

Discontinuous Element Insertion Algorithm

Timothy J. Truster^{a †}

^a Department of Civil and Environmental Engineering, University of Tennessee, Knoxville, 318 John D. Tickle Engineering Building, Knoxville, TN 37921

Manuscript Preprint, 2015

Abstract

An algorithm is presented for inserting zero-thickness interface elements, termed herein as “couplers”, into continuous finite element meshes in two and three dimensions. Insertion is governed solely by the mesh topology and is specified according to regions or subdomains within the overall analysis domain, a geometrically intuitive means to designate the coupler locations. The algorithm is self-contained and requires only nodal coordinates and element connectivity as input. A wide class of volume elements and interface couplers are treated within the framework. Since the algorithm is topologically-based, interfaces of arbitrary complexity are naturally accommodated. Separate treatment is given to inserting couplers within regions and along region boundaries to improve efficiency. Numerical tests verify that the algorithm is computationally scalable and produces analysis suitable meshes.

Key Words: Interface elements, cohesive zone models, Discontinuous Galerkin method, mesh topology, open source code

1. Introduction

The popularity of discontinuous formulations for computational solid mechanics has steadily increased in recent years. Physical applications relevant to fracture mechanics include fragmentation of brittle materials [1, 2], delamination in composite materials [3-5], and grain boundary cracks in polycrystalline materials [6]. Similarly, the Discontinuous Galerkin (DG) method [7-10] has been applied in the solid mechanics field to efficiently model sharp gradient features and to enable mesh adaptivity. A common approach for numerically realizing these methods is using the so-called zero-thickness interface finite elements, whereby an element is

[†] Assistant Professor. Corresponding author: Ph: (865) 974-1913; Fax: (865) 974-2669, e-mail: ttruster@utk.edu

created through node duplication such that the two sides of the interface initially coincide but subsequently may separate apart. Unfortunately, at the present time, standard commercial finite element codes do not contain mesh generation features for zero-thickness elements. Thus, the burden is placed on the user to create the modified mesh connectivity, which can be highly non-trivial for complex meshes in three dimensions.

Various researchers have proposed algorithms for generating zero-thickness elements for specialized applications. Some of the earliest examples were developed for adaptively inserting the elements during fragmentation simulations [1, 11]. These algorithms support the extrinsic cohesive zone (CZ) modeling approach [12] whereby elements are inserted dynamically or “on-the-fly” to track the propagation of cracks within a brittle material under impact loading. While the earlier works were limited to triangular and tetrahedral elements, subsequent developments by other authors [13] led to methods for general element types. These topological algorithms have also been applied to high-performance computing on parallel architectures [14, 15] and on graphical processing units [16]. Note that these numerical methods typically require that the interface elements are inserted during the course of the simulation, from step to step of a nonlinear analysis, which can entail revisions to the finite element code structure. Other methods, such as intrinsic cohesive zone models and Discontinuous Galerkin approaches, instead require all interface elements to be present from the start of a simulation, so that the displacement discontinuity can be evolved through activation criteria and constitutive relations. Thus, these approaches could benefit from concurrent insertion algorithms where the entire mesh is processed at once in a more efficient manner compared to inserting interfaces individually.

More recently, an algorithm and open-source code [17] have been developed for inserting zero-thickness elements throughout the mesh as a pre-processing step. This algorithm treats the mesh as consisting of several geometrical regions and can insert interface elements along all region boundaries as well as between all volume elements within regions. It has been applied to model grain boundary cracking, composite delamination, and matrix cracking [18] using a hybrid DG-CZ formulation [2]. However, this approach can be applied only when interface elements are required along all inter-region boundaries rather than a selection of them. Therefore, a method that provides the user with greater flexibility for choosing the location and type of interface elements is desirable.

In the current work, a general-purpose algorithm is presented for inserting zero-thickness interface elements, termed herein as “couplers”, into specified regions of conforming meshes. The duplication of nodes to accommodate the couplers is treated in a systematic fashion by introducing the concept of sectors of elements attached to nodes. This concept ensures that the finite element interpolation space retains the proper features of continuity and discontinuity in the vicinity of the interface. The algorithm employs only topological operations on the original element connectivity

of the mesh to perform the coupler insertions. Therefore, it can be applied to general element types for two and three dimensional problems with minimal input from the user. Couplers can be selectively inserted within specific regions or along specific interfaces. Also, different types of couplers as well as different material properties may be directly assigned to these particular sets, providing an intuitive means to complete the description of the interfacial-modified mesh for analysis purposes. An implementation of the algorithm in MATLAB[®] is provided at <https://bitbucket.org/trusterresearchgroup/deiprogram>. Numerical studies are performed for a range of two and three dimensional problems to investigate the accuracy, efficiency, and scalability of the proposed algorithm.

In what follows, the topological definitions relevant to finite element meshes and interface problems are discussed in Section 2. The phases of the insertion algorithm and corresponding pseudo-code are presented in Section 3. Section 4 contains performance aspects of the method obtained for numerical examples. Conclusions are drawn in Section 5.

2. Topological Definitions

Consider a *domain* consisting of a conforming mesh of finite *elements* in two (2D) or three (3D) dimensional space. Throughout the following discussions, topological entities are identified by *italic* typeset. Each *element* in the mesh is defined by a set of *nodes* which are points within the *domain* associated with their particular coordinates; see the 2D example in Figure 1. In 2D, meshes containing a mixture of triangular and quadrilateral *elements* are considered, and in 3D, meshes containing tetrahedral, wedge, or hexahedral *elements* are permitted. In Figure 1, *nodes* are designated by numbers and *elements* are denoted by lower-case letters. For example, *element a* is composed of the three *nodes* 1, 4, and 5. The *edge* of an *element* in 2D is defined by the line segment connecting two *nodes*, and the *face* of an *element* in 3D is defined by the set of *nodes* connected by *edges* which form a closed loop within a single plane in space. The term *facet* is applied to refer either to an *edge* in 2D or a *face* in 3D in the algorithmic descriptions that follow.

In addition to the above standard features associated with finite element discretization, we define a *region* as a contiguous set of *elements* within the *domain*, which in general may form nonconvex subdomains and consist of spatially disjoint sets of *elements*. Each *element* in the *domain* is a member of exactly one *region*. In Figure 1, the *regions* are denoted by capital letters, and the *elements* belonging to each *region* share the same color. Examples of *regions* in the context of finite element modeling include the grains within a polycrystal, fibers and the surrounding matrix in composites, concrete and steel in reinforced concrete, and so forth, where each *region* is considered to have different material properties. However, herein a *region* is a purely geometrical construct to enable completely general applications. The role played by the *regions* is central to the algorithm for inserting the “interface elements”.

