Fracture and Fragmentation of Simplicial Finite Element Meshes using Graphs

Alejandro Mota\textsuperscript{1}, Jaroslaw Knap\textsuperscript{2}

\textsuperscript{1}Sandia National Laboratories, Livermore CA
\textsuperscript{2}Army Research Laboratory, Aberdeen MD

CAT Workshop
Santa Fe, NM, August 29, 2009
Existing Fracture Algorithm

Implemented using a 4-way linked-list data structure.

Pandolfi and Ortiz (1999)
Fragmentation Cases

Change of topology according to the number of boundary sides of the triangle to be duplicated:

- 0 sides
- 1 side
- 2 sides
- 3 sides
Motivation

- Parallel 3D fracture and fragmentation.
- Parallel 3D contact.
- Same topology representation for both.
- Reuse for 2D if possible.
- Better performance for large meshes.
- Independence of interpolation scheme.
- Correctness.
Simplices are graphs vertices, and colored edges represent connectivity and orientation. Graphs are directed.
FE Meshes as Graphs

- Building of graph is top-down.
- Vertices are shared.
- Original orientation kept by color maps.
FE Meshes as Graphs

Simplex:
\[ \sigma = [x_0, \ldots, x_n] \]

Face operator:
\[ d_i(\sigma) := [x_0, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n] \]

Incidence number:
\[ [\sigma^p, \sigma^{p-1}] := \begin{cases} 
0 & \text{if } \sigma^p \cap \sigma^{p-1} = \emptyset \\
1 & \text{if } \sigma^{p-1} = d_i(\sigma^p) \\
-1 & \text{if } \sigma^{p-1} = -d_i(\sigma^p) 
\end{cases} \]

Graph:
\[ V = \{v | v = f(\sigma) \in \mathbb{N}, \sigma \in K\} \]
\[ E = \{e | e = (u, v), u = f(\sigma^p) \in V, v = f(\sigma^{p-1}) \in V, \sigma^p \in K, \sigma^{p-1} \in K, [\sigma^p, \sigma^{p-1}] \neq 0\} \]
Local Ordering as Color Maps

Triangle 0 sees point 0 as the blue point in segment 0. Triangle 1 sees it as red.
Cube Mesh as Graph

Graphs are quite complex even for meshes with a few elements.
Plate with Hole Mesh

Same graph representation works for 2D. Easy to extract useful features.
Simplex Graph Properties

- Simplices are vertices.
- Edges represent connectivity.
- Directed graph.
- Built top-down.
- Edge color is local order of n-simplex wrt to (n+1)-simplex.
- An n-simplex may be shared by many (n+1)-simplices.
Simplex Graph Properties (2)

- Edge color map modifies color of edge directly below if target vertex is shared.
- Color maps are linear maps akin to reflections and rotations, or single and double permutations.
- Graph depth represents the dimension of the mesh. Works for N dimensions.
- Connectivity array can always be recovered from graph.
Graph Fracture Algorithm

- Mark open triangles, segments, and corner points.
- Build segment subgraphs.
- Clone open triangles.
- Split segment vertices.
- Build point subgraphs.
- Split point vertices.
Graph Fracture Algorithm

Algorithm 1. SPLIT($G, U, n, i$) Split articulation points.

Require: $U \subset V_i$, $i \leq n - 1$

1. for all $v \in U$ do
2. if $i < n - 2$ then
3. SPLIT($G, D^-(v), n, i + 1$)
4. else
5. CLONE($G, D^-(v), n$)
6. end if
7. $G'' \leftarrow G'(v) \setminus v$ // Check whether $v$ is an articulation point
8. for all $j \in \{2, \ldots, N(G'')\}$ do
9. $Y \leftarrow \{u | u \in G'', g(u) = i + 1\}$
10. $V' \leftarrow \{V', z\}$ // Split the vertex in the subgraph and graph
11. for all $u \in Y$ do
12. $E' \leftarrow E' \setminus (u, v)$
13. $E' \leftarrow \{E', (u, z)\}$
14. end for
15. end for
16. end for

Algorithm 2. CLONE($G, U, n$) Duplicate fractured interface simplices.

Require: $U \subset V_{i,n-1}$

1. for all $v \in U$ do
2. if $v \in V_{F,n-1}$ then
3. $Y \leftarrow D^-(v)$ // Note that $d^-(v) = 2$, hence $|Y| = 2$
4. $u_1 \leftarrow u \in Y$ s.t. $[f^{-1}(u_1), f^{-1}(v)] = 1$
5. $u_2 \leftarrow u \in Y$ s.t. $[f^{-1}(u_2), f^{-1}(v)] = -1$
6. $V \leftarrow \{V, w\}$
7. $E \leftarrow E \setminus (u_2, v)$
8. $E \leftarrow \{E, (u_2, w)\}$
9. for all $z \in D^+(v)$ do
10. $E \leftarrow \{E, (w, z)\}$
11. end for
12. end if
13. end for
Two-Tetrahedra Mesh

- Single shared triangle.
- Open simplices are triangle 2, segments 1,3,5 and points 1,2,3.
Fracture Algorithm (1)

First selected open point is 2, so process open segments 1 and 5 attached to it.
Fracture Algorithm (2)

Extract subgraph for segment 1, which clones open triangle 2, creating triangle 7.
Fracture Algorithm (3)

There are 2 branches in segment 1's subgraph, so the segment is split.
Fracture Algorithm (4)

Next is segment 5, now attached to 0 open triangles. Its subgraph has 2 branches.
Fracture Algorithm (5)

