VorTess

Generation of 2-D random Poisson-Voronoi mosaics as framework for the micromechanical modelling of polycrystalline materials

— algorithm and subroutines description —

H. Riesch-Oppermann

Institut für Materialforschung

Forschungszentrum Karlsruhe GmbH, Karlsruhe

1999
Als Manuskript gedruckt
Für diesen Bericht behalten wir uns alle Rechte vor

Forschungszentrum Karlsruhe
Mitglied der Hermann von Helmholtz-Gemeinschaft
Deutscher Forschungszentren (HGF)

ISSN 0947-8620
Abstract

The present report contains a code and algorithm description of the code VorTess. Purpose of VorTess is to provide a framework for the stochastic description of polycrystalline materials on the basis of the grain and grain boundary structure. A 2-D random Poisson-Voronoi tessellation is generated by the code and handling of evolving crack patterns is done by specifically tailored crack extraction subroutines. Interfaces for data storage and retrieval of tessellations and related crack patterns allow easy coupling with advanced models for intercrystalline crack initiation and propagation e.g. based on fracture mechanics considerations. Auxiliary routines are provided for crack interaction handling and evaluation of statistical properties of crack patterns and tessellations. Recent application fields are mentioned briefly.

VorTess: Erzeugung zweidimensionaler Poisson-Voronoi Mosaik als Grundlage für die mikromechanische Modellierung polykristalliner Werkstoffe — Programmbeschreibung und Algorithmus —

Zusammenfassung

Contents

1 Introduction 1

2 Scheme of the algorithm 3
  2.1 General .................................................. 3
  2.2 Nomenclature ............................................. 3
  2.3 Algorithm ................................................ 4
  2.4 Data storage and retrieval .............................. 5
    2.4.1 Tessellation files format .......................... 6
    2.4.2 Crack pattern files format ........................ 7

3 Programming considerations 9
  3.1 General .................................................. 9
  3.2 Dirichlet tesselation library ........................... 9
  3.3 Damage simulation library .............................. 9
  3.4 Crack extraction and facet characterization library ... 10
  3.5 Data flow subroutines .................................. 12
  3.6 Graphics .................................................. 12

4 Applications 13

References 15

A Dirichlet tesselation library 17
  A.1 Subroutine TESSEL ..................................... 17
  A.2 Subroutine RAHMEN ...................................... 18
  A.3 Subroutine PUNKTE ..................................... 19
  A.4 Subroutine ANFANG ..................................... 19
  A.5 Subroutine UPDATE ..................................... 20
  A.6 Subroutine NACHB (IPUNKT,NNACHB) ................... 20
  A.7 Subroutine NACHF (NEXTP,NEUP,NEXPOS) ............... 21
A.8 Subroutine NEXTT (IPUNKT, NEXTP, NEUP, EX, EY, IUHRZ) .......... 22
A.9 Subroutine CONLIA (NEUP, IPUNKT, NEXTP, ECKX, ECKY) .......... 23
A.10 Subroutine CONLIN (IPUNKT, NEUP, ECKX, ECKY) .......... 24
A.11 Subroutine COLIRA (NEUP, IPUNKT, NEXTP) .......... 24
A.12 Subroutine GNEU (IPUNKT, NEXTP, NEUP, EXPOS, ECKX, ECKY) .......... 25
A.13 Subroutine DELCON (NEXTP, IPUNKT, NEUP, ECKX, ECKY) .......... 26
A.14 Subroutine DECORA (NEXTP, IPUNKT, NEUP) .......... 26
A.15 Integer function MODP .......... 27
A.16 Integer function MODR .......... 28
A.17 Integer function ISTART .......... 28
A.18 Integer function IENDE .......... 29
A.19 Integer function IFINDE .......... 29
A.20 Subroutine DRUCK (IP) .......... 30
A.21 Subroutine DRURA (IP) .......... 30
A.22 Subroutine DKOMP (IP) .......... 31
A.23 Subroutine VOLL (IP) .......... 31
A.24 Double precision function SGN (IARG) .......... 32
A.25 Double precision function DIST (IP1, IP2) .......... 32
A.26 Subroutine EINGAB .......... 33
A.27 Subroutine AUSGAB .......... 33

B Damage simulation library .... 35
B.1 Subroutine CAVIT ................. 35
B.2 Subroutine CAVNEU ................. 36
B.3 Subroutine PFAD (IRAND, EXALT, EYALT, UX, UY) ................. 37
B.4 Subroutine DAMAGE (I, NACHB, IWW, KAPUFT) ................. 38
B.5 Integer function IWCAV (NEXTP, NEUP) ................. 38
B.6 Integer function IWICHT (NEXTP, NEUP) ................. 39
B.7 Subroutine MODELL (KONF, IWW, IW, VSTERK) ................. 40