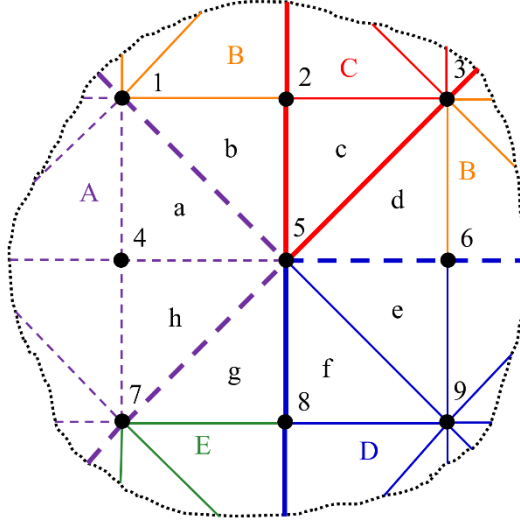


Figure 1. Conforming finite element mesh containing regions

The *facets* of all the *elements* in the *domain* can be separated into three disjoint sets. The first set are those *facets* which lie on the *domain* boundary, which are adjacent to exactly one *element*. The second set are those which lie between *elements* of two different *regions*, which are said to belong to *interfaces*. This set is further divided according to pairs of *regions*, such that $interface(A,B)$ is the set of all *facets* between *region A* and *region B*. The third set are those which lie between *elements* of the same *region*. Such *facets* belonging to *region C* will be denoted as $intraface(C)$, and so forth. *Interfaces* are shown as thick line segments in Figure 1 while *intrafaces* are shown as thin line segments.

3. Coupler Insertion Algorithm

A topological-based algorithm is presented for inserting *couplers* along the *interfaces* or *intrafaces* in the *domain*. In the finite element literature, such computational entities are typically referred to as “zero-thickness elements” or “interface elements”. Herein, we apply the term *coupler* to distinguish from the other topological definitions made in Section 2 and to provide for broader types of computational entities. Thus, a *coupler* is defined as a topological unit consisting of *nodes* from exactly two *elements* which are adjacent across either an *interface* or *intraface*. The *coupler* is generated by duplicating the *nodes* lying on the *facet* shared by the two *elements* to effectively split the *mesh* along that *facet*. These *couplers* commonly appear as numerical realizations of discontinuous formulations for modeling PDEs. For example, to model the progressive debonding in composites, intrinsic cohesive zone models can be introduced as *interface couplers* between the *elements* of the fiber and matrix constituents. The *couplers* are present in the analysis from the initial stage in order to capture the initiation and progression of fracture at the interface through a traction-separation relation. The treatment of the theory and

computational aspects of such formulations is beyond the scope of this work. Regarding the intrinsic cohesive zone models, the reader may consult [1, 2, 5, 19] for mathematical aspects and [17, 20] for notes on implementation. Similarly, the formulation [7] and implementation [8, 10] of the Discontinuous Galerkin method can be found elsewhere. The common feature of these and other discontinuous interpolation methods is that they require additional topological data beyond just the *nodes* and *elements* of the *mesh*.

Remark: *These numerical methods may be implemented into finite element codes by considering the “couplers” as generalized “elements” according to the template element approach [13, 21]. This approach enables a single assembly loop across various element types.*

Starting from an initially conforming finite element mesh, the insertion of *couplers* is accomplished by reference to the *interfaces* and *intrafaces* between various *regions*. The *region* is a natural geometric entity for assigning the desired location of the *couplers*. For example, when modeling debonding in fibrous composites or cavitation along grain boundaries in polycrystals, the *interfaces* between the different material *regions* is the desired location. Similarly, *intrafaces* are the natural location for *couplers* when using Discontinuous Galerkin numerical methods or when simulating general crack propagation with cohesive zone models.

The input data for the algorithm is simply the spatial coordinates of the *nodes*, the list of *nodes* connected to each *element*, and the list of *elements* belonging to each *region*. This topological data is provided by almost every finite element mesh generation software package, meaning that minimal data preparation is required by the user. Next, the list of *interfaces* and *intrafaces* is provided to specify which topological locations to insert the *couplers*. For example, the *interface* between a fiber *region A* and matrix *region B* would be indicated by flagging *interface(A,B)*; in Figure 1, this is indicated by the dashed lines between these *regions*. Herein, these *facets* where *couplers* are to be inserted are called “cut” *facets*. Similarly, the *intraface(A)* *facets* inside *region A* are dashed in Figure 1 to indicate that they will also be cut. The algorithm then determines the set of *couplers* to insert and the set of *nodes* to duplicate through an automated process based upon the topology of the mesh.

The crucial aspect of the insertion algorithm is the node duplication procedure, which relies upon the concept of *sectors* of *elements* surrounding a focus *node*. Two *elements* are defined to belong to a *sector* if the shared *facet* between them is not a cut *facet*, namely it is not designated for a *coupler*. A *sector* is then the largest set of *elements* satisfying this definition; a single *element* constitutes a *sector* if all of the *facets* of that *element* sharing the focus *node* are cut. Also, the union of all *sectors* is equal to the set of all *elements* surrounding the focus *node*. In general, a *sector* will consist of all *elements* for a single *region* only if all *interfaces* attached to that *region* are to be cut. Otherwise, a *sector* may consist of *elements* from multiple *regions*.

Referring to the mesh in Figure 1, *elements b, c, and d* belong to one *sector*, since *couplers* are

added along $interface(A,B)$ and $interface(B,D)$. Within the finite element discretization, each of these *elements* is required to have a continuous interpolation of the solution field. Therefore, the nodal coefficient of the solution field associated with this corner *node* must be identical for each of the three *elements*, so that the *sector* remains “solid”. *Elements* in other *sectors* will have a discontinuous functional interpolation, so the nodal value of the solution field should be independent or distinct between the *sectors*. For the example shown, the duplication of *node 5* onto each *region* does not satisfy the continuity requirement; this leads to an incorrect discretization shown in Figure 2 (a), with open gaps between *elements b* and *c*. Using such a discretization would lead to erroneous numerical results. Rather, a single copy of *node 5* should be assigned to these three *elements* which belong to the *sector*, as indicated by the single shading in Figure 2 (b). This situation arises because *regions B* and *C* are prescribed to remain connected without adding *couplers* along $interface(B,C)$. In contrast, within *region A*, *node 4* will be duplicated for all surrounding *elements*, since each *element* constitutes a *sector* due to the insertion of *intraface couplers*.

