Pontifícia Universidade Católica
do Rio de Janeiro

Andrei Alhadeff Monteiro

Mapping Cohesive Fracture and Fragmentation Simulations to GPUs

TESE DE DOUTORADO

Thesis presented to the Programa de Pós-Graduação em Informática of the Departamento de Informática, PUC-Rio as partial fulfillment of the requirements for the degree of Doutor em Ciências - Informática.

Advisor: Prof. Waldemar Celes Filho

Rio de Janeiro
September 2015

PUC-Rio - Certificação Digital Nº 1121801/CB

Pontifícia Universidade Católica
do Rio de Janeiro

Andrei Alhadeff Monteiro

Mapping Cohesive Fracture and Fragmentation Simulations to GPUs

Thesis presented to the Programa de Pós-Graduação em Informática, of the Departamento de Informática do Centro Técnico Científico da PUC-Rio, as partial fulfillment of the requirements for the degree of Doutor.

Prof. Waldemar Celes Filho
Advisor
Departamento de Informática – PUC-Rio

Profa. Noemi de La Rocque Rodriguez
Departamento de Informática – PUC-Rio

Prof. Hélio Côrtes Vieira Lopes
Departamento de Informática – PUC-Rio

Prof. Glaucio Hermogenes Paulino
University of Illinois

Prof. Diego Fernandes Nehab
IMPA

Prof. José Eugenio Leal
Coordinator of the Centro Técnico Científico da PUC-Rio

Rio de Janeiro, September 14th, 2015

PUC-Rio - Certificação Digital Nº 1121801/CB

All rights reserved.

Andrei Alhadeff Monteiro

Graduated in Computer Engineering at Pontifícia Universidade Católica do Rio de Janeiro. Obtained a Master's degree in Computer Science at Pontifícia Universidade Católica do Rio de Janeiro, acting in the areas of physics animation and engineering simulation together with GPU programming. While doing his Masters, he worked as a researcher at Tecgraf/PUC-Rio with reservoir simulation and rendering. He then did his PhD in Computer Science at Pontifícia Universidade Católica do Rio de Janeiro obtaining a full CNPQ scholarship, while at Tecgraf/PUC-Rio.

PUC-Rio - Certificação Digital Nº 1121801/CB

Bibliographic data

Monteiro, Andrei Alhadeff

Mapping Cohesive Fracture and Fragmentation Simulations to GPUs / Andrei Alhadeff Monteiro; advisor: Waldemar Celes Filho. – 2015.

155 f: il.; 30 cm

Tese (Doutorado em Informática) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2015.

Inclui bibliografia.

1. Informática – Teses. 2. Simulação de fragmentação. 3. GPUs. 4. Método dos Elementos Finitos. 5. Elementos Coesivos. 6. CUDA. I. Filho, Waldemar Celes. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

PUC-Rio - Certificação Digital Nº 1121801/CB

Acknowledgments

To my family, for all the support they have given throughout my life. To my father, Ivan de Castro Monteiro, my mother, Myriam Alhadeff Monteiro, and my sister, Camila Alhadeff Monteiro.

To my advisor, Waldemar Celes Filho, without whom the research would not be possible. Thank you for motivating me throghout these whole years as my adviser and teacher.

To Professor Glaucio H. Paulino and Dr. Sofie Leon, for giving all the support for our reasearch and making it possible.

To Rodrigo Espinha, for helping me always during my difficulties.

To Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPQ), to whom this research would not be possible.

To Tecgraf/PUC-Rio laboratory, for giving me opportunity to face such challenges and learning with them.

To all my friends for their support and friendship.

