A. Schoof and V. R. Yarberry, "EXODUS II: a finite element data model," Tech. Rep. SAND92–2137, 1994.

[143] T. D. Kostka, "Exomerge User's Manual: A lightweight Python interface for manipulating Exodus files," Tech. Rep. SAND2013-0725, 2013.

[144] A. A. Brown and D. J. Bammann, "International Journal of Plasticity," *International Journal of Plasticity*, vol. 32-33, no. C, pp. 17–35, 2012.

[145] J. E. Bishop, B. L. Boyce, T. E. Cordova, J. V. Cox, T. B. Crenshaw, K. Dion, K. J. Dowding, J. M. Emery, J. T. Foster, J. W. Foulk, D. J. Littlewood, J. T. Ostien, A. Mota, J. H. Robbins, S. A. Silling, B. W. Spencer, and G. W. Wellman, "Ductile failure X-prize," Tech. Rep. SAND2011-6801, 2011.

[146] S. H. Lo, "Generating quadrilateral elements on plane and over curved surfaces," *Computers and Structures*, vol. 31, no. 3, pp. 421–426, 1989.

[147] P. M. Knupp, "Algebraic mesh quality metrics for unstructured initial meshes," *Finite Elements in Analysis and Design*, vol. 39, no. 3, pp. 217–241, 2003.

[148] C. Talischi, G. H. Paulino, A. Pereira, and I. F. M. Menezes, "PolyMesher: a general-purpose mesh generator for polygonal elements written in Matlab," *Structural and Multidisciplinary Optimization*, vol. 45, no. 3, pp. 309–328, 2012.

[149] X.-P. Xu and A. Needleman, "Numerical simulations of fast crack growth in brittle solids," *Journal of the Mechanics and Physics of Solids*, vol. 42, no. 9, pp. 1397–1434, 1994.

[150] G. Camacho and M. Ortiz, "Computational modelling of impact damage in brittle materials," *International Journal of Solids and Structures*, vol. 33, no. 20-22, pp. 2899–2938, 1996.

[151] M. Ortiz and A. Pandolfi, "Finite-deformation irreversible cohesive elements for three-dimensional crack-propagation analysis," *International Journal for Numerical Methods in Engineering*, vol. 44, no. 9, pp. 1267–1282, 1999.

[152] A. Pandolfi and M. Ortiz, "An efficient adaptive procedure for three-dimensional fragmentation simulations," *Engineering with Computers*, vol. 18, no. 2, pp. 148–159, 2002.

[153] Z. Zhang, *Extrinsic Cohesive Modeling of Dynamic Fracture and Microbranching Instability Using A Topological Data Structure*. PhD thesis, 2007.

[154] T. Belytschko, W. Liu, and B. Moran, *Nonlinear Finite Elements for Continua and Structures*. West Sussex, England: John Wiley & Sons, Inc., 2000.

[155] N. M. Newmark, "A Method of Computation for Structural Dynamics," *Journal of the Engineering Mechanics Division*, vol. 85, no. 7, pp. 67–94, 1959.

[156] E. Hinton, T. Rock, and O. C. Zienkiewicz, "A note on mass lumping and related processes in the finite element method," *Earthquake Engineering & Structural Dynamics*, vol. 4, no. 3, pp. 245–249, 1976.

[157] T. Hughes, *The Finite Element Method*. Prentice-Hall, Inc., 1987.

[158] K. Park, G. H. Paulino, W. Celes, and R. Espinha, "Adaptive mesh refinement and coarsening for cohesive zone modeling of dynamic fracture," *International Journal for Numerical Methods in Engineering*, vol. 92, no. 1, pp. 1–35, 2012.

[159] A. Pandolfi, P. Krysl, and M. Ortiz, "Finite element simulation of ring expansion and fragmentation: the capturing of length and time scales through cohesive models of fracture," *International Journal of Fracture*, vol. 95, no. 1-4, pp. 279–297, 1999.

[160] C.-H. Sam, K. D. Papoulia, and S. A. Vavasis, "Obtaining initially rigid cohesive finite element models that are temporally convergent," *Engineering Fracture Mechanics*, vol. 72, pp. 2247–2267, Sept. 2005.

[161] I. Dooley, S. Mangala, L. Kale, and P. Geubelle, "Parallel Simulations of Dynamic Fracture Using Extrinsic Cohesive Elements," *Journal of Scientific Computing*, vol. 39, no. 1, pp. 144–165, 2008.

[162] W. Celes, G. H. Paulino, and R. Espinha, "A compact adjacency-based topological data structure for finite element mesh representation," *International Journal for Numerical Methods in Engineering*, vol. 64, no. 11, pp. 1529–1556, 2005.

[163] G. H. Paulino, W. Celes, R. Espinha, and Z. J. Zhang, "A general topology-based framework for adaptive insertion of cohesive elements in finite element meshes," *Engineering with Computers*, vol. 24, no. 1, pp. 59–78, 2008.

[164] K. Park, G. H. Paulino, and J. R. Roesler, "A unified potential-based cohesive model of mixed-mode fracture," *Journal of the Mechanics and Physics of Solids*, vol. 57, no. 6, pp. 891–908, 2009.

[165] M. J. van den Bosch, P. J. G. Schreurs, and M. G. D. Geers, "An improved description of the exponential Xu and Needleman cohesive zone law for mixed-mode decohesion," *Engineering Fracture Mechanics*, vol. 73, pp. 1220–1234, June 2006.

[166] S. Goutianos and B. F. Sørensen, "Engineering Fracture Mechanics," *Engineering Fracture Mechanics*, vol. 91, no. C, pp. 117–132, 2012.

[167] J. Bolander and N. Sukumar, "Irregular lattice model for quasistatic crack propagation," *Physical Review B*, vol. 71, no. 9, p. 094106, 2005.

[168] M. S. Ebeida and S. A. Mitchell, "Uniform random Voronoi meshes," in *Proceedings of the 20th International Meshing Roundtable*, pp. 273–290, Springer, 2012.

[169] D. O. Potyondy, P. A. Wawrzynek, and A. R. Ingraffea, "An algorithm to generate quadrilateral or triangular element surface meshes in arbitrary domains with applications to crack propagation," *International Journal for Numerical Methods in Engineering*, vol. 38, no. 16, pp. 2677–2701, 1995.

[170] M. O. Freitas, P. A. Wawrzynek, J. B. Cavalcante-Neto, C. A. Vidal, L. F. Martha, and A. R. Ingraffea, "A distributed-memory parallel technique for two-dimensional mesh generation for arbitrary domains," *Advances in Engineering Software*, vol. 59, pp. 38–52, 2013.

[171] C. Radin and L. Sadun, "The isoperimetric problem for pinwheel tilings," *Communications in mathematical physics*, vol. 177, no. 1, pp. 255–263, 1996.

[172] J. E. Bishop, "Simulating the pervasive fracture of materials and structures using randomly close packed Voronoi tessellations," *Computational Mechanics*, vol. 44, no. 4, pp. 455–471, 2009.

[173] E. L. Wachspress, *A Rational Finite Element Basis.* New York: Academic Press, 1975.

[174] N. Sukumar and A. Tabarraei, "Conforming polygonal finite elements," *International Journal for Numerical Methods in Engineering*, vol. 6, no. 12, pp. 2045–2066, 2004.

[175] C. Talischi, G. H. Paulino, A. Pereira, and I. F. M. Menezes, "PolyTop: a Matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes," *Structural and Multidisciplinary Optimization*, vol. 45, no. 3, pp. 329–357, 2012.