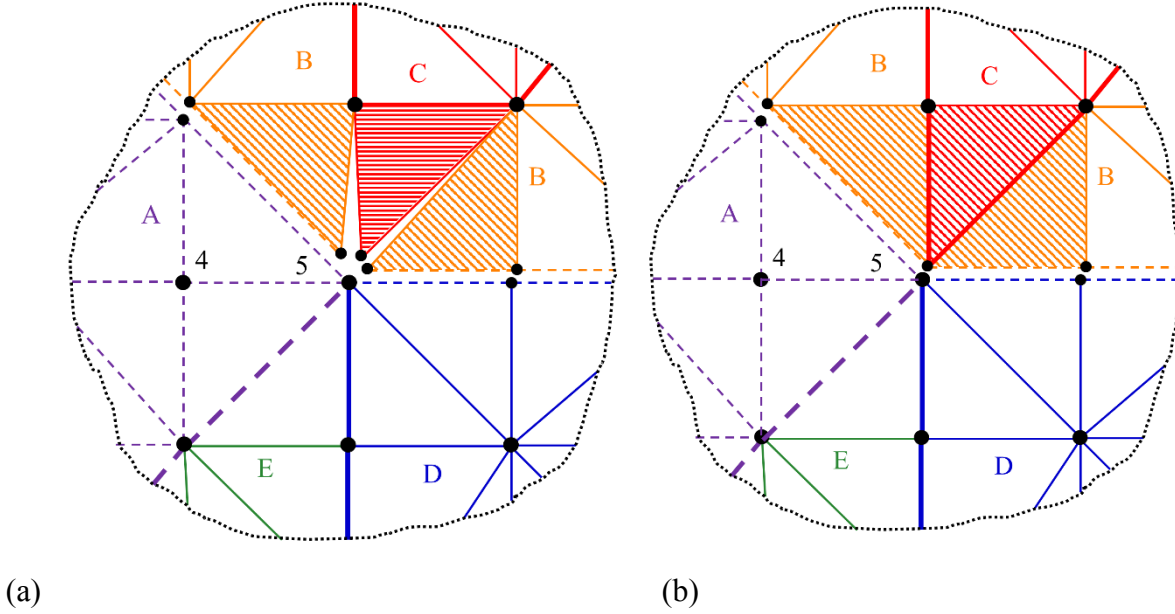


Figure 2. Node duplication: (a) incorrect discontinuity obtained by using regions; (b) correct continuity obtained by using sectors

Remark: The above definition ensures that a single sector is obtained for the cases of a node attached to only one cut edge in 2D and a node with cut faces lying on a single (non-smooth) manifold in 3D. Thus, the node would not be duplicated, and the field interpolation remains continuous at that node. See additional discussions and examples in [13].

The *coupler* insertion algorithm consists of six phases, which are summarized below:

1. Construct the set of *elements* attached to each *node*

2. Categorize all *facets* in the mesh as *boundary*, *interface*, or *intraface*.
3. Designate all cut *interface facets*, where *couplers* are to be inserted.
4. Duplicate *nodes* according to *sectors* for *interfaces*
5. Duplicate *nodes* for *intrafaces*
6. Construct the connectivity of *nodes* to *couplers* according to specified templates

Each of these phases is described in the paragraphs that follow and also outlined using pseudo-code in Box 1 – Box 6. Several explicit arrays are defined to record the relation between the entities of one topological category attached to a specific instance of an entity of another category. For example, `NodesOnElements` is the set of *nodes* attached to or “on” a particular *element*. In this array, the global *nodes* attached to a specific global *element* are assigned a local identifier, ranging from 1 to the number of *nodes* attached to the *element*. In the case shown in Figure 1, `NodesOnElement(d,2)=6` means that *node* 6 is the local-node 2 of *element* d. This nomenclature is extended to the relations of other topological entities, including `ElementsOnNode` and `FacetsOnElement`. Also, the inverse relation, between a local identifier and a global identifier, is designated by the pseudo-function `FindLoc`, such that `FindLoc(NodesOnElement(d,1:end),6)=2`, where the descriptor “1:end” implies that the search is across all local entries. For various arrays, the index “end+1” indicates that the specific local identifier is incremented by one to append a new global entity. The other pseudo-functions are denoted by self-explanatory names. Finally, short abbreviations of the topological entities are given in the Table 1. Local identifiers are generically designated by “loc” followed by the object letter, such as `locE`. Note that the distinct reference to *interfaces* excluding *intrafaces* is denoted by “ine” while the generic set of *interface* and *intraface* facets is denoted with “int”.

Remark: *The proposed algorithm does not assume any particular pattern exists between the numbering of nodes and element and their ordering in space. However, many of the arrays generated during the algorithm will possess an inherent ordering due to increasing node and element numbers. This fact can be used to speed up the searches for local indices.*

Table 1. Abbreviations for topological entities

Object name	Short name	Letter	Object name	Short name	Letter
Coupler	cou	C	Intraface	ina	I
Element	ele	E	Node	nod	N
Facet	fac	F	Region	reg	R
Interface	ine, int	I	Sector	sec	S

During the first phase in Box 1, the set of *elements* attached to each *node* in the mesh is determined, which is the inverse of the traditional finite element connectivity array. This array enables localized searches over *elements*. Due to its significant use within subsequent algorithmic phases, this array needs to be created and stored in memory. Note that the maximum number of *elements* attached to a *node*, called the valence, is not known a priori; hence the size of this array is indefinite.

Box 1. Construct the set of elements attached to each node

```

for each ele in domain
  for each locN on ele
    nod = NodesOnElements(ele,locN)
    ElementsOnNode(nod,end+1) = ele

```

During the second phase in Box 2, all *facets* in the mesh are categorized as either *boundary*, *interface*, or *intraface*. A simple way to traverse all *facets* is by looping over all *elements* and each of their local *facets*. By definition, two *elements* share a *facet* if all of the *nodes* attached to the local *facet* of the first *element* are also attached to a local *facet* of the second *element*. The second *element* is the common intersection of the sets of *elements* attached to the *nodes* on that particular *facet*. If a second *element* is not found, then that *facet* belongs to the *domain boundary*. Otherwise, the facet lies on an *interface* or *intraface* depending whether the two *elements* belong to different or identical *regions*, respectively. The *facet* is given a unique identifier, and information is recorded for both the *element-facet* and the *node-facet* relations. Note that the number of *nodes* per *facet*, denoted by NumNode, varies based on the *element* type; typically the corner *nodes* suffice for defining the *facet*. We also remark that this phase could instead be implemented as an implicit query for a *facet* rather than storing the entire array in memory. However, these topological relations are used multiple time during the algorithm, and also the information may be needed subsequently by the user to update nodal boundary conditions or other model features.

During the third phase in Box 3, each of the *interface facets* are indicated as cut if the associated *interface* is designated to have *couplers* inserted. In Figure 1, those sets indicated by thick dashed lines are *interface*(A,B), *interface*(A,E), and *interface*(B,D). Also, the associated *nodes* are flagged for duplication in phase 4. The *intraface*(A) *facets* will be treated separately in phase 5.

