



## Original software publication

## Neper2CAE and PyCiGen: Scripts to generate polycrystals and interface elements in Abaqus

Nicolò Grilli<sup>a,\*</sup>, Edmund Tarleton<sup>a,b</sup>, Alan C.F. Cocks<sup>a</sup><sup>a</sup> Department of Engineering Science, University of Oxford, Parks Road, OX1 3PJ, UK<sup>b</sup> Department of Materials, University of Oxford, Parks Road, OX1 3PH, UK

## ARTICLE INFO

## Article history:

Received 8 September 2020

Received in revised form 24 November 2020

Accepted 20 December 2020

## Keywords:

Finite element method

Fracture mechanics

Polycrystals

Abaqus

Neper

## ABSTRACT

Two codes have been developed: one allows columnar polycrystal geometries generated by software such as Neper or DREAM3D to be imported into the Complete Abaqus Environment for further geometric manipulation and meshing with arbitrary element types; the other code generates zero thickness, four noded interface elements at the boundaries between the crystals. Together these allow intergranular fracture simulations to be performed easily in Abaqus. This approach can automatically create different grain structures and allows the mechanical properties of metals and other crystalline materials to be simulated. This is important as it enables the sample to sample variability of fracture and mechanical properties that is observed in experiments to be reproduced. These scripts extend the simulation capabilities of the finite element solver Abaqus.

© 2020 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

## Code metadata

Current code version	V1.0.1
Permanent link to code	<a href="https://github.com/ElsevierSoftwareX/SOFTX-D-20-00041">https://github.com/ElsevierSoftwareX/SOFTX-D-20-00041</a>
Code Ocean compute capsule	N/A
Legal Code License	LGPL-2.1License
Code versioning system used	git
Software code languages	Python
Compilation requirements	Python packages listed in requirements
Link to developer documentation	<a href="https://github.com/ngriлли/PyCiGen">https://github.com/ngriлли/PyCiGen</a>
Support email for questions	<a href="mailto:nicolo.grilli@eng.ox.ac.uk">nicolo.grilli@eng.ox.ac.uk</a>

## Software metadata

Current code version	V1.0.1
Permanent link to code	<a href="https://github.com/ngriлли/Neper2CAE">https://github.com/ngriлли/Neper2CAE</a>
Code Ocean compute capsule	N/A
Legal Code License	LGPL-2.1License
Code versioning system used	git
Software code languages	Python
Compilation requirements	Neper 4.0.2, Abaqus 2016 and Python packages listed in requirements
Link to developer documentation	<a href="https://github.com/ngriлли/Neper2CAE">https://github.com/ngriлли/Neper2CAE</a>
Support email for questions	<a href="mailto:nicolo.grilli@eng.ox.ac.uk">nicolo.grilli@eng.ox.ac.uk</a>

## 1. Motivation and significance

To simulate the mechanical properties of metals requires analysis of the elastic and plastic response of the microstructure [1]. Metals consist of grains with a crystalline structure, whose dimensions can vary from nanometres to millimetres. Each grain

\* Corresponding author.

E-mail addresses: [nicolo.grilli@eng.ox.ac.uk](mailto:nicolo.grilli@eng.ox.ac.uk) (N. Grilli), [edmund.tarleton@eng.ox.ac.uk](mailto:edmund.tarleton@eng.ox.ac.uk) (E. Tarleton), [alan.cocks@eng.ox.ac.uk](mailto:alan.cocks@eng.ox.ac.uk) (A.C.F. Cocks).

has a specific shape and orientation. The simulation of mechanical tests can be carried out using finite element solvers, such as Abaqus [2]. User material (UMAT) subroutines can be used to model the mechanical properties of different grains [3,4]. Cohesive interfaces implemented through a zero thickness user element (UEL) subroutine are used to model fracture nucleation and propagation [5–7].