[176] J. J. Rimoli and J. J. Rojas, "Meshing strategies for the alleviation of mesh-induced effects in cohesive element models," *arXiv preprint arXiv:13021161*, 2013.

[177] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematlk*, vol. 1, no. 1, pp. 269–227, 1959.

[178] R. T. Rockafellar, R. J.-B. Wets, and M. Wets, *Variational analysis*, vol. 317. Springer, 1998.

[179] G. Geißler, C. Netzker, and M. Kaliske, "Discrete crack path prediction by an adaptive cohesive crack model," *Engineering Fracture Mechanics*, vol. 77, no. 18, pp. 3541–3557, 2010.

[180] T. Belytschko and M. Tabbara, "H-Adaptive finite element methods for dynamic problems, with emphasis on localization," *International Journal for Numerical Methods in Engineering*, vol. 36, no. 24, pp. 4245–4265, 1993.

[181] A. Henderson, J. Ahrens, and C. Law, The ParaView Guide. Clifton Park, NY: Kitware Inc, 2004.

[182] J. Burkardt, "VTK Files," http://people.sc.fsu.edu/ jburkardt/data/vtk/vtk.html.

[183] R. John and S. P. Shah, "Mixed-mode fracture of concrete subjected to impact loading," *Journal of Structural Engineering*, vol. 116, no. 3, pp. 585–602, 1990.

[184] D. B. Kirk and W. H. Wen-mei, *Programming massively parallel processors: a hands-on approach.* Morgan Kaufmann, 2010.

[185] A. R. Brodtkorb, T. R. Hagen, and M. L. Sætra, "Graphics processing unit (GPU) programming strategies and trends in GPU computing," *Journal of Parallel and Distributed Computing*, vol. 73, no. 1, pp. 4–13, 2013.

[186] A. Dziekonski, P. Sypek, A. Lamecki, and M. Mrozowski, "Generation of large finite-element matrices on multiple graphics processors," *International Journal for Numerical Methods in Engineering*, vol. 94, no. 2, pp. 204–220, 2012.

[187] C. Cecka, A. J. Lew, and E. Darve, "Assembly of finite element methods on graphics processors," *International Journal for Numerical Methods in Engineering*, vol. 85, no. 5, pp. 640–669, 2010.

[188] L. Wang, Y. S. Zhang, B. Zhu, C. Xu, X. W. Tian, C. Wang, J. H. Mo, and J. Li, "GPU Accelerated Parallel Cholesky Factorization," *Applied Mechanics and Materials*, vol. 148-149, pp. 1370–1373, 2012.

[189] O. S. Lawlor, S. Chakravorty, T. L. Wilmarth, N. Choudhury, I. Dooley, G. Zheng, and L. V. Kalé, "ParFUM: a parallel framework for unstructured meshes for scalable dynamic physics applications," *Engineering with Computers*, vol. 22, no. 3-4, pp. 215–235, 2006.

[190] R. Espinha, K. Park, G. H. Paulino, and W. Celes, "Scalable parallel dynamic fracture simulation using an extrinsic cohesive zone model ," *Computer Methods in Applied Mechanics and Engineering*, vol. 266, no. C, pp. 144–161, 2013.

[191] S. Park and H. Shin, "Efficient generation of adaptive Cartesian mesh for computational fluid dynamics using GPU," *International Journal for Numerical Methods in Fluids*, vol. 70, pp. 1393–1404, Jan. 2012.

[192] A. Alhadeff, W. Celes, and G. H. Paulino, "Mapping Cohesive Fracture and Fragmentation Simulations to GPUs," *International Journal for Numerical Methods in Engineering*, vol. In press, 2015.

[193] D. S. Boyalakuntla and J. Y. Murthy, "Hierarchical compact models for simulation of electronic chip packages," *Components and Packaging Technologies, IEEE Transactions on*, vol. 25, no. 2, pp. 192–203, 2002.

[194] F. Ducros, V. Ferrand, F. Nicoud, C. Weber, D. Darracq, C. Gacherieu, and T. Poinsot, "Large-eddy simulation of the shock/turbulence interaction," *Journal of Computational Physics*, vol. 152, no. 2, pp. 517–549, 1999.

[195] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo, "FLASH: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes," *The Astrophysical Journal Supplement Series*, vol. 131, no. 1, p. 273, 2000.

[196] M. W. Beall and M. S. Shephard, "A general topology-based mesh data structure," *International Journal for Numerical Methods in Engineering*, vol. 40, no. 9, pp. 1573–1596, 1997.

[197] R. V. Garimella, "Mesh data structure selection for mesh generation and FEA applications," *International Journal for Numerical Methods in Engineering*, vol. 55, no. 4, pp. 451–478, 2002.

[198] J.-F. o. Remacle and M. S. Shephard, "An algorithm oriented mesh database," *International Journal for Numerical Methods in Engineering*, vol. 58, no. 2, pp. 349–374, 2003.

[199] Y. Zhou, W. Xu, B. R. Donald, and J. Zeng, "An efficient parallel algorithm for accelerating computational protein design.," *Bioinformatics*, vol. 30, no. 12, pp. i255–i263, 2014.

[200] M. Joselli, J. R. da S Junior, E. W. Clua, A. Montenegro, M. Lage, and P. Pagliosa, "Neighborhood grid: A novel data structure for fluids animation with GPU computing ," *J. Parallel Distrib. Comput.*, vol. 75, pp. 20–28, 2015.

[201] D. J. Welsh and M. B. Powell, "An upper bound for the chromatic number of a graph and its application to timetabling problems," *The Computer Journal*, vol. 10, no. 1, pp. 85–86, 1967.

[202] L. Velho and J. Gomes, "Variable Resolution 4-k Meshes: Concepts and Applications," *Computer Graphics Forum*, vol. 19, no. 4, pp. 195–212, 2000.

[203] E. Sharon and J. Fineberg, "Microbranching instability and the dynamic fracture of brittle materials," *Physical Review B - Condensed Matter and Materials Physics*, vol. 54, no. 10, pp. 7128–7139, 1996.

[204] O. Miller, L. B. Freund, and A. Needleman, "Energy dissipation in dynamic fracture of brittle materials," *Modelling and Simulation in Materials Science and Engineering*, vol. 7, no. 4, p. 573, 1999.

[205] L. Roy Xu, Y. Y Huang, and A. J. Rosakis, "Dynamic crack deflection and penetration at interfaces in homogeneous materials: experimental studies and model predictions," *Journal of the Mechanics and Physics of Solids*, vol. 51, pp. 461–486, Nov. 2002.

[206] R. Peerlings, R. de borst, W. Brekelmans, and M. Geers, "Localisation issues in local and nonlocal continuum approaches to fracture," *European Journal of Mechanics: A/Solids*, vol. 21, no. 2, pp. 175–189, 2002.

[207] J. Oliver, A. E. Huespe, M. D. G. Pulido, and E. Samaniego, "On the strong discontinuity approach in finite deformation settings," *International Journal for Numerical Methods in Engineering*, vol. 56, no. 7, pp. 1051–1082, 2003.

[208] R. Rangarajan and A. J. Lew, “Universal meshes: A method for triangulating planar curved domains immersed in nonconforming meshes,” *International Journal for Numerical Methods in Engineering*, vol. 98, no. 4, pp. 236–264, 2014.

[209] R. Rangarajan, M. M. Chiaramonte, M. J. Hunsweck, Y. Shen, and A. J. Lew, “Simulating curvilinear crack propagation in two dimensions with universal meshes,” *International Journal for Numerical Methods in Engineering*, vol. In press, 2015.

