4
Fragmentation Simulation

4.1
Simulation definitions

We propose a simple but effective finite element mesh representation for fracture, microbranching, and fragmentation simulations. For a 2D simulation, we have given support for linear (T3) and quadratic (T6) triangular elements. The finite element types include bulk elements the corresponding cohesive elements, which are explicitly represented as an independent element (3). Our data structure implicitly represents facets corresponding to interfaces between two adjacent bulk elements. The intrinsic cohesive model assumes that all cohesive elements are embedded in the mesh before the simulation begins (21). This leads to an unchanged mesh connectivity during the whole simulation process, but introduces an artificial reduction of stiffness. We adopt an extrinsic cohesive model, which assumes that separation between bulk elements only occurs when the interfacial traction reaches a finite strength (27, 21). Challenges emerge when using an extrinsic model, since it requires an adaptive insertion of cohesive elements and topological changes of finite element mesh during the simulation process.

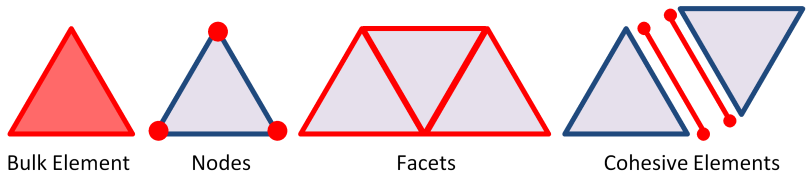

Figure 4.1: T3 mesh attributes belonging to the simulation.

The problem of fracture and fragmentation evolution occurs when a strain is applied on a finite element model that may contain an initial notch. During the simulation, internal, external, and cohesive forces at the nodes generate stresses along element interfaces, which may lead to fracture and fragmentation evolution. New nodes and cohesive elements are created whenever a facet fractures. Node attributes such as displacement, position, velocity, and acceleration are also updated from the internal, external, and

cohesive forces. In order to obtain a precise and stable simulation, one must properly adjust parameters such as material properties and adopt small time steps together with a highly discretized model.

```
1: Compute Stiffness Matrix
2: Update Nodal Mass
3: current step ← 0
4: while current step <= maximum step do
5:    Update Displacements
6:    if current step == check step then
7:       Compute Stresses
8:       if stresses > stress threshold then
9:          Insert Cohesive Elements
10:         Update Nodal Masses
11:      end if
12:   end if
13:   Compute Internal Forces
14:   Compute Cohesive Forces
15:   Update Velocities and Accelerations
16:   Update Boundary Conditions
17:   current step + = 1
18: end while
```

Table 4.1: Fragmentation algorithm

4.2 Pre-processing and updating

Given an initial triangular decomposition of the domain, during the pre-processing phase, mid-side node positions are computed in the mesh. This is done by linearly interpolating each facet edge nodes' positions. The stiffness matrix is then calculated for each bulk element. We consider the stiffness matrix to remain constant during the whole simulation. Each element's lumped mass matrix is initialized before the simulation. The lumped mass matrix contains mass values relative to each bulk element node. Therefore, the nodal masses are updated from the lumped mass matrix by going through the node's incident elements. The lumped mass matrix has to be upated every time the mesh changes. This occurs when cohesive element insertion results from a fractured facet between two bulk elements.

The simulation loop starts as the nodal displacements are updated with previous nodal velocity and accelerations using Euler´s integration rule shown in Equation 4-1.

$$\vec{u_{i+1}} = \vec{u_i} + \vec{v_i}\Delta t + \frac{1}{2}\vec{a_i}\Delta t^2 \tag{4-1}$$

4.3 Stresses

The stresses computation is the most costly step of the simulation. At a certain number of steps (we use ten), we compute the stress and strain at each bulk element node from their Gauss point evaluations using an extrapolation method. This whole procedure almost dominates the simulation time with excessive arithmetic operations and could be considered the bottleneck of the simulation loop if executed for all steps. To compute the stresses and strains at Gauss points of each bulk element, (for each of the three Gauss points considered in this work) in 2D, we first obtain the shape functions and its derivatives, compute the Jacobian matrix and its inverse, and compute the strains and displacements relation matrix. Using the material properties of the element, the constitutive matrix is calculated, followed by the stresses and strains at the Gauss points.

$$\sigma_{G_{element}} = \begin{pmatrix} \sigma_{1,1} & \sigma_{1,2} & \sigma_{1,3} & \sigma_{1,4} \\ \sigma_{2,1} & \sigma_{2,2} & \sigma_{2,3} & \sigma_{2,4} \\ \sigma_{3,1} & \sigma_{3,2} & \sigma_{3,3} & \sigma_{3,4} \end{pmatrix}_{3x4} \tag{4-2}$$