The fourth phase is the distinguishing feature of the insertion algorithm and prescribes the duplication of *interface nodes*. Because all *couplers* are predestined by the cut *interfaces* rather than inserted sequentially or adaptively, this process can be performed independently at each *node* in the *mesh*. The phase as shown in Box 4 has two steps: determining *sectors* and updating topology. One generic approach for determining *sectors* is described next; other approaches

Box 2. Categorize all facets in the mesh

```
for each ele in domain
  for each locF on ele
    for locN = 1:NumNode on locF
      nodI = NodesOnElements(ele,locN(locF,I))
      setI = ElementsOnNode(nodI,1:end)
[ele1,ele2] = SetIntersection(set1,set2,...,setNumNode)
    if ele1 == ele and ele2 > 0 then
      locF2 = DetermineLocalFacet(ele2,[nod1,nod2,...,nodNumNode])
      if FacetsOnElement(ele2,locF2) = 0 then
        fac = fac + 1
        reg1 = RegionOnElement(ele1); reg2 = RegionOnElement(ele2)
        int = Interface(reg1,reg2)
        ElementsOnFacet(fac,1:4) = [ele1,locF,ele2,locF2]
        FacetsOnElement(ele1,locF) = fac; FacetsOnElement(ele2,locF2) = fac
        FacetsOnInterface(int,end+1) = fac
        FacetsOnNode(nodI,end+1,1:3) = [fac,int,false] for each nodI
      else if only ele1 then
        fac = fac + 1
        ElementsOnFacet(fac,1:2) = [ele1,locF]
        FacetsOnBoundary(end+1) = fac
```

Box 3. Designate all cut interface facets

```
for each nod in domain
  for each ine in CutInterfaces
    set = FindLoc(FacetsOnNode(nod,1:end,2),ine)
    FacetsOnNode(nod,set,3) = true
    if size(set) > 0 then
      NodesOnInterface(end+1) = nod
```

could also be devised. First, each *element* attached to the *node* is assumed to belong to a different *sector*. Then, a loop is performed over all *facets* attached to the *node*. If a *facet* is not being cut, then the *sectors* containing the two *elements* sharing that *facet* are merged together. At the completion of the loop, all *elements* that are linked by uncut *interface facets* will be agglomerated into common *sectors*. Since all logic operations are performed one-way (merger) rather than two-way (merger/separation) and each *element* is a member of exactly one *sector* at each step, this *procedure* is a complete process for determining the *sectors*. Both corner *nodes* as well as mid-edge *nodes* for 3D *facets* are properly treated by the operations in Box 4. However, mid-edge *nodes* in 2D and mid-face *nodes* in 3D can be handled more easily since they are only

attached to a single *facet*. Also note that the array `FacetsOnNode` must be explicitly stored for this implementation. As an alternative, a double loop could instead be performed over each *element* attached to the *node* and each local *facet* containing that *node*.

Remark: *This procedure involving sectors is similar to the topology traversal presented in [13] regarding the adaptive insertion of interface elements for extrinsic cohesive zone modeling. The conceptual difference is that herein each node is evaluated once during insertion of multiple couplers while therein each node is considered multiple times during insertion of individual couplers.*

Once the *sectors* around a *node* are identified, each *sector* is assigned a unique copy of the *node*. This step involves three topological changes: (i) addition of global *nodes*; (ii) duplication of nodal coordinates; and (iii) updates of *element* connectivity. Also, the new global *node* identifiers assigned to each *element* are stored in the array `ElementsOnNodeDup` for reference by the user to update other *mesh* information such as boundary conditions.

Box 4. Duplicate nodes according to sectors for interfaces

```

for each nod in NodesOnInterface
    Sectors(1:end,1) = ElementsOnNode(nod,1:end)
    for each locF on FacetsOnNode(nod,1:end)
        if not FacetsOnNode(nod,locF,3) then
            fac = FacetsOnNode(nod,locF)
            ele1 = ElementsOnFacet(fac,1); ele2 = ElementsOnFacet(fac,3)
            sec1 = FindParentSector(Sectors,ele1)
            sec2 = FindParentSector(Sectors,ele2)
            if sec1 not equal sec2 then
                Merge(Sectors(sec1,1:end),Sectors(sec2,1:end))
                Delete(Sectors(sec2,1:end))
            for each sec in Sectors(2:end)
                newnod = newnod + 1
                for each ele in Sectors(sec,1:end)
                    Coordinates(newnod,1:nD) = Coordinates(nod,1:nD)
                    locN = FindLoc(NodesOnElement(ele,1:end),nod)
                    NodesOnElementNew(ele,locN) = newnod
                    locE = FindLoc(ElementsOnNode(nod,1:end),ele)
                    ElementsOnNodeDup(nod,locE) = newnod

```

During the fifth phase in Box 5, *node* duplication is performed for all cut *intraface facets*. This operation results in a fully discontinuous interpolation of the fields within the associated *region*, such as *region A* in Figure 1. The duplication of *nodes* within a *region* is quite simple: each

element is ascribed a unique instance of all *nodes* attached to itself. Observe that the revised topology from phase 4 leads to individual *regions* that have the appearance of smaller *domains* due to the *node* duplication along *interfaces*. Thus, the process of treating *intrafaces* inside individual *regions* is identical to generating a completely discontinuous representation of a *domain*. While *intrafaces* could be treated alongside *interfaces* within Box 4, the separate treatment is computationally more efficient when there are significantly fewer *interface facets* than *intraface facets*, which is usually the case. The speed-up is achieved because the logical tests involving *sectors* are avoided for all *nodes* attached only to *intraface facets*. Note that phases 4 and 5 will lead to correct *meshes* only if, for each *region* with *intraface couplers*, all *interfaces* adjoining that *region* are also designated for *couplers*. Otherwise, cutting of both *interfaces* and *intrafaces* should be treated concurrently within phases 3 and 4.

Box 5. Duplicate nodes for intrafaces

```

for each ina in CutIntrafaces
  for each ele such that RegionOnElement(ele) = reg
    for each nod on NodesOnElement(ele,1:end)
      if InstancesOf(nod) > 1
        newnod = newnod + 1
        Coordinates(newnod,1:nD) = Coordinates(nod,1:nD)
        locN = FindLoc(NodesOnElement(ele,1:end),nod)
        NodesOnElementNew(ele,locN) = newnod
        locE = FindLoc(ElementsOnNode(nod,1:end),ele)
        ElementsOnNodeDup(nod,locE) = newnod