[210] C. Araujo and W. Celes, “Quadrilateral Mesh Generation with Deferred Constraint Insertion,” *Procedia Engineering*, vol. In press, 2014.

[211] L. Velho, “Quadrilateral meshing using 4-8 clustering,” in *CILANCE 2000*, pp. 61–64, 2000.

[212] R. El Khaoulani and P. O. Bouchard, “An anisotropic mesh adaptation strategy for damage and failure in ductile materials,” *Finite Elements in Analysis and Design*, vol. 59, pp. 1–10, 2012.

[213] A. R. Khoei, H. Azadi, and H. Moslemi, “Modeling of crack propagation via an automatic adaptive mesh refinement based on modified superconvergent patch recovery technique,” *Engineering Fracture Mechanics*, vol. 75, no. 10, pp. 2921–2945, 2008.

[214] A. R. Khoei, H. Moslemi, and M. Sharifi, “Three-dimensional cohesive fracture modeling of non-planar crack growth using adaptive FE technique,” *International Journal of Solids and Structures*, vol. 49, no. 17, pp. 2334–2348, 2012.

[215] A. R. Khoei, M. Eghbalian, H. Moslemi, and H. Azadi, “Crack growth modeling via 3D automatic adaptive mesh refinement based on modified-SPR technique,” *Applied Mathematical Modelling*, vol. 37, pp. 357–383, Jan. 2013.

[216] K. Murotani, G. Yagawa, and J. B. Choi, “Adaptive finite elements using hierarchical mesh and its application to crack propagation analysis,” *Computer Methods in Applied Mechanics and Engineering*, vol. 253, no. C, pp. 1–14, 2013.

[217] C. Miehe and E. Gürses, “A robust algorithm for configurational-force-driven brittle crack propagation with R-adaptive mesh alignment,” *International Journal for Numerical Methods in Engineering*, vol. 72, no. 2, pp. 127–155, 2007.

[218] M. Floater, A. Gillette, and N. Sukumar, “Gradient bounds for Wachspress coordinates on polytopes,” *arXiv preprint arXiv:1306.4385v2*, June 2013.

[219] M. S. Floater, G. Kós, and M. Reimers, “Mean value coordinates in 3D,” *Computer Aided Geometric Design*, vol. 22, pp. 623–631, Oct. 2005.

[220] M. S. Ebeida, S. A. Mitchell, A. Patney, A. A. Davidson, and J. D. Owens, “A Simple Algorithm for Maximal Poisson-Disk Sampling in High Dimensions,” *Computer Graphics Forum*, vol. 31, no. 2pt4, pp. 785–794, 2012.

[221] M. M. Rashid and M. Selimotic, “A three-dimensional finite element method with arbitrary polyhedral elements,” *International Journal for Numerical Methods in Engineering*, vol. 67, no. 2, pp. 226–252, 2006.

[222] A. L. Gain, C. Talischi, and G. H. Paulino, “On the Virtual Element Method for Three-Dimensional Elasticity Problems on Arbitrary Polyhedral Meshes,” *Computer Methods in Applied Mechanics and Engineering*, p. In press.

[223] L. Beirão da Veiga, F. Brezzi, A. Cangiani, G. Manzini, L. D. Marini, and A. Russo, “Basic principles of virtual element methods,” *Mathematical Models and Methods in Applied Sciences*, vol. 23, no. 01, pp. 199–214, 2013.

[224] D. Boffi, F. Brezzi, and M. Fortin, *Mixed Finite Element Methods and Applications*, vol. 44 of *Springer Series in Computational Mathematics*. Springer, 2013.

[225] R. Courant, K. Friedrichs, and H. Lewy, "On the partial difference equations of mathematical physics," *Ibm Journal of Research and Development*, vol. 11, no. 2, pp. 215–234, 1967.

[226] T. Belytschko, H. Yen, and R. Mullen, "Mixed methods for time integration," *Computer Methods in Applied Mechanics and Engineering*, vol. 17, pp. 259–275, 1979.

[227] A. Prakash and K. D. Hjelmstad, "A FETI-based multi-time-step coupling method for Newmark schemes in structural dynamics," *International Journal for Numerical Methods in Engineering*, vol. 61, no. 13, pp. 2183–2204, 2004.

[228] A. Combescure and A. Gravouil, "A numerical scheme to couple subdomains with different time-steps for predominantly linear transient analysis," *Computer Methods in Applied Mechanics and Engineering*, vol. 191, no. 11, pp. 1129–1157, 2002.

Appendix A

Scripted procedures for remeshing and mapping internal state variables

The procedure to map internal state variables was scripted using python. All of the analysis modules are called from a single script that is easily executed by the end user. Two versions of the script are included here for reference. First, the script to map internal state variables from the source mesh to the target mesh, where the target mesh is the same as the source mesh is presented in Section A.1. Next, the script to map internal state variables from the source mesh to a new target mesh is presented in Section A.2.Additionally, an example of one of the post-processing scripts is included in Section A.3.