Abstract

Monteiro, Andrei Alhadeff; Filho, Waldemar Celes (Advisor). **Mapping Cohesive Fracture and Fragmentation Simulations to GPUs**. Rio de Janeiro, 2015. 155p. PhD Thesis — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A GPU-based computational framework is presented to deal with dynamic failure events simulated by means of cohesive zone elements. We employ a novel and simplified topological data structure relative to CPU implementation and specialized for meshes with triangles or tetrahedra, designed to run efficiently and minimize memory requirements on the GPU. We present a parallel, adaptive and distributed explicit dynamics code that implements an extrinsic cohesive zone formulation where the elements are inserted "on-the-fly", when needed and where needed. The main challenge for implementing a GPU-based computational framework using an extrinsic cohesive zone formulation resides on being able to dynamically adapt the mesh, in a consistent way, by inserting cohesive elements on fractured facets and inserting or removing bulk elements and nodes in the adaptive mesh modification case. We present a strategy to refine and coarsen the mesh to handle dynamic mesh modification simulations on the GPU. We use a reduced scale version of the experimental specimen in the adaptive fracture simulations to demonstrate the impact of variation in floating point operations on the final fracture pattern. A novel strategy to duplicate *ghost* nodes when distributing the simulation in different compute nodes containing one GPU each is also presented. Results from parallel simulations show an increase in performance when adopting strategies such as distributing different jobs amongst *threads* for the same element and launching many *threads* per element. To avoid concurrency on accessing shared entities, we employ graph coloring for non-adaptive meshes and node traversal for the adaptive case. Experiments show that GPU efficiency increases with the number of nodes and bulk elements.

Keywords

Fragmentation simulation; GPUs; Finite Element Method; Cohesive elements; CUDA;

PUC-Rio - Certificação Digital Nº 1121801/CB

Resumo

Monteiro, Andrei Alhadeff; Filho, Waldemar Celes. **Mapeamento de Simulação de Fratura e Fragmentação Coesiva para GPUs**. Rio de Janeiro, 2015. 155p. Tese de Doutorado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Apresentamos um método computacional na GPU que lida com eventos de fragmentação dinâmica, simulados por meio de zona coesiva. Implementamos uma estrutura de dados topológica simples e especializada para malhas com triângulos ou tetraedros, projetada para rodar eficientemente e minimizar ocupação de memória na GPU. Apresentamos um código dinâmico paralelo, adaptativo e distribuído que implementa a formulação de modelo zona coesiva extrínsica (CZM), onde elementos são inseridos adaptativamente, onde e quando necessários. O principal objetivo na implementação deste *framework* computacional reside na habilidade de adaptar a malha de forma dinâmica e consistente, inserindo elementos coesivos nas facetas fraturadas e inserindo e removendo elementos e nós no caso da malha adaptativa. Apresentamos estratégias para refinar e simplificar a malha para lidar com simulações dinâmicas de malhas adaptativas na GPU. Utilizamos uma versão de escala reduzida do nosso modelo para demonstrar o impacto da variação de operações de ponto flutuante no padrão final de fratura. Uma nova estratégia de duplicar nós conhecidos como *ghosts* também é apresentado quando distribuindo a simulação em diversas partições de um cluster. Deste modo, resultados das simulações paralelas apresentam um ganho de desempenho ao adotar estratégias como dsitribuir trabalhos entre *threads* para o mesmo elemento e lançar vários *threads* por elemento. Para evitar concorrência ao acessar entidades compartilhadas, aplicamos a coloração de grafo para malhas não-adaptativas e percorrimento nodal no caso adaptativo. Experimentos demonstram que a eficiência da GPU aumenta com o número de nós e elementos da malha.

Palavras–chave

Simulação de fragmentação; GPUs; Método dos Elementos Finitos; Elementos Coesivos; CUDA;

PUC-Rio - Certificação Digital Nº 1121801/CB

Contents

1 Introduction 18
1.1 Related work 21
1.2 Many-core devices 24

2 Cohesive Fracture and Fragmentation Simulation 29
2.1 Numerical representation of quasi-brittle dynamic fracture 29
2.2 Simulation steps 30

3 Two-dimensional Cohesive Fracture and Fragmentation Simulation 37
3.1 Data Structure 37
3.2 Parallel Implementation 44
3.3 Experimental results 56

4 Massively Parallel Adaptive Mesh Refinement and Coarsening for 2D Dynamic Fracture Simulations 66
4.1 Adaptive mesh modification on Graphical Processing Units 67
4.2 Adaptive cohesive fracture and fragmentation simulation 79
4.3 Experimental results 80