Software for the generation of polycrystal geometries are available, such as Neper [8] and DREAM3D [9]. They can generate polycrystals with arbitrary grain sizes, distributions and orientations. They can also import experimental images. Meshing capabilities of the polycrystal geometry are usually included in the above-mentioned software. However, the geometric shapes that can be generated and meshing options are limited. Specifically, a graphical user interface that allows a wide variety of geometrical manipulation options is not available.

Usually, only parallelepiped geometries with four node tetrahedral elements can be generated. Hexahedral elements are preferred to tetrahedral ones, which give an excessively stiff response during bending [10]. Four node tetrahedral elements exhibit the so-called volumetric locking [11]. This numerical phenomenon takes place when the displacement field has nearly zero volumetric strain. The shape functions of tetrahedral elements are less accurate in capturing those deformations and this results in pairs of neighbouring elements carrying positive and negative volumetric strains respectively. Therefore, spurious high stresses are introduced. This is particularly true for crystal plasticity constitutive models, in which the plastic deformation is modelled as an isochoric process. [12–14].

Abaqus CAE (Complete Abaqus Environment) offers more advanced capabilities. The geometry can be modified using extrusions of arbitrary shapes and multiple parts can be assembled in a model, therefore samples with arbitrary shapes can be reproduced. Moreover, internal voids can be introduced. If external devices apply load to the sample, they can be included in the model. Arbitrary boundary conditions and contact between parts can be introduced. The mesh generation features of Abaqus CAE are also more advanced. Different element types and meshing algorithms can be used.

Therefore, linking polycrystal generation software, such as Neper, with Abaqus CAE represents a strong need for the scientific and engineering community. Such a link is provided by the program Neper2CAE, described in this manuscript.

Automated generation of polycrystal geometries is also useful when running a set of simulations with different grain size and shapes. Thus, the sample to sample variability of fracture and mechanical properties can be understood and safety limits can be established during component design.

Grain boundaries, which are the 2D interfaces between neighbouring grains, provide crack nucleation and propagation sites in a range of metals [15]. This phenomenon can be modelled using interface elements, whose mechanical properties are described by a traction–separation law [16]. Interface elements constitute a 2D mesh on the grain boundaries. However, commercial finite element software does not offer the possibility to automatically generate zero thickness cohesive elements. Available software for polycrystal generation can create only triangular interface elements. If hexahedral elements are used inside the grains, quadrilateral interface elements must be used. Therefore, an additional program is needed, which is PyCiGen, described in this manuscript.

The software has been developed specifically for polycrystalline metals, but its applicability can be easily extended to different materials with interfaces, e.g. composite materials.

## 2. Software description: polycrystal generation

In this section, the main Abaqus scripting commands used for the polycrystal generation are reported. A Neper parser (.tess file) is also present in the code developed, but it is not described in detail in the manuscript and the reader is referred to the code repository.

A 2D polycrystal is generated using Neper [8], as shown in Fig. 1. The generated .tess file contains the coordinates ( $x, y$ ) of the vertices  $v_i$  of the grains after the **\*\*vertex** keyword. These are introduced in the upper surface of the Abaqus geometry using the command:

```
Part.DatumPointByCoordinate(coords = (x, y, z))
```

where  $z$  is the user-defined depth of the parallelepiped geometry along the  $z$  axis.

The edges that connect two vertices  $v_i$  and  $v_j$  of the grains are described after the keyword **\*\*edge** in the .tess file. They are included in the Abaqus geometry as partitions of the upper surface using the command:

```
Part.PartitionFaceByShortestPath
(point1= $v_i$ , point2= $v_j$ , faces=pickedFaces)
```

A 2D grain structure is therefore generated on the upper surface, as shown by the grain inside the red dashed square in Fig. 1.