A.1 Python script for mapping internal state variables without remeshing

```
1  from exodus import exodus
2  import exomerge
3  import os
4  import io
5  import sys
6
7  # global variables - user inputs (may be edited here or in command line)
8  adagio = '/localhome/sleon/sierra/bin/adagio'
9  sierra_in = 'sierra_in.i'
10 template = 'template.i'
11 init_mesh = 'hexes_fine.g'
12 nintervals = 10
13 nremaps = 1
14 etime = 0.25
15 dt = 0.0025
16 nproc = 20
17
18 # support functions
19 def renameFields(filename) :
20     #print 'renaming file ' + filename
21     model = exomerge.import_model(filename)
22
23     nGPs = 8
24     endings = ['xx', 'xy', 'yy', 'yz', 'zx', 'zz']
25     for i in range(len(endings)) :
```

```
26          for j in range(nGPs) :
27              old_stretch = 'exp_log_left_stretch_' + endings[i] + '_' + str(j+1)
28              new_stretch = 'left_stretch_' + endings[i] + '_' + str(j+1)
29              model.rename_element_field(old_stretch, new_stretch)
30
31              remove_stretch = 'log_left_stretch_' + endings[i] + '_' + str(j+1)
32              model.delete_element_field(remove_stretch)
33
34          endings = ['xx', 'xy', 'xz', 'yx', 'yy', 'yz', 'zx','zy', 'zz']
35          for i in range(len(endings)) :
36              for j in range(nGPs) :
37                  old_rotation = 'exp_log_rotation_' + endings[i] + '_' + str(j+1)
38                  new_rotation = 'rotation_' + endings[i] + '_' + str(j+1)
39                  model.rename_element_field(old_rotation, new_rotation)
40                  remove_rotation = 'log_rotation_' + endings[i] + '_' + str(j+1)
41                  model.delete_element_field(remove_rotation)
42
43          model.export(filename)
44
45  def nameFile (type, nAnalysis, nRemap) :
46      if (nRemap == 0) :
47          filename = type + '.' + str(nAnalysis)
48      else :
49          filename = type + '_remap' + str(nRemap) + '.' + str(nAnalysis)
50      return filename
51
52  def copyFile (copyfrom, copyto) :
53      os.system('cp ' + copyfrom + ' ' + copyto)
54
55  def modifyInputFile_DB (filename, exo_in, exo_out, sm_dat) :
56      old_text = ['Database name = {exo_in}', 'stream name = {sm_dat}',\
57                  'Database name = {exo_out}']
58      new_text = ['    Database name = ' + exo_in + '\n', \
59                  '           stream name = ' + sm_dat + '\n', \
60                  '    Database name = ' + exo_out + '\n']
61      editLine(filename, old_text, new_text, len(old_text))
62
63  def editLine (filename, old_text, new_text, nreplace) :
64      with open(filename, 'r') as file:
65          data = file.readlines()
66
67      j = 0
68      for i in range(len(data)) :
69          if old_text[j].lower() in data[i].lower() :
70              data[i] = new_text[j]
71              j+=1
72              if j == nreplace:
```

```
73                  break
74              if j != nreplace:
75                  print sierra_in + ' not modified correctly'
76                  exit(1)
77
78      with open(filename, 'w') as file:
79          file.writelines(data)
80
81  def callSierra(nproc, sm_in, sm_log, start_time, end_time, dt, read_time, do_proj,
82                  do_init, reg_step, equilibrium_step):
83      sm_run = 'sierra -j ' + str(nproc) + ' ' + adagio +' -i ' + sm_in + \
84                ' --logfile ' + sm_log + ' --aprepro --define ' + \
85                '"start_time=' + str(start_time) + ' end_time=' +
86                str(end_time) + ' dt=' + str(dt) + ' read_time=' + \
87                str(read_time) + ' projection=' + str(do_proj) + ' initialize=' + \
88                str(do_init) + ' reg_step=' + str(reg_step) + \
89                ' equilibrium_step=' + str(equilibrium_step) + '"'
90      cmdLine(sm_run, 'sierra_'+str(nrun) +'_out')
91
92  def callLogVars (exo_in, exo_out, sm_in, exp_flag = 0) :
93      if exp_flag == 0 :
94          log_run = './logVars -source ' + exo_in + ' -out ' + \
95                    exo_out + ' -cmd ' + sm_in
96      else :
97          log_run = './logVars -exp -source ' + exo_in + ' -out ' + \
98                    exo_out + ' -cmd ' + sm_in
99      cmdLine(log_run, 'logVars_out')
100     return exo_out
101
102 def callPushForward (exo_in, exo_out) :
103     push_run = './pushforwardMesh -source_e ' + exo_in + ' -target_e ' + exo_out
104     cmdLine(push_run, 'pushforward_out')
105     return exo_out
106
107 def callInterpolate (exo_in, target, exo_out, sm_in) :
108     interp_run = './interpolateVars -source_e ' + exo_in + \
109                  ' -target_g ' + target  + ' -target_e ' + exo_out + \
110                  ' -source_i ' +  sm_in +  ' -target_i ' +  sm_in
111     cmdLine (interp_run, 'interpolate_out')
112     return exo_out
113
114 def callPullBack (exo_in, exo_out) :
115     pull_run = './pullbackMesh -source_e ' + exo_in + ' -target_e ' + exo_out
116     cmdLine (pull_run, 'pullback_out')
117     return exo_out
118
119 def cmdLine (str, filename) :
```

```
120     print str + '\n'
121     os.system(str + ' >>console.txt')
122
123 # main script if __name__ == "__main__":
124     # read inputs
125     if len(sys.argv) != 6 :
126         print 'usage python auto_remap.py [num intervals] ' + \
127                 '[num remaps] [end time] [time step] [num processors]'
128         try :
129             user_in = input ('use default values? (1 = yes, 0 = no)' + \
130                     ' [num intervals = ' + str(nintervals) + '] ' + \
131                     ' [num remaps = ' + str(nremaps) + ']' + \
132                     ' [end time = ' + str(etime) + ']'  + \
133                     ' [time step = ' + str(dt) + ']' + \
134                     ' [num processors = ' +  str(nproc) + ']' + '\n')
135         except :
136             print 'Invalid input. Rerun and specify input values,
137                     or rerun and accept default values.'
138             exit(1)
139
140         try:
141             use_default = int(user_in)
142         except :
143             print 'Invalid input. Rerun and specify input values,
144             or rerun and accept default values.'
145             exit(1)
146
147         if use_default != 1 :
148             print 'exiting auto_remap, restart with correct user inputs'
149             exit(1)
150     else :
151         nintervals = int(sys.argv[1])
152         nremaps = int(sys.argv[2])
153         etime = float(sys.argv[3])
154         dt = float(sys.argv[4])
155         nproc = int(sys.argv[5])
156
157     int_time = etime/nintervals # length of each interval
158     nrun = 0
159     nremap = 0
160     tremap = 0
161
162     for i in range(nintervals-1):
163         nrun = i+1;
164
165         # run analysis from t = st to et
166         print '\nAnalsyis ' + str(nrun) + '\n---------------------------'
```

```
167         if i==0:
168             exo_in = init_mesh
169             do_init = 0
170         else:
171             exo_in = exo_out
172             do_init = 1
173
174         int_start = i*int_time;
175         int_end = (i+1)*int_time
176         read_time = int_start
177         filename = 'analysis.' + str(nrun)
178         exo_out = filename + '.e'
179         sm_log  = filename + '.log'
180         sm_dat  = filename + '.dat'
181         do_proj = 0
182         reg_step = 1
183         equilibrium_step = 0
184         copyFile (template, sierra_in)
185         modifyInputFile_DB (sierra_in, exo_in, exo_out, sm_dat)
186         callSierra(nproc, sierra_in, sm_log, int_start, int_end, dt,
187                    read_time, do_proj, do_init, reg_step, equilibrium_step)
188
189         # Remap nremaps times
190         for j in range(nremaps):
191             nrun = i+1
192             nremap = j+1
193             tremap += 1
194
195             print '\nRemap ' + str(nremap) + ' on Analysis ' + \
196                   str(nrun) + '\n----------------------------'
197
198             # perform logaritmic map on flagged variables
199             exo_in = exo_out
200             exo_out = nameFile('log', nrun, nremap) + '.e'
201             exo_out = callLogVars (exo_in, exo_out, sierra_in)
202
203             # project element variables to nodes
204             int_start = int_end-dt;
205             read_time = int_end
206             exo_in  = exo_out
207             filename = nameFile('projection', nrun, nremap)
208             exo_out = filename + '.e'
209             sm_log = filename + '.log'
210             sm_dat = filename + '.dat'
211             do_proj = 1
212             do_init = 1
213             reg_step = 0
```

```
214                 equilibrium_step = 0
215                 copyFile (template, sierra_in)
216                 modifyInputFile_DB (sierra_in, exo_in, exo_out, sm_dat)
217                 callSierra(nproc, sierra_in, sm_log, int_start, int_end, dt,
218                            read_time, do_proj, do_init, reg_step, equilibrium_step)
219
220                 # push forward to current configuration
221                 exo_in = exo_out
222                 exo_out = nameFile('pushforward', nrun, nremap) + '.e'
223                 exo_out = callPushForward(exo_in, exo_out)
224
225                 # interpolate nodal variables from old mesh to new mesh
226                 exo_in = exo_out
227                 target = exo_out # with no remesh
228                 exo_out = nameFile('interpolation', nrun, nremap) + '.e'
229
230                 # pull back to reference configuration
231                 exo_in = exo_out
232                 exo_out = nameFile('pullback', nrun, nremap) + '.e'
233                 exo_out = callPullBack (exo_in, exo_out)
234
235                 # perform exponential map on flagged variables
236                 exo_in = exo_out
237                 exo_out = nameFile('exponentiation', nrun, nremap) + '.e'
238                 exo_out = callLogVars(exo_in, exo_out, sierra_in, 1)
239
240                 # rename variables if another remap is next
241                 renameFields(exo_out)
242
243                 # zero velocity step to bring back in to equilibrium
244                 if j!=nremaps-1 :
245                     print 'equilibrium step after remapping'
246                     int_start = int_end-dt
247                     read_time = int_end
248                     exo_in  = exo_out
249                     filename = nameFile('equilibrium', nrun, nremap)
250                     exo_out = filename + '.e'
251                     sm_log = filename + '.log'
252                     sm_dat = filename + '.dat'
253                     do_proj = 0
254                     do_init = 1
255                     reg_step = 0
256                     equilibrium_step = 1
257                     copyFile (template, sierra_in)
258                     modifyInputFile_DB (sierra_in, exo_in, exo_out, sm_dat)
259                     callSierra(nproc, sierra_in, sm_log, int_start, int_end, dt,
260                                read_time, do_proj, do_init, reg_step, equilibrium_step)
```

```
261
262         # perform zero velocity step to come back into equilibrium
263         print 'equilibrium step'
264         int_start = int_end-dt
265         read_time = int_end
266         exo_in  = exo_out
267         filename = nameFile('equilibrium', nrun, nremap)
268         exo_out = filename + '.e'
269         sm_log = filename + '.log'
270         sm_dat = filename + '.dat'
271         do_proj = 0
272         do_init = 1
273         reg_step = 0
274         equilibrium_step = 1
275         copyFile (template, sierra_in)
276         modifyInputFile_DB (sierra_in, exo_in, exo_out, sm_dat)
277         callSierra(nproc, sierra_in, sm_log, int_start, int_end, dt, read_time,
278                    do_proj, do_init, reg_step, equilibrium_step)
279
280     # run analysis for last time
281      if nrun==0 :
282         int_start = 0
283         int_end = etime
284         nrun = 1
285         exo_in = init_mesh
286         do_init = 0
287      else :
288         int_start = (i+1)*int_time;
289         int_end = (i+2)*int_time
290         nrun = i + 2
291         exo_in = exo_out
292         do_init = 1
293      read_time = int_start
294      print '\nAnalsyis ' + str(nrun) + '\n---------------------------'
295      filename = 'analysis.' + str(nrun)
296      exo_out = filename + '.e'
297      sm_log  = filename + '.log'
298      sm_dat  = filename + '.dat'
299      do_proj = 0
300      reg_step = 1
301      equilibrium_step = 0
302      copyFile (template, sierra_in)
303      modifyInputFile_DB (sierra_in, exo_in, exo_out, sm_dat)
304      callSierra(nproc, sierra_in, sm_log, int_start, int_end, dt,
305                 read_time, do_proj, do_init, reg_step, equilibrium_step)
```