```

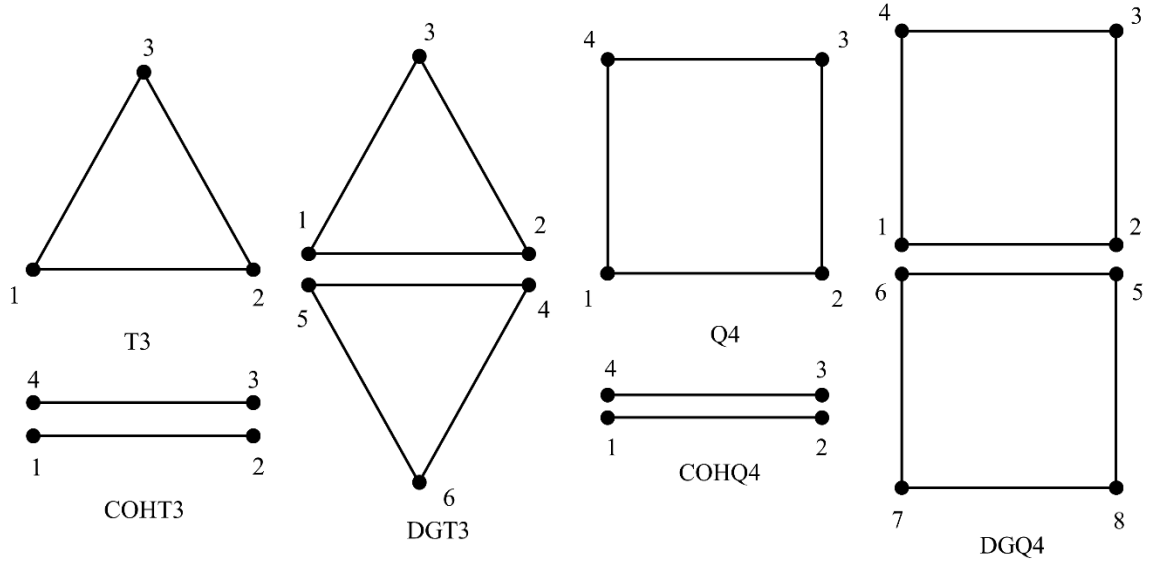
During the sixth phase in Box 6, the *coupler* connectivity is generated. All *node* duplication has been performed before this phase, such that all adjacency information in the mesh is prepared for *coupler* insertion. Similar to *elements*, each *coupler* type has a specific template for the arrangement of local *nodes* on either side of the cut *facet*. For cohesive zone modeling, typically only the *nodes* lying on the *interface* are required for the computations because these formulations are displacement-based. However, Discontinuous Galerkin (DG) methods usually require computations of stresses and strains along the *interface*, which involve displacement-gradient calculations using all of the *nodes* in the adjoining *elements*. Examples of templates for linear *elements* and *couplers* in 2D and 3D are shown in Figure 3; higher-order counterparts are naturally treated by adding mid-edge, mid-face, and interior *nodes*. Volume *elements* are denoted by letter(s) and the number of *nodes* per *element* such as “T3”, cohesive *couplers* are denoted by adding the prefix “COH” such as “COHQ4”, and DG *couplers* are denoted similarly as “DGHEX8”. Dissimilar *element* types across *couplers* are also supported, such as “COHT3Q4”, so long as the original mesh is conforming.

Box 6. Construct the connectivity of couplers

```
for each cut int
  for each fac in FacetsOnInterface(int,1:end)
    cou = cou + 1
    [ele1,locF1,ele2,locF2] = ElementsOnFacet(fac,1:4)
    CouplerSet = PermuteNodes(ele1,locF1,ele2,locF2,NodesOnElementNew,template)
    NodesOnCoupler(cou,1:end) = CouplerSet
```

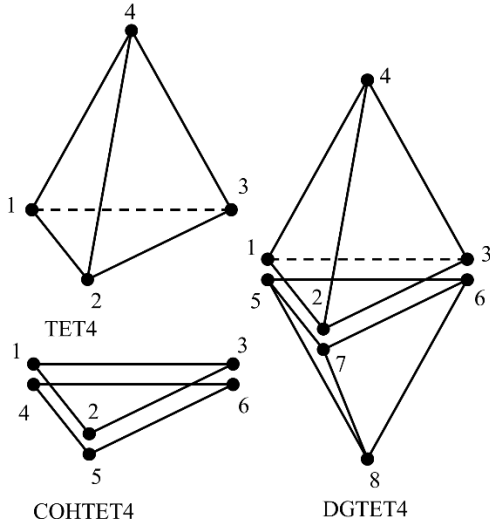
Using these *coupler* templates, the *nodes* from the parent *elements* on either side of a *coupler* are permuted to conform to the respective template. This permutation involves the renumbering of the local *nodes* so that the *element's* local *facet* is transformed to lie on the *coupler* surface. Note that the connectivity of the volume *element* is not modified, which would be impossible since a single *element* may have *couplers* on each of its *facets*. Also, the *coupler* connectivity does not at all influence the *node* duplication process. Hence, different *coupler* types may be inserted into the same mesh from phase 5, and indeed certain *interfaces* may be left without *couplers* in order to create traction-free *boundaries* within the mesh. The separation of *node* duplication and *coupler* insertion phases was also recommended in [17]. Finally, because the relation between *couplers*, *interfaces*, and *regions* is explicitly available, the assignment of different material properties to different *interfaces* is also trivial. As an example, for the modeling of crack propagation in multi-phase composites where each phase is represented as a *region*, different fracture toughness could be assigned to the *interfaces* lying between specific phases.

At the conclusion of the sixth phase using Box 6, the modified mesh containing *couplers* is suitable for analysis. Updates may also be required for boundary conditions or other sets relating to *nodes* and *elements*; the necessary mapping of old to new identifiers is explicitly stored to enable the updating of such sets. Other *element* types and *coupler* types could be added within this proposed framework so long as the convention that a *coupler* is adjacent to exactly two *elements* is maintained. Operations on 2D meshes could be trivially extended to manifolds (shells) in 3D by adding the third spatial coordinate to all *nodes*. Also, the version of the algorithm presented herein is skewed towards the explicit creation and storage of the topological relations in computer memory. Other approaches could be devised using implicit relations [22] if memory is limited.



(a)

(b)



(c)

(d)

Figure 3. Linear element and coupler templates: (a) triangular 2D element; (b) quadrilateral 2D element; (c) tetrahedral 3D element; (d) hexahedral 3D element

A realization of the proposed algorithm written in MATLAB[®] is provided <https://bitbucket.org/trusterresearchgroup/deiprogram>; the code has also been successfully tested in the GNU interpreted language Octave. Furthermore, the modified mesh corresponding to Figure 1 that is generated by this script is provided in Section 4.1.

4. Numerical Results

The following numerical examples verify that the proposed algorithm produces analysis suitable discontinuous meshes. Focus is placed upon the nodes which are duplicated and upon the zero-thickness interface elements (referred to as couplers) which are inserted. The efficiency and scaling properties of the algorithm are also investigated. All finite element simulations are performed using codes written in MATLAB[®] and FORTRAN that require as input only the traditional arrays of nodal coordinates and element connectivity, which ensures compatibility of the algorithm with commercial finite element codes.

4.1 Verification example from Section 2

The geometric and topological information corresponding to the mesh from Figure 1 is listed in Table 2 through Table 4. Providing these arrays as input to the program, the resulting discontinuous mesh is shown in Figure 4. The node duplication is carried out exactly as described in Section 3 to yield the numbering pattern that is shown. The mesh is intentionally expanded to show the location of the couplers and the duplicated nodes. Notice that four separate copies of node 5 appear and that all elements in region A have been completely disconnected.