The geometry is partitioned into columnar grains by sweeping the edges  $e_i$ , generated on the upper surface, along a sweep direction, which correspond to the negative  $z$  direction, as shown in Fig. 1. This is done using the command:

```
Part.PartitionCellByExtrudeEdge
(line=sweepDir, cells=pickedCells, edges= $e_i$ , sense=REVERSE)
```

At this point, one cell for each grain is present. A user material is defined using:

```
Model.materials[grainName].UserMaterial
(mechanicalConstants=mechConstTuple)
```

The mechanical constants indicate the orientation of the grain and the material type. In this way, dual-phase materials, with grains constituted of different crystal structures, can be included. The corresponding solid section is created and assigned to each grain using:

```
Model.HomogeneousSolidSection
(name=grainName, material=grainName, thickness=None)
```

```
p.SectionAssignment
(region=regionName, sectionName=grainName, offset=0.0,
offsetType=MIDDLE_SURFACE, offsetField="", thicknessAssignment=FROM_SECTION)
```

Finally the Abaqus model is ready and can be transformed using the Abaqus CAE capabilities. Boundary conditions can be assigned and a mesh can be created.

**Usage instructions:** in the file `main.py`, the following parameters must be set: width, height and depth. These will be the dimensions of the parallelepiped geometry along the  $x$ ,  $y$  and  $z$  axes generated in Abaqus. The parameter determining the number of grains `Ngrains` must also be set. The code is then executed by:

```
python3.6 main.py
```

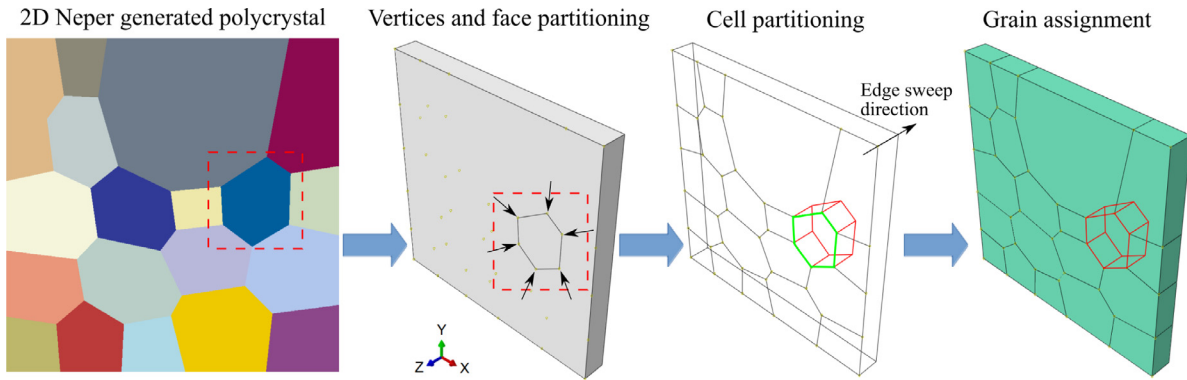


Fig. 1. Steps of the polycrystal generation algorithm.