A.2 Python script for mapping internal state variables with remeshing

```
1  from exodus import exodus
2  import exomerge
3  import os
4  import io
5  import sys
6
7  # global variables - user inputs (may be edited here or in command line)
8  adagio = '/localhome/sleon/sierra/bin/adagio'
9  cubit_cmd = 'cubit_new 13.1_64'
10 sierra_in = 'sierra_in.i'
11 template = 'template.i'
12 cubit_jou = 'remesh.jou'
13 cubit_template = 'cubit_template.jou'
14 init_mesh = 'hexes_fine.g'
15 nintervals = 10
16 nremaps = 1
17 etime = 0.25
18 dt = 0.0025
19 nproc = 20
20
21 # support functions
22 def renameFields(filename) :
23     #print 'renaming file ' + filename
24     model = exomerge.import_model(filename)
25
26     nGPs = 8
27     endings = ['xx', 'xy', 'yy', 'yz', 'zx', 'zz']
28     for i in range(len(endings)) :
29         for j in range(nGPs) :
30             old_stretch = 'exp_log_left_stretch_' + endings[i] + '_' + str(j+1)
31             new_stretch = 'left_stretch_' + endings[i] + '_' + str(j+1)
32             model.rename_element_field(old_stretch, new_stretch)
33
34             remove_stretch = 'log_left_stretch_' + endings[i] + '_' + str(j+1)
35             model.delete_element_field(remove_stretch)
36
37         endings = ['xx', 'xy', 'xz', 'yx', 'yy', 'yz', 'zx','zy', 'zz']
38         for i in range(len(endings)) :
39             for j in range(nGPs) :
40                 old_rotation = 'exp_log_rotation_' + endings[i] + '_' + str(j+1)
41                 new_rotation = 'rotation_' + endings[i] + '_' + str(j+1)
42                 model.rename_element_field(old_rotation, new_rotation)
43                 remove_rotation = 'log_rotation_' + endings[i] + '_' + str(j+1)
```

```
44                  model.delete_element_field(remove_rotation)
45
46          model.export(filename)
47
48  def nameFile (type, nAnalysis, nRemap) :
49      if (nRemap == 0) :
50          filename = type + '.' + str(nAnalysis)
51      else :
52          filename = type + '_remap' + str(nRemap) + '.' + str(nAnalysis)
53      return filename
54
55  def copyFile (copyfrom, copyto) :
56      os.system('cp ' + copyfrom + ' ' + copyto)
57
58  def modifyInputFile_DB (filename, exo_in, exo_out, sm_dat) :
59      old_text = ['Database name = {exo_in}', 'stream name = {sm_dat}',\
60                  'Database name = {exo_out}']
61      new_text = ['    Database name = ' + exo_in + '\n', \
62                  '              stream name = ' + sm_dat + '\n', \
63                  '    Database name = ' + exo_out + '\n']
64      editLine(filename, old_text, new_text, len(old_text))
65
66  def editLine (filename, old_text, new_text, nreplace) :
67      with open(filename, 'r') as file:
68          data = file.readlines()
69
70      j = 0
71      for i in range(len(data)) :
72          if old_text[j].lower() in data[i].lower() :
73              data[i] = new_text[j]
74              j+=1
75              if j == nreplace:
76                  break
77              if j != nreplace:
78                  print sierra_in + ' not modified correctly'
79                  exit(1)
80
81      with open(filename, 'w') as file:
82          file.writelines(data)
83
84  def callSierra(nproc, sm_in, sm_log, start_time, end_time, dt, read_time, do_proj,
85                  do_init, reg_step, equilibrium_step):
86      sm_run = 'sierra -j ' + str(nproc) + ' ' + adagio +' -i ' + sm_in + \
87               ' --logfile ' + sm_log + ' --aprepro --define ' + \
88               '"start_time=' + str(start_time) + ' end_time=' +
89               str(end_time) + ' dt=' + str(dt) + ' read_time=' + \
90               str(read_time) + ' projection=' + str(do_proj) + ' initialize=' + \
```

```
91              str(do_init) + ' reg_step=' + str(reg_step) + \
92              ' equilibrium_step=' + str(equilibrium_step) + '"'
93      cmdLine(sm_run, 'sierra_'+str(nrun) +'_out')
94
95  def callCubit (cubit_in) :
96      cubit_run = cubit_cmd + ' -nographics -nojournal ' + cubit_in
97      cmdLine (cubit_run)
98
99  def callLogVars (exo_in, exo_out, sm_in, exp_flag = 0) :
100     if exp_flag == 0 :
101         log_run = './logVars -source ' + exo_in + ' -out ' + \
102                   exo_out + ' -cmd ' + sm_in
103     else :
104         log_run = './logVars -exp -source ' + exo_in + ' -out ' + \
105                   exo_out + ' -cmd ' + sm_in
106     cmdLine(log_run, 'logVars_out')
107     return exo_out
108
109 def callPushForward (exo_in, exo_out) :
110     push_run = './pushforwardMesh -source_e ' + exo_in + ' -target_e ' + exo_out
111     cmdLine(push_run, 'pushforward_out')
112     return exo_out
113
114 def callInterpolate (exo_in, target, exo_out, sm_in) :
115     interp_run = './interpolateVars -source_e ' + exo_in + \
116                  ' -target_g ' + target  + ' -target_e ' + exo_out + \
117                  ' -source_i ' +  sm_in +  ' -target_i ' +  sm_in
118     cmdLine (interp_run, 'interpolate_out')
119     return exo_out
120
121 def callPullBack (exo_in, exo_out) :
122     pull_run = './pullbackMesh -source_e ' + exo_in + ' -target_e ' + exo_out
123     cmdLine (pull_run, 'pullback_out')
124     return exo_out
125
126 def cmdLine (str, filename) :
127     print str + '\n'
128     os.system(str + ' >>console.txt')
129
130 # main script if __name__ == "__main__":
131     # read inputs
132     if len(sys.argv) != 6 :
133         print 'usage python auto_remap.py [num intervals] ' + \
134               '[num remaps] [end time] [time step] [num processors]'
135         try :
136             user_in = input ('use default values? (1 = yes, 0 = no)' + \
137                       ' [num intervals = ' + str(nintervals) + '] ' + \
```

```
138                     ' [num remaps = ' + str(nremaps) + ']' + \
139                     ' [end time = ' + str(etime) + ']'  + \
140                     ' [time step = ' + str(dt) + ']' + \
141                     ' [num processors = ' +  str(nproc) + ']' + '\n')
142         except :
143             print 'Invalid input. Rerun and specify input values,
144                     or rerun and accept default values.'
145             exit(1)
146
147         try:
148             use_default = int(user_in)
149         except :
150             print 'Invalid input. Rerun and specify input values,
151             or rerun and accept default values.'
152             exit(1)
153
154         if use_default != 1 :
155             print 'exiting auto_remap, restart with correct user inputs'
156             exit(1)
157     else :
158         nintervals = int(sys.argv[1])
159         nremaps = int(sys.argv[2])
160         etime = float(sys.argv[3])
161         dt = float(sys.argv[4])
162         nproc = int(sys.argv[5])
163
164     int_time = etime/nintervals # length of each interval
165     nrun = 0
166     nremap = 0
167     tremap = 0
168
169     for i in range(nintervals-1):
170         nrun = i+1;
171
172         # run analysis from t = st to et
173         print '\nAnalsyis ' + str(nrun) + '\n----------------------------'
174         if i==0:
175             exo_in = init_mesh
176             do_init = 0
177         else:
178             exo_in = exo_out
179             do_init = 1
180
181         int_start = i*int_time;
182         int_end = (i+1)*int_time
183         read_time = int_start
184         filename = 'analysis.' + str(nrun)
```

```
185         exo_out = filename + '.e'
186         sm_log  = filename + '.log'
187         sm_dat  = filename + '.dat'
188         do_proj = 0
189         reg_step = 1
190         equilibrium_step = 0
191         copyFile (template, sierra_in)
192         modifyInputFile_DB (sierra_in, exo_in, exo_out, sm_dat)
193         callSierra(nproc, sierra_in, sm_log, int_start, int_end, dt,
194                    read_time, do_proj, do_init, reg_step, equilibrium_step)
195 
196         # Remap nremaps times
197         for j in range(nremaps):
198             nrun = i+1
199             nremap = j+1
200             tremap += 1
201 
202             print '\nRemap ' + str(nremap) + ' on Analysis ' + \
203                   str(nrun) + '\n---------------------------'
204 
205             # perform logaritmic map on flagged variables
206             exo_in = exo_out
207             exo_out = nameFile('log', nrun, nremap) + '.e'
208             exo_out = callLogVars (exo_in, exo_out, sierra_in)
209 
210             # project element variables to nodes
211             int_start = int_end-dt;
212             read_time = int_end
213             exo_in  = exo_out
214             filename = nameFile('projection', nrun, nremap)
215             exo_out = filename + '.e'
216             sm_log = filename + '.log'
217             sm_dat = filename + '.dat'
218             do_proj = 1
219             do_init = 1
220             reg_step = 0
221             equilibrium_step = 0
222             copyFile (template, sierra_in)
223             modifyInputFile_DB (sierra_in, exo_in, exo_out, sm_dat)
224             callSierra(nproc, sierra_in, sm_log, int_start, int_end, dt,
225                        read_time, do_proj, do_init, reg_step, equilibrium_step)
226 
227             # push forward to current configuration
228             exo_in = exo_out
229             exo_out = nameFile('pushforward', nrun, nremap) + '.e'
230             exo_out = callPushForward(exo_in, exo_out)
231 
```

```
232             # remesh
233             copyFile(cubit_template, cubit_jou)
234             filename = nameFile('remesh', nrun, nremap)
235             new_mesh = filename + '.g'
236             modifyJournalFile (cubit_jou, nremap, nrun, new_mesh)
237             callCubit (cubit_jou)
238
239             # interpolate nodal variables from old mesh to new mesh
240             exo_in = exo_out
241             target = new_mesh
242             exo_out = nameFile('interpolation', nrun, nremap) + '.e'
243             exo_out = callInterpolate (exo_in, target, exo_out, sierra_in)
244
245             # pull back to reference configuration
246             exo_in = exo_out
247             exo_out = nameFile('pullback', nrun, nremap) + '.e'
248             exo_out = callPullBack (exo_in, exo_out)
249
250             # perform exponential map on flagged variables
251             exo_in = exo_out
252             exo_out = nameFile('exponentiation', nrun, nremap) + '.e'
253             exo_out = callLogVars(exo_in, exo_out, sierra_in, 1)
254
255             # rename variables if another remap is next
256             renameFields(exo_out)
257
258             # zero velocity step to bring back in to equilibrium
259             if j!=nremaps-1 :
260                 print 'equilibrium step after remapping'
261                 int_start = int_end-dt
262                 read_time = int_end
263                 exo_in  = exo_out
264                 filename = nameFile('equilibrium', nrun, nremap)
265                 exo_out = filename + '.e'
266                 sm_log = filename + '.log'
267                 sm_dat = filename + '.dat'
268                 do_proj = 0
269                 do_init = 1
270                 reg_step = 0
271                 equilibrium_step = 1
272                 copyFile (template, sierra_in)
273                 modifyInputFile_DB (sierra_in, exo_in, exo_out, sm_dat)
274                 callSierra(nproc, sierra_in, sm_log, int_start, int_end, dt,
275                         read_time, do_proj, do_init, reg_step, equilibrium_step)
276
277         # perform zero velocity step to come back into equilibrium
278         print 'equilibrium step'
```

```
279         int_start = int_end-dt
280         read_time = int_end
281         exo_in  = exo_out
282         filename = nameFile('equilibrium', nrun, nremap)
283         exo_out = filename + '.e'
284         sm_log = filename + '.log'
285         sm_dat = filename + '.dat'
286         do_proj = 0
287         do_init = 1
288         reg_step = 0
289         equilibrium_step = 1
290         copyFile (template, sierra_in)
291         modifyInputFile_DB (sierra_in, exo_in, exo_out, sm_dat)
292         callSierra(nproc, sierra_in, sm_log, int_start, int_end, dt, read_time,
293                    do_proj, do_init, reg_step, equilibrium_step)
294
295    # run analysis for last time
296     if nrun==0 :
297         int_start = 0
298         int_end = etime
299         nrun = 1
300         exo_in = init_mesh
301         do_init = 0
302     else :
303         int_start = (i+1)*int_time;
304         int_end = (i+2)*int_time
305         nrun = i + 2
306         exo_in = exo_out
307         do_init = 1
308     read_time = int_start
309     print '\nAnalsyis ' + str(nrun) + '\n---------------------------'
310     filename = 'analysis.' + str(nrun)
311     exo_out = filename + '.e'
312     sm_log  = filename + '.log'
313     sm_dat  = filename + '.dat'
314     do_proj = 0
315     reg_step = 1
316     equilibrium_step = 0
317     copyFile (template, sierra_in)
318     modifyInputFile_DB (sierra_in, exo_in, exo_out, sm_dat)
319     callSierra(nproc, sierra_in, sm_log, int_start, int_end, dt,
320                read_time, do_proj, do_init, reg_step, equilibrium_step)
```