C Crack extraction and facet characterization library .... 41
C.1 Subroutine CRACKI (NCRACK) ................. 41
C.2 Subroutine CRACK (NEXTP, NEUP, NCRACK) ................. 42
C.3 Subroutine DIRECT (NXP, NUP, NRP, NLP, ILNRP, KMO) ................. 43
C.4 Subroutine CHARAK (NCRACK) ................. 43
C.5 Subroutine BNRR0 (KR, KL, KM, ICRAK) ................. 45
C.6 Subroutine BNRR1 (KD, KM, ICRAK) ................. 45
C.7 Subroutine SETCAV (K1, K2) ................. 46
C.8 Subroutine DRUNRR ................. 46
C.9 Integer function KM0 (KR, KL) ................. 47
D  Data flow subroutines

D.1  Subroutine TLOAD .......................... 50
D.2  Subroutine TSTORE .......................... 51
D.3  Subroutine RLOAD (NCRACK) ................. 51
D.4  Subroutine RSTORE (NCRACK, IUNIT) ............ 52

E  Auxiliary subroutines

E.1  Subroutine ERSDZ (NCRACK, ICNR, PXR, PYR) ................ 53
E.2  Subroutine AREA .................................. 54
Introduction

The following report contains a description of the code VorTess. The development of the code extended over several years and was possible by financial support of the Deutsche Forschungsgemeinschaft (DFG) under grants No. Mu-466/15 (creep lifetime prediction), Mu-466/20 and Mi-362/5 (thermal fatigue lifetime).

The purpose of this code is to generate a random cell structure, the so-called Dirichlet tesselation or Poisson-Voronoi mosaic, which can be used to simulate grain structures as obtained by planar modelling of polycrystalline materials.

The obtained Dirichlet tesselation is used to deal with certain mechanisms of damage in these polycrystalline materials. Basically, all kinds of damage which affect the grain boundaries can be handled. Damaged grain boundaries are marked and can be extracted from the grain structure given by the Dirichlet tesselation to allow separate treatment, e.g. as cracks in the material.

Apart from the attractive modelling capabilities of this approach, the algorithm is also able to handle configurations with comparatively large numbers of grains in a very efficient way by keeping track of relations between grains located next to each other.

Up to now, most of the simulations that deal with multiple crack interactions have led to a prohibitively large computational effort caused by the steep increase of potential interaction partners for a given crack with increasing number of cracks. This effect is avoided by using the Dirichlet tesselation, because for every crack it is possible to reduce the potential interaction partners to those located in the immediate neighbourhood.

The main part of the following report shall give an overview of the scheme of the algorithm used for the construction of the Dirichlet tesselation.

Then, the program structure and the meaning of the variables is given.

Possible application fields that developed during the past few years are indicated mainly for reference purposes.

The description of the different subroutines is given in the Appendix which is divided into different parts.

Appendix A describes subroutines related to the creation of a Dirichlet tesselation.

Appendix B describes subroutines related to the simulation of damage of the facets of the Dirichlet tesselation.

Appendix C describes subroutines related to the separate treatment of damaged facets as cracks and their relation to the underlying mosaic.
Finally, in Appendix D some auxiliary routines are given which can be used to determine some useful quantities characterizing the mosaic or the crack patterns due to damage simulation. This includes a subroutine for the efficient handling of crack interaction effects in the fracture mechanics description of neighboring cracks.

The subroutine libraries are organized in such a way that creating a Dirichlet tessellation and simulating damage of facets are independent tasks. Therefore, great flexibility is obtained and it is no problem to incorporate different damage simulation models, as long as the grain boundary facets are the only elements suffering from damage. It is only necessary to provide suitable subroutines for the damage simulation library.

The present algorithm for the underlying point process which generates the Dirichlet tessellation give uniformly distributed points within a rectangular window. This corresponds to a POISSON point process. Other point processes in polyhedral-shaped windows may be easily incorporated by changes of the subroutine FUNKTE (e.g. to allow for hardcore or cluster processes) or the input data for the window coordinates, respectively.

**Note:** If the shape of the window is changed from rectangular to polyhedral, it will be necessary to adapt the point-generating subroutine FUNKTE accordingly in order to avoid points generated outside of the window. A suitable algorithm is given e.g. in Ref. [1].