Table 2. Contents of array Coordinates, transposed

Node	1	2	3	4	5	6	7	8	9
x	0.0	1.0	2.0	0.0	1.0	2.0	0.0	1.0	2.0
y	2.0	2.0	2.0	1.0	1.0	1.0	0.0	0.0	0.0

Table 3. Contents of array NodesOnElement, transposed

Element	a	b	c	d	e	f	g	h
Node	1	1	5	5	5	8	7	7
	4	5	3	6	9	9	8	5
	5	2	2	3	6	5	5	4

Table 4. Contents of array RegionsOnElement, transposed

Element	a	b	c	d	e	f	g	h
Region	A	B	C	B	D	D	E	A

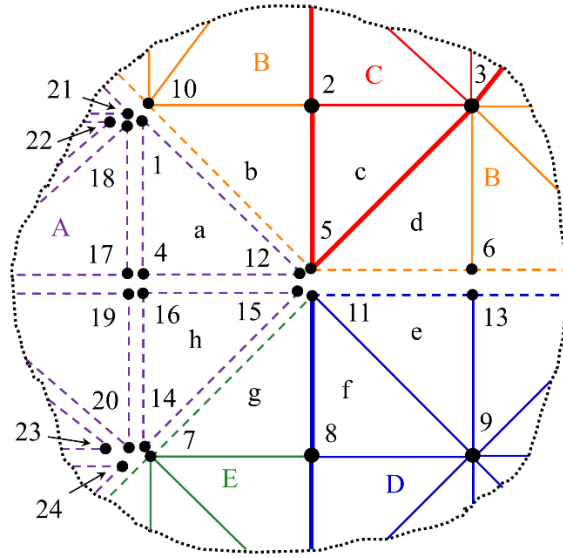


Figure 4. Resulting mesh after node duplication

4.2 Two dimensional patch test

The second numerical example further tests the ability of the algorithm to treat complex patterns of regions and interfaces. A rectangular 4 mm $\times$ 3 mm domain is considered with both linear triangular and linear quadrilateral elements, as shown in Figure 5 (a). The elements are grouped into a series of regions as indicated by numbers as well as the colors in the legend. Couplers are then inserted along various interfaces in the domain as well as within the intraface of region 6; the list of region pairs identifying the interfaces is given in Table 5. Each of the resulting portions of the domain have been separated in Figure 5 (b) in order to highlight the location of the couplers. This figure was created after the completion of the insertion algorithm; hence, the duplicated nodes allowed the nodal coordinates of different regions to be translated separately. Notice that the smaller mesh from Figure 1 is contained within the lower-left portion of this mesh, with the intersection of five regions at a single node. A motivating physical problem for this mesh would be a polycrystalline material with multiple grain boundaries that are primary sources for cracks as well as a single weak inclusion in which matrix cracking is expected.

Table 5. Region identifier pairs for interfaces

Interfaces	1,3	1,5	1,8	2,6	3,5	4,6	5,6	5,8
------------	-----	-----	-----	-----	-----	-----	-----	-----

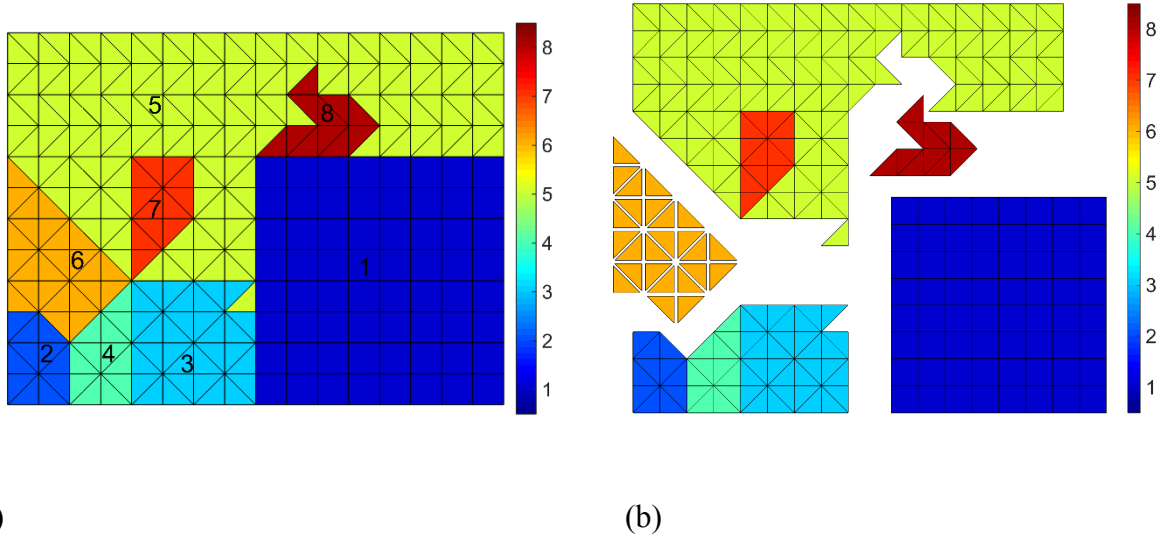


Figure 5. Mesh containing multiple regions: (a) continuous mesh; (b) discontinuous mesh with artificial separation indicating coupler locations

A patch test is then performed using the Discontinuous Galerkin formulation from [9, 10] to test the validity of the generated topology in Figure 5 (b). Plane stress conditions are considered, and each of the regions is assigned the material properties $E = 100,000$ MPa and $\nu = 0.25$. The horizontal displacement is prescribed as zero along the left boundary of the domain, and a tensile traction of $\Sigma = 2,500$ MPa is applied to the right boundary. These boundary conditions result in a 0.1 mm horizontal displacement of the right face. The computed nodal stress field is shown in Figure 6 on the deformed geometry. Observe that the stress field is essentially constant everywhere; also, no gaps or overlaps can be seen in the mesh. Therefore, we conclude that the DG couplers have been correctly inserted so that the patch test is satisfied by the variationally consistent DG method.