Both open segments for point 2 were split, its subgraph has 2 branches now.
Fracture Algorithm (6)

Next is point 1, with segment 3 the only remaining open segment.
Fracture Algorithm (6)

Next is point 1, with segment 3 the only remaining open segment.
Fracture Algorithm (7)

Segment 3 is split.
Fracture Algorithm (8)

Point 1 is split.
Finally point 3, with 0 open segments, is split, obtaining 2 separate tetrahedra.
Two-Tet Mesh and Graphs

Before

After
Correctness – Point Cubes

Two cubes joined by a point. Fracture all internal triangles. Current algorithm:

Graph representation:
Correctness – Edge Cubes

Two cubes joined by an edge. Fracture all internal triangles. Current algorithm:

Graph representation:
Graph Fracture Properties

- Subgraphs greatly simplify fracture.
- Operations on subgraphs are mirrored on parent graph.
- Localized operations confined to subgraphs, essential for parallelization.
- Time complexity of initialization linear with number of elements.
- Time complexity of fracture linear with number of open simplices.
Graph Fracture Properties (2)

- Non-manifold cases handled correctly.
- Significant reuse of code.
- Marking of open simplices is top-down.
- Building of subgraphs is bottom-up.
- Fracture is top-down.
- Works for both 2D and 3D.
- Recursive with each level in the graph.
C++ Boost Graph Library

- Free peer-reviewed portable C++ source library.
- Works well with the C++ Standard Library.
- Generic, STL-like interface for manipulating and traversing graphs.
- Hides details of the graph data structure implementation.
Performance Evaluation

- Reference fracture implementation is the one used in the ARES FE code.
- Tested 20 3D meshes of various sizes and geometries.
- Initialization time plotted as a function of the mesh size.
- Fracture time plotted as a function of triangles to fracture.
Performance Evaluation (2)

- Improvements in C++ Boost library result in immediate performance gain.
- First implementation.
- Some performance tuning.
- No thorough profiling and extensive performance tuning yet.
- Tests performed using GCC on an Intel Xeon 1.5 GHz machine.
Initialization Time

**Time Complexity of Initialization**

Comparison of 20 meshes

- **Graph** $t_i = 4.5949 \times 10^{-5} n_E^{1.1153}$
- **List** $t_i = 4.7271 \times 10^{-7} n_E^{2.0946}$
Fracture Time

Time Complexity of Fracture Algorithm

Comparison of 20 meshes

Graph \( t_f = 2.2881 \times 10^{-4} n_T^{1.0343} \)

List \( t_L = 2.1735 \times 10^{-8} n_T^{1.8651} \)
3D Graph Parallel Fracture

Partition mesh and create graph for each partition.
3D Graph Parallel Fracture

- On each fracture step mark open simplices.
- Partition-boundary points are missing parts of their subgraphs. Supplement from remote partitions.
3D Graph Parallel Fracture

Apply serial fracture algorithm to each partition.
Subgraphs for partition-boundary points are in a consistent state across partition boundaries after serial fracture on each partition.
3D Graph Parallel Fracture

Exchange information between partition-boundary point subgraphs to identify and match newly created simplices across partition boundaries.
3D Graph Parallel Fracture

• Update boundary communicator.
• Discard the remote part of the graph for each partition and proceed with mechanics until next fracture step.
Testing

- Test on simple meshes yield the same results as serial fragmentation.
- The number of partitions is incremented until the partition algorithm (METIS) produces partitions with zero elements.

Cube example
2 to 14 partitions
Testing

- Test on simple meshes yield the same results as serial fragmentation.
Coarse FE Mesh - Serial

Material model: Porous plasticity informed by QC
Interface model: Strain localization element
Failure criterion: critical plastic strain $\sim 0.12$
Mesh: 39150 elements, 78850 nodes
Machine: Single-processor Linux PC, $\sim 1$ week run time

Flaw resolved explicitly
Material model: Porous plasticity informed by QC
Interface model: Strain localization element
Failure criterion: critical plastic strain ~ 0.12
Mesh: 220945 elements, 399998 nodes
Machine: LLNL ALC, 400 processors, 8 hours run time
Partition mesh and create graph for each partition.
Results: Coarse, Fine Meshes

Configuration at 75 μs with initial torsion

- Coarse Mesh - Serial
- Kinked cracks
- Fine Mesh - Parallel
Crack Angles

Serial simulation
Peak pressure: 6.1 MPa
Crack kink angle ~ 14 degrees

Experimental results
(Chao and Shepherd, 2004)

<table>
<thead>
<tr>
<th>Peak pressure [MPa]</th>
<th>Crack angle [degrees]</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>7</td>
</tr>
<tr>
<td>5.1</td>
<td>13</td>
</tr>
<tr>
<td>4.1</td>
<td>38</td>
</tr>
<tr>
<td>3.3</td>
<td>50</td>
</tr>
</tbody>
</table>

Parallel simulation
Peak pressure: 6.1 MPa
Crack kink angle ~ 16 degrees
Conclusions

- Graphs allow fracture by simple, repetitive operations.
- The use of the Boost library reduced the work load significantly.
- Localized fracture operations suitable for parallel implementation.
- Time complexity reduced practically from quadratic to linear.
- Non-manifold cases handled correctly.
Future Work

- Extend graph representation for CW complexes.
- Run very large fracture problems.
- Reuse graph mesh representation for serial and parallel contact.
- Replace connectivity arrays and use the graph for all FE computations.
- Mesh subdivision.