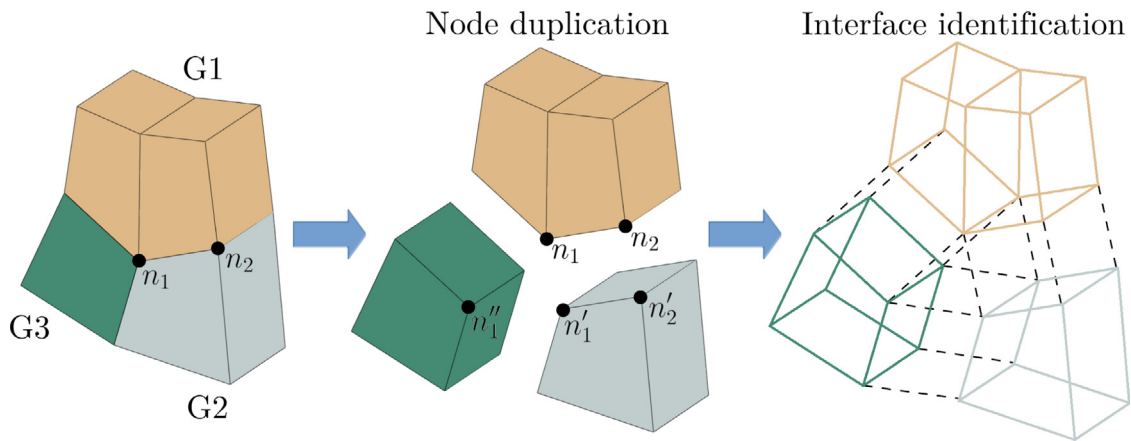


Fig. 2. Steps of the interface elements generation.

### 3. Software description: interface elements generation

In this section, a mesh of hexahedral elements is considered where each bulk element has 8 nodes, corresponding to the vertices of the hexahedron. Adjacent elements are connected by a face, which contains 4 nodes. Each node and each element is characterised by a unique positive integer index.

First a data structure called “connectivity” is constructed that stores the indices of all bulk elements that contain a particular node  $n_i$ . Bulk element sets are built by Abaqus that contain all the elements belonging to a certain grain.

Interface elements are built at the interface between grains. Therefore, the nodes that are at those interfaces have to be identified. For instance, in Fig. 2, the node  $n_1$  belongs to three different grains (G1, G2 and G3), while node  $n_2$  belongs to two different grains (G1 and G2).

In order to identify interface nodes, the “connectivity” is used: given a node  $n_i$ , the bulk elements that contain that node and the grains to which they belong can be found. If the bulk elements that contain  $n_i$  belong to different grains, then  $n_i$  is an interface node.

Adjacent bulk elements at the grain boundary are identified as elements containing interface nodes. “Face” objects are created at the interface between two adjacent bulk elements. The pairs of adjacent bulk elements are identified by checking if they share four interface nodes. In Fig. 2, each “face” object is represented by four dashed lines connecting nodes of bulk elements in different grains.

Interface nodes are multiplied. For instance, in Fig. 2, the node  $n_1$  is triplicated because it belongs to three different grains. Two new nodes  $n'_1$  and  $n''_1$  are created. The node  $n_2$  is duplicated and a

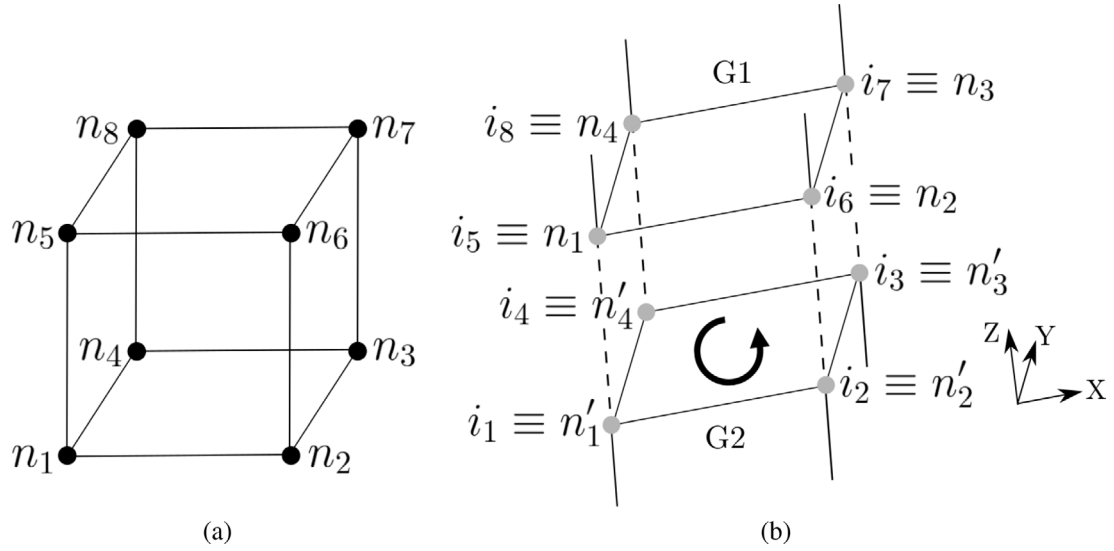
new node  $n'_2$  is created. Each duplicated node contains a variable that points to the original node. Each duplicated node is assigned to a different bulk element belonging to a different grain. For instance, in Fig. 2,  $n_1$ ,  $n'_1$  and  $n''_1$  are assigned to three different grains. After this operation, there are no more interface nodes: each node belongs to a unique grain.