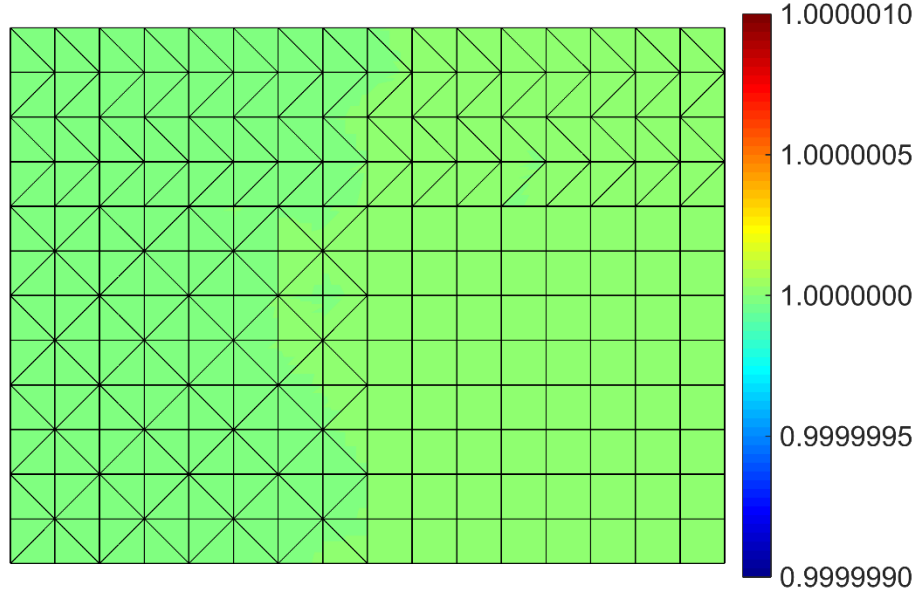


Figure 6. Normalized stress field σ_{xx}/Σ on deformed configuration for DG method

A second simulation is performed using cohesive zone (CZ) couplers in place of the DG couplers. The initial elastic stiffness of the cohesive zone is set to 50,000 MPa/mm in order to exaggerate the presence of the couplers in the mesh. Hence, the expected gap across the couplers under a stress of $\Sigma = 2,500$ MPa is 0.05 mm. Normal displacements of 0.1 mm are prescribed on the top and right domain boundaries, and symmetry boundary conditions are applied to the bottom and left surfaces. The normalized stress field σ_{xx}/Σ with $\Sigma = 2,500$ MPa is shown on

the deformed configuration in Figure 7; the deformations have been magnified by a factor of two. The zero-thickness elements are also visualized in the mesh as solid elements. Note that voids have appeared in the domain where multiple couplers intersect at multiple angles. Also, regions 6 and 8 are carrying relatively lower stresses due to the compliance of the interfaces so that the applied stress is redistributed throughout the domain. However, no discontinuities in displacement or stress are present between regions 5 and 7 since these interfaces were not decoupled and uniform elastic properties were employed; we remark that nodal stress projection has been applied within contiguous regions for both Figure 6 and Figure 7. Notice that the single triangular element of region 5 has been fully decoupled since all three of its facets are part of the cut interface(1,5) and interface(3,5). Thus, these results highlight that the mesh produced by the proposed algorithm is suitable for CZ modeling as well.

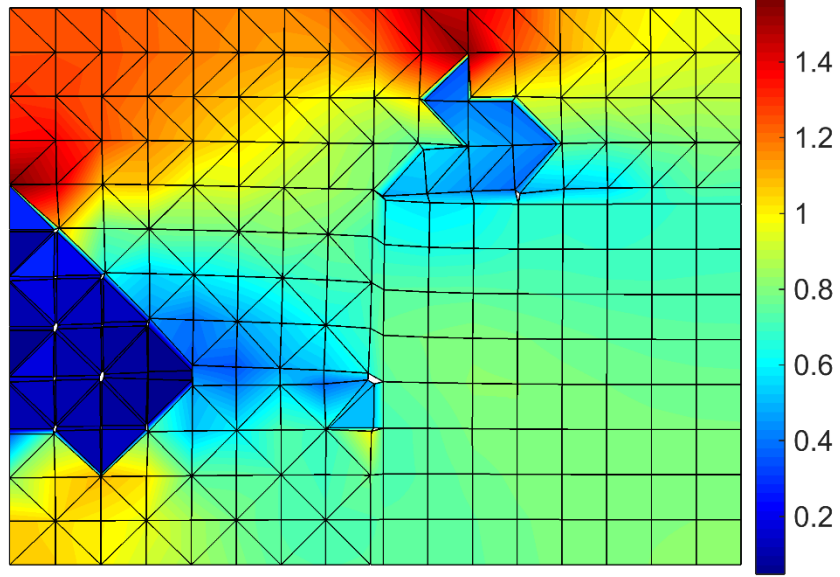


Figure 7. Normalized stress field σ_{xx}/Σ on deformed configuration for intrinsic CZ method

4.3 Three dimensional scalability study

A numerical scalability study is performed on a geometry of practical engineering significance. The domain is a representative volume element (RVE) of a polycrystalline material that contains 100 grains with conforming grain boundaries, as shown in Figure 8. The meshes of 10-node quadratic tetrahedral elements were generated using the open-source software Neper [23]. Three levels of mesh refinement are considered, and the number of nodes and elements in each mesh is listed in Table 6. Because each mesh is regenerated by the program, the number of elements does not exactly increase by a multiple of eight. Each mesh contains non-uniform topology; for example, the number of elements attached to a node varies between 1 and 40 on the coarse mesh.

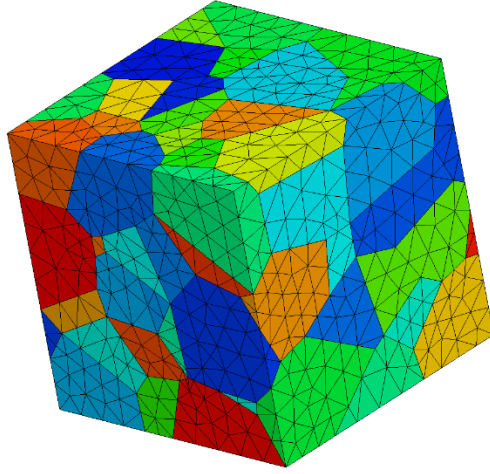


Figure 8. Polycrystalline material domain discretized with quadratic tetrahedral elements

Our objective is to assess the average runtime of the algorithm for inserting the couplers similar to the study in [13]. Two scenarios are performed: the insertion of couplers only along interfaces between the grains, and the insertion along all interface and intraface facets in the mesh. The timings were performed using the MATLAB[®] implementation of the algorithm on a serial desktop computer; the recorded time is for each of the six phases from Section 3 not including the mesh loading time. Random orderings of the nodes and elements were employed as input to the script. The total execution time for the script on each mesh is reported in Table 6, which is the average of three analyses. Note that patch tests have also been conducted on this three dimensional geometry using the DG method [9, 10] to verify that the modified meshes are analysis suitable.

Table 6. Mesh statistics and elapsed time for inserting interface and interface couplers

Mesh	Elements	Nodes	Interface Couplers	Interface Time (s)	All Couplers	Interface Fraction	Total Time (s)
Coarse	10,402	15,761	3,921	7.55	19,896	0.1971	19.5
Medium	68,713	97,720	12,111	36.2	134,474	0.0901	118
Fine	539,293	742,167	42,220	237	1,067,885	0.0395	797

Approximate linear scaling of the computing time is observed with respect to the number of elements in the mesh or the number of couplers inserted. Similar performance was obtained for the adaptive coupler insertion in [13]. Also, the recorded times for the proposed algorithm are on the same order of magnitude as reported therein. We expect that the runtimes could be slightly improved by using a compiled language rather than a scripting language. Also, an accounting for

the time expended during each phase is presented in Table 7 for one analysis of the coarse mesh. Clearly, phase 4 for duplicating nodes using the sector approach is the most expensive operation of the algorithm, followed by the generation of coupler connectivity in phase 6. Substantial speedup is achieved by avoiding these operations for the duplication on intrafaces during phase 5. By comparing Table 6 and Table 7, we conclude that approximately five to ten times as many nodes are duplicated within one tenth the time in phase 5 compared to phase 4. These cost savings are dependent upon the ratio of interface to intraface facets in the mesh. In general, the insertion algorithm is seen to possess optimal scaling properties.