Using an extrapolation method, the stresses and strains matrices at the elements' nodes are obtained using the previous calculated stresses and strains matrices at the Gauss points and the element shape functions.

$$\begin{aligned} U_{\text{node i}} &= \text{Shape function} \\ \sigma_{\text{node 0}} &= U_{\text{node 0}} * \sigma_{G_{\text{element j}}} \end{aligned} \tag{4-3}$$

$$\begin{pmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{yx} & \sigma_{yy} \end{pmatrix}_{\text{node i}} = \begin{pmatrix} u_{1,1} \\ u_{1,2} \\ u_{1,3} \end{pmatrix}^T_{\text{node i}} \begin{pmatrix} \sigma_{1,1} & \sigma_{1,2} & \sigma_{1,3} & \sigma_{1,4} \\ \sigma_{2,1} & \sigma_{2,2} & \sigma_{2,3} & \sigma_{2,4} \\ \sigma_{3,1} & \sigma_{3,2} & \sigma_{3,3} & \sigma_{3,4} \end{pmatrix}_{3x4} \tag{4-4}$$

The principal stresses and their directions are calculated for each node. The principal stresses at each facet between two bulk elements are checked by analyzing if they exceed a limit for each of its three nodes. Average cohesive strengths are computed to check for cohesive element insertion. We indicate that a facet is fractured if a given stress value exceeds a threshold and the cohesive strength exceeds a material dependent limit.

4.4 Insertion of cohesive elements

Insertion of cohesive elements imposes topological changes in the mesh (21). After inserting the new cohesive element, each facet node is checked for duplication. Figure 4.2 illustrates a CPU algorithm for duplicating nodes on a T3 mesh. In 2D, the facet mid-side node must be duplicated. However, this assertion does not apply to the corner nodes, which must be checked by going through incident elements to which they belong. For each fractured facet, we verify if each of its corner nodes needs duplication. From a node, we traverse all its incident elements starting with one of the two adjacent elements the facet belongs to. If we reach the other adjacent element to the facet, the node is not duplicated. However, if not reached, the node must be duplicated. The global node counter is incremented and the new node index is retrieved from it. Once again we must traverse the adjacent elements to update incidence with the new node index. Finally, the facet mid-side node is updated with the node index also retrieved from the node incremented global counter.

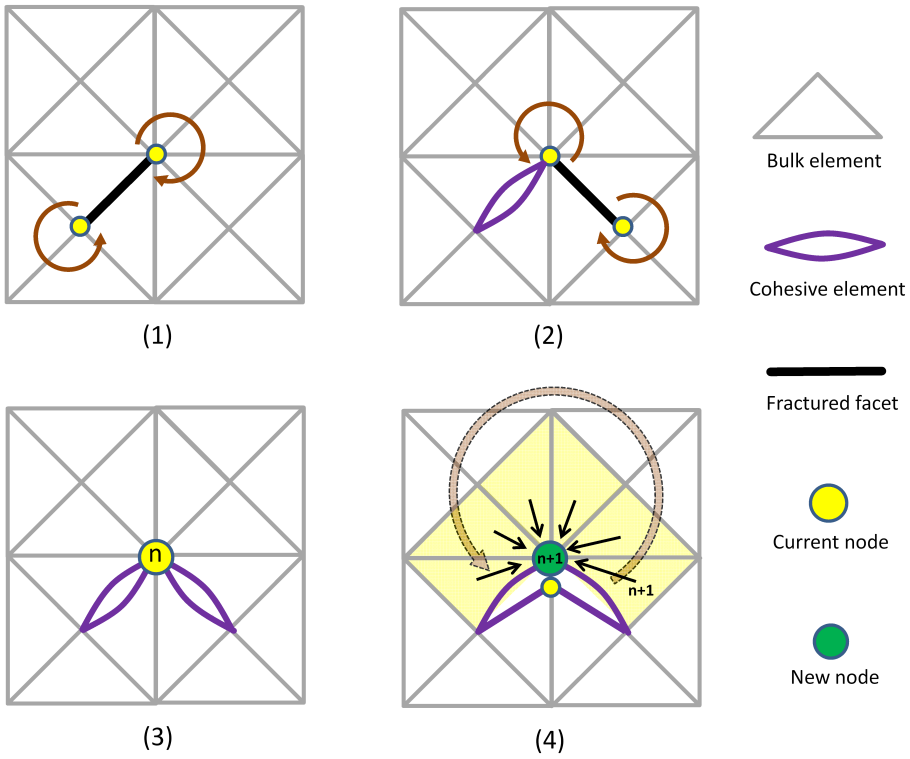

Figure 4.2: Cohesive element insertion algorithm on a T3 mesh. (1) Mesh with initial facets that need to be fractured. Elements belonging to each node are traversed and cohesive element is inserted but no node is duplicated. (2, 3) The other fractured facet is checked for node duplication, the cohesive 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.