A.3 Python script to diff two Exodus II files

```
1  from exodus
2  import exodus
3  import exomerge
4  import numpy
5
6  file1 = 'analysis.1.e'
7  file2 = 'equilibrium.1.e'
8  newfile = 'diff.e'
9
10 # copy an old model to a new model in exomerge
11 newmodel = exomerge.import_model(file1, timesteps='last')
12
13 # rename all element fields
14 elem_fields = newmodel.get_element_field_names()
15 for i in range(len(elem_fields)) :
16     newmodel.rename_element_field(elem_fields[i], 'd' + elem_fields[i])
17
18 # rename all node fields
19 node_fields = newmodel.get_node_field_names()
20 for i in range(len(node_fields)) :
21     newmodel.rename_node_field(node_fields[i], 'd' + node_fields[i])
22
23 # add displacement so paraview can read it
24 newmodel.create_node_field('displacement_x')
25 newmodel.create_node_field('displacement_y')
26 newmodel.create_node_field('displacement_z')
27
28 # rename all global fields
29 global_fields = newmodel.get_global_variable_names()
30 for i in range(len(global_fields)) :
31     newmodel.rename_global_variable(global_fields[i], 'd' + global_fields[i])
32
33 # save new model
34 newmodel.export_model(filename=newfile)
35
36 # import models into exodus
37 model1 = exodus(file1, mode='r')
38 model2 = exodus(file2, mode='r')
39 newmodel = exodus(newfile, mode='a')
40
41 # get last times
42 last1 = model1.num_times()
43 last2 = model2.num_times()
44
45 # diff node variables
```

```
46 node_vars = model1.get_node_variable_names()
47 for i in range(len(node_vars)) :
48     val1 = model1.get_node_variable_values(node_vars[i], last1)
49     val2 = model2.get_node_variable_values(node_vars[i], last2)
50
51     dval = []
52     for j in range(len(val1)) :
53         dval.append(val2[j]-val1[j])
54
55     newmodel.put_node_variable_values('d' + node_vars[i], 1, dval)
56
57 # add original displacement back, so quantities can be viewed on deformed shape
58 newmodel.put_node_variable_values('displacement_x', 1,
59 model1.get_node_variable_values('displacement_x', last1))
60 newmodel.put_node_variable_values('displacement_y', 1,
61 model1.get_node_variable_values('displacement_y', last1))
62 newmodel.put_node_variable_values('displacement_z', 1,
63 model1.get_node_variable_values('displacement_z', last1))
64
65 # diff element variables
66 elem_vars = model1.get_element_variable_names()
67 for i in range(len(elem_vars)) :
68     val1 = model1.get_element_variable_values(1, elem_vars[i], last1)
69     val2 = model2.get_element_variable_values(1, elem_vars[i], last2)
70
71     dval = []
72     for j in range(len(val1)) :
73         dval.append(val2[j]-val1[j])
74
75         newmodel.put_element_variable_values(1, 'd' + elem_vars[i], 1, dval)
76
77 # diff global variables
78     global_vars = model1.get_global_variable_names()
79     for i in range(len(global_vars)) :
80         val1 = model1.get_global_variable_value(global_vars[i], last1)
81         val2 = model2.get_global_variable_value(global_vars[i], last2)
82
83         dval = val2-val1
84
85         newmodel.put_global_variable_value('d' + global_vars[i], 1, dval)
86
87 # close model
88 model1.close()
89 model2.close()
90 newmodel.close()
```