5 Three-dimensional Distributed Cohesive Fracture and Fragmentation Simulations 94
5.1 Distributed mesh and communication layer representation 94
5.2 Construction of the communication layer 98
5.3 Data Structure 98
5.4 Insertion of cohesive elements 102
5.5 Parallel simulation 105
5.6 Message extraction and sending 108
5.7 Experimental results 109

6 Physics-based Fracture and Fragmentation Simulation 124
6.1 Our approach 125
6.2 Fracture mode and constraint dynamics 126
6.3 Rigid body simulation 131
6.4 Collision detection and response 132
6.5 Physics-based simulation 133
6.6 Experimental results 136

7 Conclusion and Future Work 145

8 Bibliography 148

Bibliography 148

PUC-Rio - Certificação Digital Nº 1121801/CB

List of Figures

Figure 1.1 Diagram of a G80 architecture with 16 SMs and 128 SPs, based on the figures presented in [1]. 26
Figure 1.2 CUDA memory hierarchy, based on the figures presented in [1]. 27

Figure 2.1 Schematic of the cohesive zone model approach. The cohesive zone ahead of the macro crack tip consists of voids and micro-cracks, 29
Figure 2.2 The surfaces of the macro crack tip are completely separated and are traction free. Then separation decreases and traction increases into the cohesive zone. 30
Figure 2.3 T6 mesh attributes belonging to the simulation. 31
Figure 2.4 Cohesive element insertion algorithm on a T3 mesh. (1) Mesh with initial facets that need to be fractured. Elements belonging to each node are traversed and cohesive element is inserted but no node is duplicated. (2, 3) The other fractured facet is checked for node duplication, the cohesive element is inserted and the node is marked as needing duplication. (4) Node is duplicated by traversing through the elements and updating the node index of the node belonging to them. 35

Figure 3.1 A special-purpose simplified data structure with mesh parameters of a T6 mesh. 38
Figure 3.2 Simulation's computational patterns. (a) a node can be updated based on its own information; (b) a bulk element can be udpated based on its own information; (c) a bulk element can be updated based on information of its nodes; and (c) a node can be updated based on information of its incident bulk elements. 39
Figure 3.3 Simulation parameters data structure diagram of FEM model. Global memory is used for attributes that change throghout the simulation. Texture memory is used for attributes that are constant during the entire simulation, but occupy too much memory space. Constant memory is used for attributes that are constant during the entire simulation, but are common to all elements and node, therefore requiring few memory space. 40
Figure 3.4 Traversal algorithm from a given node using the proposed data structure. The illustrated path does not contain cohesive elements. 42

PUC-Rio - Certificação Digital Nº 1121801/CB

Figure 3.5 Traversal algorithm from a given node using the proposed data structure, with cohesive elements along the path. From a bulk element, the algorithm starts by accessing a node whose opposite element is incident to the traversed node (central node) (1). The opposite element to that node is obtained (2) followed by the next node (3). The third bulk element is accessed (4), followed by its respective node (5). A cohesive element opposite to a node (or adjacent to a bulk element's facet) can also be reached (6), since it is explicitly represented in the element table (see Figure 3.1). 42
Figure 3.6 Node update algorithms: (1) incident elements traversal (or gather), and (2) element sweep (or scatter). 44
Figure 3.7 (1) Bulk elements are re-arranged in color groups (preferable balanced) and the same kernel per color group is called to avoid writing conflicts. (2) Example of a colored T6 structured mesh (3) and using the colored mesh and scatter strategy to update nodal masses of the group of elements in the current color in parallel. 46
Figure 3.8 Fracture and fragmentation simulation loop. 48
Figure 3.9 Splitting the kernel that computes stress and strain into simpler kernels. 49
Figure 3.10 To accumulate the stresses and strains on the nodes, we launch 12 threads per element, where each thread will accumulate part of the stress and strain matrices by fetching from the element shape functions and from the stress and strain at the Gauss points. 50
Figure 3.11 Cohesive elements insertion on a T6 mesh. (1) Mesh with initial cracks and facets that fractured facets. Coloring is used to avoid duplicating nodes of elements that share nodes in parallel. (2) From each facet node belonging to the element in the current color group, the algorithm traverses through its incident elements. (3) Nodes that need duplication. (4) T6 mesh with final node duplications and new cracks and cohesive elements. The fractured facets from the next color group are checked for cohesive elements insertion. 51
Figure 3.12 When computing internal forces, a thread per stiffness matrix line is launched using the color model and used to perform a dot product with the displacement vector in shared memory. In this example, the first image shows two elements per block used. The second image shows the matrices transposed so memory reads can be coalesced (that is, each consecutive thread reads consecutive memory addresses). 54
Figure 3.13 Splitting the kernel that computes cohesive forces into simpler kernels. 55
Figure 3.14 T6 disc mesh used to test insertion of cohesive element decoupled from analysis code. 57
Figure 3.15 Time for cohesive elements insertion of a T6 mesh. 58