At this point, the face object contains information about the 8 nodes that will form the interface element and information about the 2 adjacent bulk elements. The first 4 nodes ( $i_1, i_2, i_3, i_4$ ) of the interface element belong to one grain and the others ( $i_5, i_6, i_7, i_8$ ) belong to the other grain.

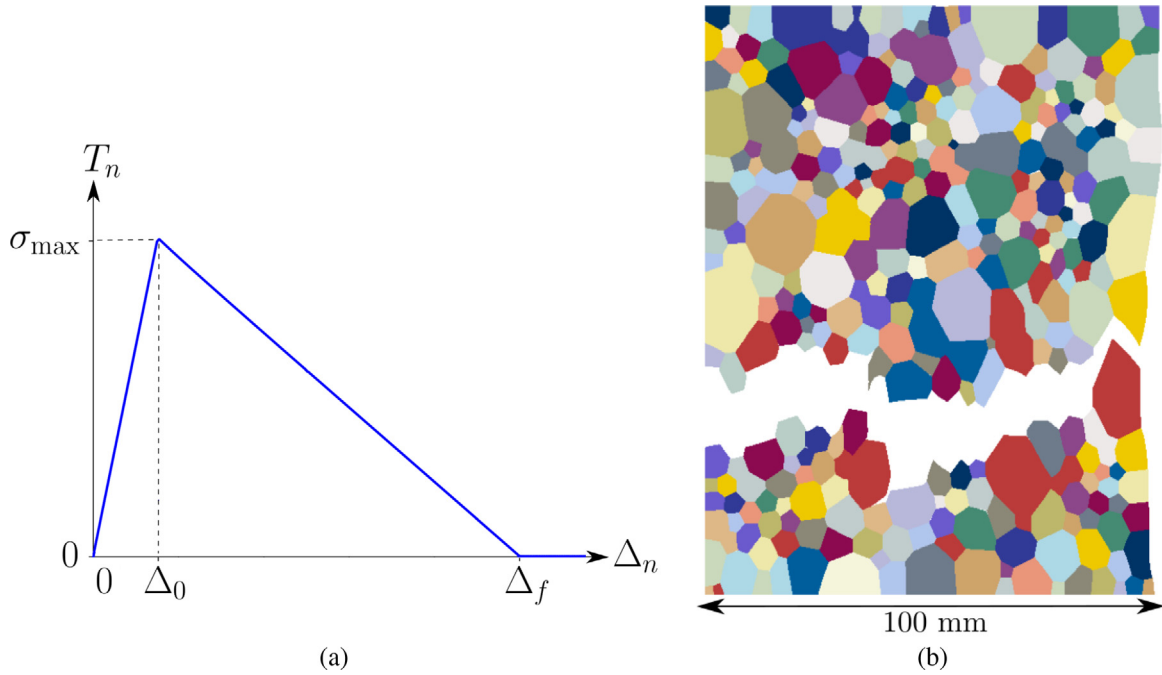
In a finite element simulation, a deformation vector is associated to each node and a traction-separation law is used, which is based on that deformation. Therefore, the order of the nodes in the interface element is important. At zero deformation, node  $i_1$  coincides with  $i_5$ , node  $i_2$  with  $i_6$ , node  $i_3$  with  $i_7$ , node  $i_4$  with  $i_8$ , as shown in Fig. 3(b). The distances between these pairs of nodes determine the interface separation.