If there were new cohesive elements added to the mesh, the topological changes indicate that some nodal masses also changed since bulk elements loose adjacency relationship. Therefore, the nodal mass must be updated again like on the pre-processing phase. We then initialize the nodal internal, external, and cohesive forces for future computations.

4.5 Internal and cohesive forces

The nodal internal force computation is another bottleneck for the simulation loop, since it must be done every step and requires a large number of arithmetic operations. The internal force vector results from a product of the stiffness matrix and the element displacement vector containing displacements for its six nodes in a T6 mesh, as shown in Equation 4-5. This means we

are multiplying a 12x12 matrix with a 12x1 vector, and it greatly reduces the performance of our parallel implementation due to its numerous global memory accesses.

$$R_{int_{12,12}} = \begin{pmatrix} k_{1,1} & k_{1,2} & \cdots & k_{1,12} \\ k_{2,1} & k_{2,2} & \cdots & k_{2,12} \\ \vdots & \vdots & \ddots & \vdots \\ k_{12,1} & k_{12,2} & \cdots & k_{12,12} \end{pmatrix} \begin{pmatrix} u_{1x} \\ u_{1y} \\ \vdots \\ u_{6y} \end{pmatrix} \tag{4-5}$$

The cohesive forces are then calculated by traversing through all cohesive elements and calculating their contributions to each node attached to them. The element vector of the deformed configuration is obtained, followed by the cohesive separations in the local coordinate system. Then, the separations and tractions at each Gauss point are calculated, together with the cohesive shape functions. Finally, the nodal cohesive force vector is obtained from cohesive traction and shape function. Together with the internal force and stresses, calculating the cohesive forces is one of the most costly computation steps within the simulation loop.

Accelerations are then computed from the cohesive and internal forces and nodal masses, which are then used to update the nodal velocities according to the following equations:

$$\begin{aligned} \vec{a_i} &= \vec{a_{i+1}} \\ \vec{a_{i+1}} &= \frac{\vec{R_{coh_i}} - \vec{R_{int_i}}}{m_i} \\ \vec{v_{i+1}} &= \vec{v_i} + \frac{1}{2}(\vec{a_i} + \vec{a_{i+1}})\Delta t \end{aligned} \tag{4-6}$$

Boundary conditions are then applied for each node belonging to the external model boundary by either imposing velocity or nodal force.

4.6 Node and element update

One key topological adjacent information needed to perform the simulation is the set of adjacent elements of a given node. This is necessary, for instance, to update element incidence when a node is duplicated due to the insertion of a new cohesive element. During the simulation step, we can also identify computations where such topological relationship can be used. As an example, we can consider the mass associated to a node, which depends on

contributions of all adjacent elements. However, accumulating contributions of adjacent elements to nodes is more efficiently handled by traversing all the elements in the model. For each element, we then accumulate its contribution to all incident nodes. In the end, the contributions of all corresponding adjacent elements will be accumulated for each node. In a serial code, this algorithm is straightforward and very efficient. In a parallel environment, writing conflicts arise, and one needs to ensure consistency, as we shall discuss. Figure 4.3 illustrates both strategies to compute nodal information from its adjacent elements.

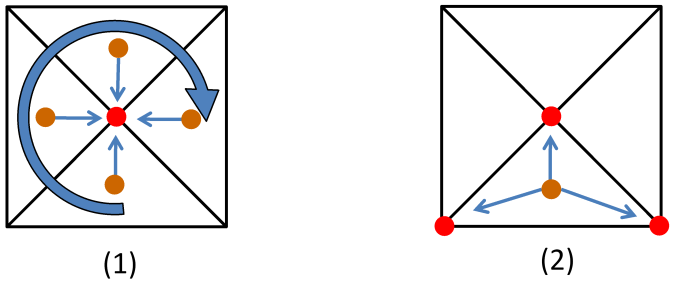

Figure 4.3: Node update algorithms. Incident elements traversal, or gather (1) and element sweep, or scatter (2).