Appendix B

Callback functions for adaptive mesh refinement and coarsening

These callback functions are necessary for communication between the TopS data structure and the application code during refinement and coarsening of the mesh. The C code for the refinement callback is given in Section B.1 and the series of callback and application functions for coarsening are given in Section B.2.

B.1 Callback functions for adaptive refinement of 3D 4k mesh

The following call back function is called by TopS each time a new node is inserted due to adaptive refinement of the 3D 4k mesh. Please see section 4.4 for details on the implementation.

```
1  void Callback_InsertNode (TopRefinement4K3D* refinement, TopNode node, void*
       userdata) {
2
3      int intCount = 0;
4      int i;
5      TopModel* model;
6      ModelAtt* mAtt = (ModelAtt*) userdata;
7      NodeAtt* nAtt;
8      ElemAtt* eAtt;
9      ElemAtt*
10     parentAtt;
11     TopNodeElemItr* nElemItr;
12     TopElement elem, parentElem;
13     TopElement interpElems[8];
14     TopElement interpNodes[4];
15
16     // get model associated with the refinement manager
17     model = topRefinement4K3D_GetModel (refinement);
18
19     // Assign a node id if node does not have one
20     if (topNode_GetId (model, node) == -1) {
21         mAtt->nodeIds++;
22         topNode_SetId (model, node, mAtt->nodeIds);
23     } else {
24         printf ("New node already has an ID \n");
25         exit(1);
26     }
27
```

```
28      // Assign nodal attribute
29      nAtt = (NodeAtt*) malloc(sizeof(NodeAtt));
30      assert(nAtt);
31      initNodeAtt(nAtt);
32      topNode_SetAttrib (model, node, nAtt);
33
34      // Traverse elements incident to this new node
35      nElemItr = topModel_CreateNodeElemItr(model, node);
36      for (topNodeElemItr_Begin(nElemItr); topNodeElemItr_IsValid(nElemItr);
37      topNodeElemItr_Next(nElemItr)) {
38          elem = topNodeElemItr_GetCurr(nElemItr);
39          eAtt = (ElemAtt*)malloc(sizeof(ElemAtt));
40
41          // initialize new elements' attributes
42          initElemAtt (eAtt);
43          topElement_SetAttrib (model, elem, eAtt);
44          eAtt->flag_refined = 1;
45
46          // Assign new element ID
47          if (topElement_GetId(model, elem)==-1) {
48              mAtt->elemIds++;
49              topElement_SetId (model,elem, mAtt->elemIds);
50          } else {
51              printf("new element already has an ID = %d\n", topElement_GetId(model,
                     elem));      }
52
53          // call get parent element
54          parentElem = topRefinement4K3D_GetParentElem (refinement, elem);
55
56          // get the parent attribute
57          parentAtt = (ElemAtt*) topElement_GetAttrib(model, parentElem);
58
59          // set the refinement level of the new element
60          eAtt->refinement_level = parentAtt->refinement_level+1;
61
62          // check if a cohesive element is being split
63          if (topElement_IsCohesive(model, parentElem)) {
64              printf("Error: Cohesive element being split\n");
65          exit(1);
66          }
67
68          // save elements for interpolations
69          if (intCount == 0) {
70              interpElems[intCount] = parentElem;
71              intCount++;
72          }
```

```
73          if (topElement_GetId(model, interpElems[intCount-1]) != topElement_GetId(
                model, parentElem) ) {
74              interpElems[intCount] = parentElem;
75              intCount++;
76          }
77      } // end of node-element iterator
78      topNodeElemItr_Destroy(nElemItr);
79
80      // choose  first element for interpolation (interpElems[0])
81      for (i=0; i<nelnode; i++) {
82          interpNodes[i] = topElement_GetNode(model, interpElems[0], i);
83      }
84      elemInterpolate (model, node, interpNodes);
85
86      // delete parent element attribute (TopS deletes elements)
87      for (i=0; i<intCount; i++) {
88          eAtt = (ElemAtt*)topElement_GetAttrib (model, interpElems[i]);
89          free(eAtt);
90      }
91  }
```