PUC-Rio - Certificação Digital Nº 1121801/CB

Figure 3.16 Two-dimensional model of a rectangular specimen with initial notch of 2 mm. Initial strain is 0.015, with node thickness of 1 mm. Model dimensions are 16mm per 4mm. 59
Figure 3.17 T6 FEM mesh with 36,864 bulk elements at the end of the fragmentation simulation. 60
Figure 3.18 Refined T6 FEM mesh with 147,456 bulk elements at the end of the fragmentation simulation. 60
Figure 3.19 Strain energy evolution with crack propagation. 61
Figure 3.20 Extruded view of fragmented 2D plate with 74,257 nodes and 36,864 bulk elements. 61
Figure 3.21 Average time of each *kernel* of the simulation for a T6 mesh with 36,864 bulk elements. 62
Figure 3.22 Total time each kernel takes in the entire simulation for a T6 mesh with 36,864 bulk elements. 63
Figure 3.23 2D model of a ring specimen. Initial pressure is 400 MPa, with node thickness of 1 mm. The inner radius is 0.08 m and the outer radius is 0.15 m. 63
Figure 3.24 The figure shows a T6 FEM mesh with 362,020 bulk elements and the strain energy's evolution with the crack propagation for times 5 μs (1), 20 μs (2), 25 μs (3), 50 μs (4), 60 μs (5), and 68 μs (6). 65

Figure 4.1 Schematic of GPU data structure for adaptive 4k mesh (a) progression of mesh refinement and element labeling, (b) node numbering on refined mesh, (c) Edge 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 Edge labels. 69
Figure 4.2 (a) Refinement of elements 1, 2 and 3 from level 0 (white) to level 2 (dark grey) 72
Figure 4.3 Cohesive elements insertion on non-colored mesh: (a) Cohesive elements are inserted on fractured facets (in black) by launching one thread per bulk element that contains at least one fractured facet. (b) Launching one thread per node and by accessing one of the node's adjacent element, traversal begins in one direction until first cohesive element is reached (1b). Traversal direction changes (2b) and when second coehesive element is reached, the nodes of the bulk elements after the cohesive elements are duplicated (5b). Steps are repeated until adjacent element is reached again (6b). 74
Figure 4.4 Identifying facets of elements that need refinement given local region cells. (a) Crack tips emerge from the simulation analysis and (b) each define a refinement region radius. (c) The domain is discretized into cells, (d) in which each contain its own counter to account for overlapping regions. 75
Figure 4.5 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. 77

PUC-Rio - Certificação Digital Nº 1121801/CB

Figure 4.6 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. 78
Figure 4.7 Kalthoff-Winkler problem geometry and loading conditions 81
Figure 4.8 Strain contour and crack propagation plots of the Kalthoff-Winkler experiment at different time instants: **a** 25 μs; **b** 32 μs; **c** 55 μs; **d** 90 μs 82
Figure 4.9 Inclined Plane problem geometry and loading conditions 83
Figure 4.10 Strain contour and crack propagation plots of the Inclined Plane experiment at different time instants: **a** 70 μs; **b** 195 μs; **c** 500 μs; 84
Figure 4.11 Micro-branching problem geometry and loading conditions 85
Figure 4.12 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. 86
Figure 4.13 Details of crack branching including kink in the main crack, crack branches, and secondary branches 87
Figure 4.14 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 90
Figure 4.15 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. 92
Figure 4.16 Detailed view of fracture pattern for the full scale micro-branching problem with an externally applied strain of 0.003 93