A right-handed reference system XYZ is built, as shown in Fig. 3(b). X points from  $i_1$  to  $i_2$ , Y is perpendicular to X and belongs to the plane containing the nodes ( $i_1, i_2, i_3, i_4$ ). The Z axis points from grain 2 (G2) to grain 1 (G1) in Fig. 3(b). The direction of Z is important for the finite element solver because it discriminates between opening and closing of the cohesive interface. In Fig. 3(b), the quadrilateral constituted by the nodes ( $i_5, i_6, i_7, i_8$ ) has moved towards the positive Z direction, indicating that the cohesive interface is opening. Z must always point from the grain that contains the interface nodes ( $i_1, i_2, i_3, i_4$ ) to the grain that contains the interface nodes ( $i_5, i_6, i_7, i_8$ ).

To determine the correct direction of Z and the correct reference system XYZ, it is important to associate properly the



**Fig. 3.** Node numbering of (a) bulk elements and (b) interface elements.



**Fig. 4.** (a) Traction–separation law that relates the traction force with the separation between the nodes of the interface elements. (b) Fracture along an arbitrary path constituted of multiple grain boundaries.

nodes of the interface element ( $i_1, i_2, i_3, i_4$ ) with the nodes of the corresponding bulk element. The criterion is based on the right-hand rule: if the right hand fingers curl along the path  $i_1 \rightarrow i_2 \rightarrow i_3 \rightarrow i_4$ , the thumb must point towards the other grain. This direction determines the Z axis. For the other adjacent bulk element if the fingers curl along the path  $i_5 \rightarrow i_6 \rightarrow i_7 \rightarrow i_8$ , the right thumb must also point along Z towards the grain containing ( $i_5, i_6, i_7, i_8$ ).

This association is done by using the following property of the hexahedral elements generated by Abaqus: the elements can be arbitrarily oriented in space but the node numbering is always the one represented in Fig. 3(a). Six possible faces could be associated with interface elements. The face object contains information about the adjacent bulk elements. Therefore, a “face-type” variable is defined that contains the information about the bulk element face that lies on the grain boundary.

For instance, if a bulk element face with nodes ( $n_1, n_2, n_3, n_4$ ) is associated with the interface element nodes ( $i_1, i_2, i_3, i_4$ ), the right-hand rule explained above applies. The correspondence between nodes of the bulk elements and nodes of the interface elements will be:  $i_1 \equiv n_4$ ,  $i_2 \equiv n_3$ ,  $i_3 \equiv n_2$ ,  $i_4 \equiv n_1$ . If the same bulk element face is associated with the interface element nodes ( $i_5, i_6, i_7, i_8$ ), then the correspondence is:  $i_5 \equiv n_1$ ,  $i_6 \equiv n_2$ ,  $i_7 \equiv n_3$ ,  $i_8 \equiv n_4$ . This is shown in Fig. 3(b) for the bulk element in grain 1 (G1).

For the six face-types, the correspondence between nodes of the interface elements and nodes of the bulk elements is reported in Table 1.

The behaviour of the interface elements has been studied by simulating two bulk elements and one interface element at the common interface between them. Fig. 4(a) shows the result of this test. The plot shows the traction on the interface element as



**Table 1**  
Possible correspondences between bulk element nodes and interface element nodes.

$(i_1, i_2, i_3, i_4)$	$(i_5, i_6, i_7, i_8)$
$(n_4, n_3, n_2, n_1)$	$(n_1, n_2, n_3, n_4)$
$(n_5, n_6, n_7, n_8)$	$(n_8, n_7, n_6, n_5)$
$(n_2, n_3, n_7, n_6)$	$(n_6, n_7, n_3, n_2)$
$(n_1, n_5, n_8, n_4)$	$(n_4, n_8, n_5, n_1)$
$(n_1, n_2, n_6, n_5)$	$(n_5, n_6, n_2, n_1)$
$(n_3, n_4, n_8, n_7)$	$(n_7, n_8, n_4, n_3)$

a function of the separation of the nodes. The maximum traction  $\sigma_{\max}$  is reached at separation  $\Delta_0$  and zero traction is obtained at separation  $\Delta_f$ . Fig. 4(b) shows a more extensive example using the traction law in Fig. 4(a). In this case the fracture along an arbitrary path constituted of multiple grain boundaries is shown.