B.2 Callback functions for adaptive coarsening of 3D 4k mesh

The following series of call back functions are called by TopS to coarsen regions of the 3D 4k mesh. Please see section 4.5 for details on the implementation.

B.2.1 CallBack_CanCollapseNode

```
1  int CallBack_CanCollapseNode    (TopRefinement4K3D* r,
2                                   TopNode node,
3                                   int nelems,
4                                   const TopElement elems1[],
5                                   const TopElement elems2[],
6                                   const TopElement
7                                   candidate_elems[],
8                                   void* userdata) {
9
10     int i,j,n;
11     double error;
12     ModelAtt* mAtt = (ModelAtt*) userdata;
13     TopModel* model;    model = topRefinement4K3D_GetModel (r);
14     TopElement* e1;
15     TopElement* e2;
16     TopElement* e3;
17
18     e1 = (TopElement *) calloc(nelems, sizeof(TopElement));
19     e2 = (TopElement *) calloc(nelems, sizeof(TopElement));
20     e3 = (TopElement *) calloc(nelems, sizeof(TopElement));
21
22     for (i=0; i<nelems; i++) {
23         e1[i] = elems1[i];
24         e2[i] = elems2[i];
25         e3[i] = candidate_elems[i];
26     }
27
28
29     // calculate strain on original and new patches
30     error = getPatchStrainError (model, mAtt, e3, e1, e2, nelems);
31
32     free(e1);
33     free(e2);
34     free(e3);
35
36     if (error < mAtt->AMC_tol) {
37         mAtt->nElemsRemoved+=nelems;
38         return 1;
39     } else {
40         return 0;
```

```
41     }
42 }
```

B.2.2 Callback_MergeElements

```
1  void Callback_MergeElements (TopModel* model,  TopElement old_el[2],
2                                  TopElement new_el, void* data) {
3
4      int i;
5      int nodeid;
6      ElemAtt* eAtt_new;
7      ElemAtt *eAtt_old[2];
8      ModelAtt* mAtt = (ModelAtt*) data;
9
10     // assign new element ID for new merged element
11     if (topElement_GetId(model, new_el)==-1) {
12         mAtt->elemIds++;
13         topElement_SetId (model,new_el, mAtt->elemIds);
14     } else {
15         printf("new element already has an ID = %d\n", topElement_GetId(model,
               new_el));
16     }
17
18     // initialize new element's attributes
19     eAtt_new = (ElemAtt*)malloc(sizeof(ElemAtt));
20     initElemAtt (eAtt_new);
21     topElement_SetAttrib (model, new_el, eAtt_new);
22     eAtt_new->flag_coarsened = 1;
23
24     // copy old element attributes to new element
25     eAtt_old[0] = (ElemAtt*) topElement_GetAttrib (model, old_el[0]);
26     eAtt_old[1] = (ElemAtt*) topElement_GetAttrib (model, old_el[1]);
27     eAtt_new->refinement_level = eAtt_old[0]->refinement_level-1;
28     topElement_SetAttrib (model, old_el[0], NULL);
29     topElement_SetAttrib (model, old_el[1], NULL);
30     free (eAtt_old[0]);
31     free (eAtt_old[1]);
32 }
```

B.2.3 Callback_RemoveNode

```
1 void Callback_RemoveNode (TopModel* model, TopNode node, void* data) {
2     NodeAtt* nAtt_old = (NodeAtt*) topNode_GetAttrib (model, node);
3     topNode_SetAttrib (model, node, NULL);
4     free (nAtt_old);
5 }
```

B.2.4 getPatchStrainError

```
1  double getPatchStrainError (TopModel* model, ModelAtt *mAtt,
2                              TopElement* coarseElems,  TopElement* fineElems1,
3                              TopElement* fineElems2, int numElemPairs) {
4
5      int i;
6      double error;
7      double *stress_coarse;  // element stress [xx, yy, zz, xy, yz, zx]
8      double *stress_fine1;   // element stress [xx, yy, zz, xy, yz, zx]
9      double *stress_fine2;   // element stress [xx, yy, zz, xy, yz, zx]
10     double *strain_coarse;  // element strain [xx, yy, zz, xy, yz, zx]
11     double *strain_fine1;   // element strain [xx, yy, zz, xy, yz, zx]
12     double *strain_fine2;   // element strain [xx, yy, zz, xy, yz, zx]
13     double *strain_diff1;   // diff btwn strain field on coarse element and one
           fine elem pair
14     double *strain_diff2;   // diff btwn strain field on coarse element and one
           fine elem pair
15     double norm_diff1;
16     double norm_diff2;
17
18
19     // allocate space
20     stress_coarse = createDoubleArray (nstrains, 1);
21     stress_fine1  = createDoubleArray (nstrains, 1);
22     stress_fine2  = createDoubleArray (nstrains, 1);
23     strain_coarse = createDoubleArray (nstrains, 1);
24     strain_fine1  = createDoubleArray (nstrains, 1);
25     strain_fine2  = createDoubleArray (nstrains, 1);
26     strain_diff1 = createDoubleArray (nstrains, 1);
27     strain_diff2 = createDoubleArray (nstrains, 1);
28
29     error = 0;
30     for (i=0; i<numElemPairs; i++) {
31         // get element strains
32         elemStressStrain (model, mAtt, coarseElems[i], stress_coarse,
               strain_coarse);
33         elemStressStrain (model, mAtt, fineElems1[i], stress_fine1, strain_fine1);
34         elemStressStrain (model, mAtt, fineElems2[i], stress_fine2, strain_fine2);
35
36         // get difference between strain fields on coarse and fine elements
37         addVectors (strain_diff1, strain_coarse, strain_fine1, nstrains, -1.0);
38         norm_diff1 = dotProduct (strain_diff1, strain_diff1, nstrains);
39         error += norm_diff1;
40         addVectors (strain_diff2, strain_coarse, strain_fine2, nstrains, -1.0);
41         norm_diff2 = dotProduct (strain_diff2, strain_diff2, nstrains);
42         error += norm_diff2;
43     }
44
```

```
45     // destroy matrices
46     destroyDoubleArray(stress_coarse);
47     destroyDoubleArray(stress_fine1);
48     destroyDoubleArray(stress_fine2);
49     destroyDoubleArray(strain_coarse);
50     destroyDoubleArray(strain_fine1);
51     destroyDoubleArray(strain_fine2);
52     destroyDoubleArray(strain_diff1);
53     destroyDoubleArray(strain_diff2);
54
55     return error;
56 }
```