Figure 5.1 3D Beam mesh with over 2.3 million bulk elements in (a) its original form; (b) discretized into 15 mesh parts; and (c) showing the communication layer for each partition in red. 95
Figure 5.2 2-dimensional mesh partitioning. (a) Initial mesh; (b) Mesh partitioned and (c) with the communication layer and proxy and ghost entities. (d) illustrates each proxy and ghost entity refering to their local entity. 97

PUC-Rio - Certificação Digital Nº 1121801/CB

Figure 5.3 3-dimensional mesh partitioning. (a) Intial mesh. (b) Mesh partitioned and (c) with the communication layer. (d) Illustration of proxy (in red) and ghost (in white) nodes and proxy elements in red. 97
Figure 5.4 Construction of the communication layer of a T6 mesh. (a) shows the initial mesh. (b) illustrates the partitioned mesh without the communication layer, but with the already classified local and proxy nodes. Adjacent elements from other partitions belonging to border nodes are swept and added to the communication layer (c) and (d) ghost nodes are classified according to facets. 98
Figure 5.5 Element types used in the 3D analysis code. Bulk elements are volumetric tetrahedron elements and cohesive elements are linear triangular elements. 100
Figure 5.6 Data strucuture for distributed 3-dimensional finite element analysis. The node table contains the positions, one adjacent element *id* belonging to its incidence, and the owner partition and entity *id* it belongs to. If the owner patition is the one it is in, the owner *id* is the same as the node table. The partition *id* contains 2 bits that indicate if the node is local, proxy, or ghost. The remaining bits refer to the owner partition identifier. The element table contains six nodes, in which four are used for tetrahedron bulk elements and six are used for triangular cohesive elements. Four opposite identifiers indicate elements opposite to the bulk elements' four nodes and the cohesive element's two facets. The owner partition *id* reserves one bit to indicate if the element is local or proxy and the remaining bits to indicate the partition of the owner. The owner identifier of the entity follows next. 101
Figure 5.7 Messaging Procedure to obtain topology or analysis data. (1) The node accesses the adjacent element and sends a message to its owner with the tuple *(owner element handle, local id)*. (2) The owner of the element receives the message and accesses the node with it. (3) The receiver responds the message with the topology or analysis attribute requested by the sender. 102
Figure 5.8 Insertion of cohesive elements in 3D. Node is traversed using a Breadth-first search in shared memory. If there is at minimum one path to the adjacent element, the node is not duplicated. 103
Figure 5.9 Phase 1 of inserting cohesive elements in partitioned meshes. Local and proxy cohesive elements are inserted. 104
Figure 5.10 Phase 1 of insertion of cohesive elements in partitioned meshes. Local, proxy, and ghost nodes are duplicated as a result of the cohesive elements insertion. Adjacent elements are updated accordingly. 105
Figure 5.11 Phase 2 of insertion of cohesive elements. References of proxy nodes and cohesive elements to their owners are updated by sending message via their adjacent elements. 106