Table 7. Execution time (seconds) for each algorithmic phase for the coarse mesh

Phase 1	Phase 2	Phase 3	Phase 4	Phase 5	Phase 6 (Interface)	Phase 6 (All)
0.182	2.797	0.778	8.675	0.680	0.497	6.302

5. Conclusion

A topological algorithm was presented for inserting zero-thickness interface elements, referred to as “couplers”, into conforming finite element meshes. The method is applicable for a wide class of problems involving: (i) intrinsic cohesive zone modeling, Discontinuous Galerkin formulations, or other interface methods; (ii) linear and higher order finite elements of various shapes; and (iii) two and three dimensional domains. The key feature of the method is use of “regions” of elements in the domain to indicate the locations for insertion, where the regions may be purely geometrical or may be a collection of elements with common material properties. Collections of couplers may be designated along interface facets of elements between regions or along intraface facets of elements within a region. Using as input only the mesh connectivity and the region designations, appropriate couplers are inserted and nodes are duplicated using topological operations alone. Furthermore, the concept of element sectors surrounding a node is introduced in order to ensure that the proper level of interpolation continuity is preserved during the insertion process. The phases of inserting interface couplers and intraface couplers are distinguished for increased computational efficiency. Numerical tests for two and three dimensional problems verify that the algorithm scales linearly with the number of elements and produces correct patterns of node duplication and coupler insertion to retain desirable continuous features in the domain. In particular, the algorithm has proved to be suitable for complex three dimensional meshes.

Acknowledgements

T. Truster was supported by a subcontract through the project DE-AC05-000R22725 at Oak Ridge National Laboratory. This support is gratefully acknowledged.

References

1. Pandolfi A, Ortiz M. Solid modeling aspects of three-dimensional fragmentation. *Engineering with Computers* 1998; **14**(4):287-308.
2. Radovitzky R, Seagraves A, Tupek M, Noels L. A scalable 3D fracture and fragmentation algorithm based on a hybrid, discontinuous Galerkin, cohesive element method. *Computer Methods in Applied Mechanics and Engineering* 2011; **200**(1-4):326-344.
3. Alfano G, Crisfield MA. Finite element interface models for the delamination analysis of laminated composites: mechanical and computational issues. *International Journal for Numerical Methods in Engineering* 2001; **50**(7):1701-1736.
4. Raghavan P, Ghosh S. A continuum damage mechanics model for unidirectional composites undergoing interfacial debonding. *Mechanics of Materials* 2005; **37**:955-979.
5. Truster TJ, Masud A. A Discontinuous/continuous Galerkin method for modeling of interphase damage in fibrous composite systems. *Computational Mechanics* 2013; **52**(3):499-514.
6. Chandra N, Li H, Shet C, Ghonem H. Some issues in the application of cohesive zone models for metal-ceramic interfaces. *International Journal of Solids and Structures* 2002; **39**(10):2827-2855.
7. Arnold DN, Brezzi F, Cockburn B, Marini LD. Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM Journal on Numerical Analysis* 2002; **39**(5):1749-1779.
8. Liu R, Wheeler MF, Dawson CN. A three-dimensional nodal-based implementation of a family of discontinuous Galerkin methods for elasticity problems. *Computers & Structures* 2009; **87**(3):141-150.
9. Truster TJ, Masud A. Primal interface formulation for coupling multiple PDEs: A consistent derivation via the Variational Multiscale method. *Computer Methods in Applied Mechanics and Engineering* 2014; **268**:194-224.
10. Masud A, Truster TJ, Bergman LA. A unified formulation for interface coupling and frictional contact modeling with embedded error estimation. *International Journal for Numerical Methods in Engineering* 2012; **92**(2):141-177.
11. Pandolfi A, Ortiz M. An Efficient Adaptive procedure for three-dimensional fragmentation simulations. *Engineering with Computers* 2002; **18**(2):148-159.
12. Ortiz M, Pandolfi A. Finite-deformation irreversible cohesive elements for three-dimensional crack-propagation analysis. *International Journal for Numerical Methods in Engineering* 1999; **44**(9):1267-1282.
13. Paulino GH, Celes W, Espinha R, Zhang Z. A general topology-based framework for adaptive insertion of cohesive elements in finite element meshes. *Engineering with Computers* 2007; **24**(1):59-78.
14. Dooley I, Mangala S, Kale L, Geubelle P. Parallel Simulations of Dynamic Fracture

- Using Extrinsic Cohesive Elements. *Journal of Scientific Computing* 2008; **39**(1):144-165.
15. Espinha R, Celes W, Rodriguez N, Paulino GH. ParTopS: compact topological framework for parallel fragmentation simulations. *Engineering with Computers* 2009; **25**(4):345-365.
 16. Alhadeff A, Celes W, Paulino GH. Mapping Cohesive Fracture and Fragmentation Simulations to Graphics Processor Units. *International Journal for Numerical Methods in Engineering* 2015, doi 10.1002/nme.4842.
 17. Nguyen VP. An open source program to generate zero-thickness cohesive interface elements. *Advances in Engineering Software* 2014; **74**:27-39.
 18. Nguyen VP. Discontinuous Galerkin/extrinsic cohesive zone modeling: Implementation caveats and applications in computational fracture mechanics. *Engineering Fracture Mechanics* 2014; **128**:37-68.
 19. Needleman A. A continuum model for void nucleation by inclusion debonding. *J. Appl. Mech.* 1987; **54**(3):525.
 20. Park K, Paulino GH. Computational implementation of the PPR potential-based cohesive model in ABAQUS: Educational perspective. *Engineering Fracture Mechanics* 2012; **93**:239-262.
 21. Beall MW, Shephard MS. A general topology-based mesh data structure. *International Journal for Numerical Methods in Engineering* 1997; **40**(9):1573-1596.
 22. Celes W, Paulino GH, Espinha R. A compact adjacency-based topological data structure for finite element mesh representation. *International Journal for Numerical Methods in Engineering* 2005; **64**(11):1529-1556.
 23. Quey R, Dawson PR, Barbe F. Large-scale 3D random polycrystals for the finite element method: Generation, meshing and remeshing. *Computer Methods in Applied Mechanics and Engineering* 2011; **200**(17–20):1729-1745.