**Usage instructions:** the mesh file created using the polycrystal model generated by Neper2CAE and named Job-1.inp must be added to the code folder. The code is then executed by:

```
python3.6 main.py
```

The interface elements will be saved in the file Job-1-int-elems.inp.

#### 4. Conclusions

Two programs have been developed to generate polycrystal representative volumes and interface elements at the grain boundaries. These features are not available in Abaqus CAE but they are normally required to model mechanical properties at the micrometre length scale.

The algorithm for interface element generation is of general interest. It represents an example of interface detection in a mesh. The strategy used to find the orientation of the interface elements, described in Section 3, is based on the ordering of the interface element nodes and on the specific correspondence between interface element nodes and bulk element nodes. It can be used to find the directions pointing towards the inside and outside of a generic element set.

This software has been used in a recent study on the intergranular fracture of cast  $\alpha$ -uranium [17,18] and it is available in the following repositories [19,20]. The implementation of different types of interface elements can be a future improvement of PyCiGen.

#### Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

#### Acknowledgements

The authors acknowledge financial support from AWE, UK plc for this research, program manager: Dr John Askew. ET acknowledges support from the Engineering and Physical Sciences Research Council, UK under Fellowship grant EP/N007239/1.

#### References

- [1] Roters F, Eisenlohr P, Bieler TR, Raabe D. Crystal plasticity finite element methods in materials science and engineering. Wiley; 2010, URL <https://onlinelibrary.wiley.com/doi/book/10.1002/9783527631483>.
- [2] Smith M. ABAQUS/Standard user's manual, version 6.9. United States: Dassault Systèmes Simulia Corp; 2009.
- [3] Roters F, Diehl M, Shanthraj P, Eisenlohr P, Reuber C, Wong S, et al. DAMASK – the Düsseldorf advanced material simulation kit for modeling multi-physics crystal plasticity, thermal, and damage phenomena from the single crystal up to the component scale. Comput Mater Sci 2018. <http://dx.doi.org/10.1016/j.commatsci.2018.04.030>.
- [4] Dunne F, Rugg D, Walker A. Lengthscale-dependent, elastically anisotropic, physically-based hcp crystal plasticity: Application to cold-dwell fatigue in Ti alloys. Int J Plast 2007;23(6):1061–83. <http://dx.doi.org/10.1016/j.ijplas.2006.10.013>, URL <http://www.sciencedirect.com/science/article/pii/S0749641906001641>.
- [5] Camacho G, Ortiz M. Computational modelling of impact damage in brittle materials. Int J Solids Struct 1996;33(20):2899–938. [http://dx.doi.org/10.1016/0020-7683\(95\)00255-3](http://dx.doi.org/10.1016/0020-7683(95)00255-3).
- [6] Ortiz M, Pandolfi A. Finite deformation irreversible cohesive elements for three dimensional crack propagation analysis. Internat J Numer Methods Engrg 1999;44(9):1267–82. [http://dx.doi.org/10.1002/\(SICI\)1097-0207\(19990330\)44:9<1267::AID-NME486>3.0.CO;2-7](http://dx.doi.org/10.1002/(SICI)1097-0207(19990330)44:9<1267::AID-NME486>3.0.CO;2-7).