PUC-Rio - Certificação Digital Nº 1121801/CB

Figure 5.12 This figure shows the compacting and sending of messages to neighbor partitions. First, ghost nodes of the local partition are swept and count how many messages it must send to each partition via atomic intrinsincs. Next, we scan the number of messages per partitions to create an offset array to fill the messages. We sweep the nodes again and using the offset array and the "number of messages per partition" array, we fill the tuple messages per partition and copy them to the CPU. 108
Figure 5.13 This figure illustrates the message trasfer between two computer nodes with GPUs. It is a costly stage of the simulation due to the fact of the memory trasfer between CPU and GPU before sending messages to other computer nodes and after receiving messages. 109
Figure 5.14 Colored 3D ring specimen (a) and after the insertion of cohesive elements (b). 110
Figure 5.15 3D mixed-mode problem geometry and loading conditions. 111
Figure 5.16 Strain contour and crack propagation plots of the 3D mixed-mode experiment at different time instants: **a** 0 μs; **b** 100 μs; **c** 130 μs; **d** 180 μs; **e** 200 μs; **f** 270 μs 113
Figure 5.17 Crack propagation plots of the refined version 3D mixed-mode experiment with 716,736 bulk elements, at time instant t = 270 μs. 114
Figure 5.18 Average time of each *kernel* of the simulation for a Tet4 mixed-mode 3D beam mesh with 113,984 bulk elements. 114
Figure 5.19 (a) 3D Ring model with 5 partitions after the cohesive elements insertion. (b) and (c) Cohesive insertion-time graph for each partition and discretization. 115
Figure 5.20 Distributed 3D mixed-mode problem geometry and loading conditions. 117
Figure 5.21 Number of compute nodes versus time Graph for the reduced and mid-scale 3D mixed-mode beam specimens. The time for Beams 38x13x4 and 76x26x8 are in seconds and minutes, respectively. 118
Figure 5.22 Mesh partitioning at the end of the simulation at t = 270 μs for the reduced-scale 3D mixed-mode beam. Notice the uneven distribution of cohesive elements (shown in red) between partitions. 119
Figure 5.23 GPU profiling for each node of the reduced-scale 32x13x4 beam, running in 5 compute nodes, not including message passing from OpenMPI. 120
Figure 5.24 Graph showing the distribution of simulation time in each node between node synchronization and GPU computations for the reduced-scale 3D mixed-mode beam specimen. 120
Figure 5.25 Graph showing the distribution of kernels along time in CUDA streams. Empty spaces that appear repeatedly in the timeline occur immediately after the nodes copy the message to the CPU and immediately before sending them to the receiver. 121

PUC-Rio - Certificação Digital Nº 1121801/CB

Figure 5.26 First large-scale 3D mixed-mode beam specimen at simulation end at t = 219 μs with 2,351,424 bulk elements. 122
Figure 5.27 Second large-scale 3D mixed-mode beam specimen at simulation end at t = 220 μs with 5,495,168 bulk elements. Cohesive elements are unevenly distributed, which leads to extra cohesive computations in certain partitions, fewer in some, and idle in others. The main crack propagates like in the results of the single GPU, at 30°. 123

Figure 6.1 The figure illustrates our framework to deal with physics-based animation of brittle and quasi-brittle objects. The objects starts falling in rigid body mode. When it collides, it switches to fracture mode and cohesive elements are inserted. When no cohesive elements are added in a while, it switches back to rigid body mode. 126
Figure 6.2 The figure illustrates the nodes belonging to the tetrahedra elements and connected by rigid bars. We use relaxation method with constraint functions proposed by Müller et al. to avoid time step instability. (a) shows each node moving away too much from its resting position as an effect of instability. (b) shows the correction we have to make in one node (m is the number of neighbor bars of the node). 127
Figure 6.3 Bunny model with 69,668 nodes and 208,353 bulk elements. Initial material parameters are as follows: initial velocity = 0 m/s, elastic modulus = 0.6 Pa, Poisson coefficient = 0.23, specific mass = 2400 kg/m3. Fracture energy materials are as follows: fracture energy G_I = 22 N/m, cohesive strength smax = 0.1 mPa, and shape parameter $\alpha = 2$. Time step is 10e-4 s, with stress calculated at every time step. Position-based parameters are: $k_{stiffness}$ = 0.9, $k_{damping}$ = 1 and number of iterations = 3. (a) The bunny begins falling under gravity action in rigid body mode. No fracture propagates in the model, although the simulation is done using the Cohesive Zones Model (CZM) (i.e. using explicit integration) and Müller´s Positions-based Dynamics (to avoid instability with increased time step); (b) The bunny collides with the ground. When the first node(s) collide(s), we switch to fracture mode. Stress calculation lead to cohesive elements verification, which leads to node duplication and fracture propagation; (c, d) The bunny continues in fracture mode because cohesive elements are still being added to the mesh; (e, f, g, h) Cohesive elements were not added at a certain number of steps, so the simulation is switched to rigid body mode. No fracture will propagate in the mode. Instead, each separate component of the bunny will be treated as a rigid body. 128
Figure 6.4 Rigid bar constraint and the displacements needed to "pull back" the nodes to the rest distance position. The projection of the constraint $C(p_1, p_2) = |p_1 - p_2| - d$, where the corrections $\Delta \mathbf{p}_i$ are weighted by the inverse masses $1/m_i$. 130