- [7] Elmukashfi E, Tarleton E, Cocks ACF. A modelling framework for coupled hydrogen diffusion and mechanical behaviour of engineering components. Comput Mech 2020;66(1):189–220. <http://dx.doi.org/10.1007/s00466-020-01847-9>, URL <https://link.springer.com/article/10.1007/s00466-020-01847-9#citeas>.
- [8] Quey R, Dawson P, Barbe F. Large-scale 3D random polycrystals for the finite element method: Generation, meshing and remeshing. Comput Methods Appl Mech Engrg 2011;200(17):1729–45. <http://dx.doi.org/10.1016/j.cma.2011.01.002>, URL <http://www.sciencedirect.com/science/article/pii/S004578251100003X>.
- [9] Diehl M, Groeber M, Haase C, Molodov DA, Roters F, Raabe D. Identifying structure–property relationships through dream.3D representative volume elements and DAMASK crystal plasticity simulations: An integrated computational materials engineering approach. JOM 2017;69:848–55. <http://dx.doi.org/10.1007/s11837-017-2303-0>, URL <https://link.springer.com/article/10.1007/s11837-017-2303-0>.
- [10] Cheng J, Shahba A, Ghosh S. Stabilized tetrahedral elements for crystal plasticity finite element analysis overcoming volumetric locking. Comput Mech 2016;57:733–53. <http://dx.doi.org/10.1007/s00466-016-1258-2>.
- [11] Dolbow J, Belytschko T. Volumetric locking in the element free Galerkin method. Internat J Numer Methods Engrg 1999;46(6):925–42. [http://dx.doi.org/10.1002/\(SICI\)1097-0207\(19991030\)46:6<925::AID-NME729>3.0.CO;2-Y](http://dx.doi.org/10.1002/(SICI)1097-0207(19991030)46:6<925::AID-NME729>3.0.CO;2-Y).
- [12] Grilli N, Cocks AC, Tarleton E. Crystal plasticity finite element modelling of coarse-grained  $\alpha$ -uranium. Comput Mater Sci 2020;171:109276. <http://dx.doi.org/10.1016/j.commatsci.2019.109276>, URL <http://www.sciencedirect.com/science/article/pii/S0927025619305750>.
- [13] Irastorza-Landa A, Grilli N, Van Swygenhoven H. Laue micro-diffraction and crystal plasticity finite element simulations to reveal a vein structure in fatigued Cu. J Mech Phys Solids 2017;104:157–71. <http://dx.doi.org/10.1016/j.jmps.2017.04.010>, URL <http://www.sciencedirect.com/science/article/pii/S0022509616304781>.
- [14] Grilli N, Cocks A, Tarleton E. Crystal plasticity finite element simulations of cast  $\alpha$ -uranium. In: Onate E, Owen DRJ, Peric D, Chiumenti M, editors. Computational plasticity XV: fundamentals and applications, 2019. 15th international conference on computational plasticity - fundamentals and applications. 2019.
- [15] Powell GW. The fractography of casting alloys. Mater Charact 1994;33(3):275–93. [http://dx.doi.org/10.1016/1044-5803\(94\)90048-5](http://dx.doi.org/10.1016/1044-5803(94)90048-5).
- [16] Barenblatt G. In: Dryden H, [von Kármán] T, Kuerti G, [van den Dungen] F, Howarth L, editors. The mathematical theory of equilibrium cracks in brittle fracture. Advances in applied mechanics, vol. 7, Elsevier; 1962, p. 55–129. [http://dx.doi.org/10.1016/S0065-2156\(08\)70121-2](http://dx.doi.org/10.1016/S0065-2156(08)70121-2).
- [17] Grilli N, Cocks AC, Tarleton E. Coupling a discrete twin model with cohesive elements to understand twin-induced fracture. Int J Fracture [in press].
- [18] Grilli N, Cocks AC, Tarleton E. Modelling the nucleation and propagation of cracks at twin boundaries [in preparation].
- [19] Grilli N. Neper2CAE. GitHub; 2020. <https://github.com/ngrilli/Neper2CAE>.
- [20] Grilli N. PyCiGen. GitHub; 2020. <https://github.com/ngrilli/PyCiGen>.