PUC-Rio - Certificação Digital Nº 1121801/CB

Figure 6.5 The figure illustrates the simulation flow chart. According to the chart, we can see clearly when the fracture and rigid body modes act on the simulation, which uses the same procedure from the CZM simulation, combined with Müller et al.'s constraint projection to avoid instability. 133
Figure 6.6 Verification of the engineering model with the relaxation method proposed by Müller et al [2] with time step 1,000 times greater than the versiona from chapter 5.7. 137
Figure 6.7 Three animations of a falling hollow sphere under gravity action. (a) The sphere collides and bounces in rigid form with $k_{stiffness}$ = 1. (b) The sphere collides and bounces in deformed form with $k_{stiffness}$ = 0.1. (c) The sphere collides and bounces in extreme deformed form with $k_{stiffness}$ = 0.01. 138
Figure 6.8 Two animations of a falling hollow sphere under gravity action. (a) The sphere collides and bounces in rigid form with $k_{stiffness}$ = 1 with a time step of 1e-4 s. (b) The sphere collides and bounces in deformed form with $k_{stiffness}$ = 1 and increased time step of 5e-4 s. 138
Figure 6.9 Animation of a falling plate under gravity action. (a) The plate collides and (b) bounces in deformed form with $k_{damping}$ = 0. 139
Figure 6.10 Animation of a falling rigid body plate under gravity action. (a) The plate collides and (b) bounces in rigid form with $k_{damping}$ = 1 and increased time step. 139
Figure 6.11 Two animations of a falling and breaking sphere under gravity action. (a) The sphere collides and breaks with smax = 5 mPa. (b) The sphere collides and breaks with smax = 500 mPa. 140
Figure 6.12 Animation of a falling hollow sphere under gravity action and breaking when colliding with the floor. 141
Figure 6.13 Animation of a hollow bunny mesh hit by a sphere under velocity v = 10 m/s. 142
Figure 6.14 Animation of a falling glass plate and breaking when colliding with the floor. 144

PUC-Rio - Certificação Digital Nº 1121801/CB

List of Tables

Table 2.1 Fragmentation algorithm 32

Table 3.1 Kernel subroutine call algorithm using mesh coloring 45
Table 3.2 Parallel Fracture Algorithm 47
Table 3.3 Parallel Node Duplication Algorithm 52
Table 3.4 Results for insertion of cohesive elements decoupled from analysis code. 57
Table 3.5 Simulation and mesh parameters for a T6 mesh and its refined version. 60
Table 3.6 Simulation and mesh parameters and results (GPU speedup and GPU and CPU time) for a T6 mesh and its refined version. 60
Table 3.7 Simulation and mesh parameters for a T6 mesh and its refined version. 64
Table 3.8 Simulation and mesh parameters and results (GPU speedup and efficiency and GPU and CPU time) for a T6 mesh and its refined version. 64

Table 4.1 3D distributed fracture and fragmentation algorithm 80
Table 4.2 Comparison of final quantities between Uniform, AMR and AMR+C simulations 87
Table 4.3 Variation in crack tip velocity, energy released, and occurrence of branching for 20 simulations of each the AMR and AMR+C enabled meshes 88
Table 4.4 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) 90

Table 5.1 3D distributed fracture and fragmentation algorithm 107
Table 5.2 Results for 3D insertion of cohesive elements in Ring specimen, decoupled from analysis code. 110
Table 5.3 Simulation and mesh parameters for mixed-mode 3D beam mesh and its refined version. 112
Table 5.4 Simulation and mesh parameters and results (GPU and CPU time) for mixed-mode 3D beam and its refined version. 112
Table 5.5 Topology and simulation data used to simulate the 3D distributed mixed-mode beams. Results are shown in time and speedup compared to a single GPU. 117

Table 6.1 Damping calculation for rigid bodies [2]. 132
Table 6.2 Proposed Physics-based simulation algorithm. 135

PUC-Rio - Certificação Digital Nº 1121801/CB

PUC-Rio - Certificação Digital Nº 1121801/CB

Insanity: doing the same thing over and over again and expecting different results.

Albert Einstein