**Acknowledgement**

The code VorTess was developed during several research projects sponsored by the Deutsche Forschungsgemeinschaft (DFG) under grants No. Ma-466/15, Ma-466/20 and Mi-362/5. Financial support of the DFG is gratefully acknowledged.
Scheme of the algorithm

2.1 General

The Dirichlet or Voronoi tesselation of the plane represents a special case of a partition of the plane into convex open polygons, called the tessels (or cells, or grains) of the mosaic. In mathematical terms, a tesselation means that the polygons are pairwise disjoint and the union of their closures fills the plane. In our case, the Dirichlet tesselation is generated using central points of the polygons, which are denoted simply as the points of the tesselation. Only a finite part of the plane is considered, bounded by a finite number of edges whose vertices are given. This model is also referred to as a germ-grain model because each point can be seen as a germ for a certain polygon, the grain.

2.2 Nomenclature

For brevity, the following terms are used to characterize the elements of a tesselation.

tessel element of a tesselation

grain synonym for tessel (used in view of its possible physical interpretation)

window the finite part of the plane, in which the Dirichlet tesselation is to be constructed

germs the convex polygons which are located in a certain surrounding of the germs

neighbours neighbours of a grain are all grains sharing a common edge with it

contiguity list list of all neighbours of a specific grain

vertex list list of the coordinates of all vertices of a specific grain

In the following, a grain is often referred to by its germ, and the term ’point’ is used as a general term for germ, grain or tessel, respectively. ’Contiguity list of a point’ and ’contiguity list of a grain’ therefore have an identical meaning.
2.3 Algorithm

An algorithm based on ideas given by Green and Sibson [2] was used to construct a Dirichlet tesselation of a set of given points at random locations within a prescribed window. This algorithm is based on the fact that it is possible to order the neighbours of a given grain in a clockwise or anticlockwise manner. This is a specific feature of this kind of planar mosaics and essential to establish relations between adjacent grains.

The algorithm allows to generate the Dirichlet tesselation point by point. Starting from an initial stage, where the first point constitutes the first grain which is identical with the whole window, each further step means that one additional point is added to the tesselation and the tesselation is updated. This means that all tessels that are affected by the new point have to be modified. Thus, it is necessary

- to generate the contiguity list of the additional point including its vertex list and
- to modify the contiguity list of tessel which are affected by the introduction of the new point.

A detailed view on the used algorithm shows that an additional point IPUNKT is introduced in an existing tesselation performing the following steps:

1. search for the point NNACHB in the existing tesselation in whose tessel the new point IPUNKT is located
2. determine the midpoint of the line connecting IPUNKT with its nearest neighbour NNACHB; this is the starting point from which the search for the succeeding neighbours of the new point begins
3. search for the intersection point (ECKX, ECKY) of a straight halfl ine originating from the starting point with one of the edges of NNACHB, where the orientation of the halfl ine is anticlockwise with respect to IPUNKT
4. determine the neighbour NEUP of NNACHB lying adjacent to the edge containing the intersection point (ECKX, ECKY)
5. insert entry IPUNKT into the contiguity list and entries (ECKX, ECKY) into the vertex list of NEUP
6. set NEXTP = NEUP, set starting point for search equal to (ECKX, ECKY)
7. look for the intersection point (ECKX, ECKY) of a straight halfl ine originating from the starting point with one of the edges of NEXTP, where the orientation of the halfl ine is anticlockwise with respect to IPUNKT
8. insert entry NEXTP into the contiguity list and entries (ECKX, ECKY) in the vertex list of IPUNKT
9. delete superfluous entries in the contiguity list and the vertex list of NEXTP; insert vertex (ECKX, ECKY) in the vertex list of NEXTP
10. if NEXTP = NNACHB, the tessel is complete; otherwise continue with step 5

If NEUP is an edge of the window (i.e. it has negative sign), the steps 5 to 8 have to be replaced by the following steps:
1. determine succeeding neighbour MEXPOS of MEXTP on the edge NEUP
2. insert entry IPUNKT into the contiguity list of NEUP
3. set MEXTP = NEUP
4. determine the succeeding point NEUP on the edge MEXTP and the corresponding vertex (ECKX, ECKY)
5. insert entry MEXTP into the contiguity list and entries (ECKX, ECKY) in the vertex list of IPUNKT

Figure 2.1 shows an intermediate stage of generating a new tessell. It can be seen how the inserted tessell 'cuts out' parts of the adjacent tessells. From Figure 2.1 it also becomes clear that the computing expenditure required to generate an additional tessell is largely independent of the number of tessells already present in the tessellation. Only step 1 will require additional effort with increasing number of points, but due to the search algorithm applied and the randomness of the generated point locations, computational effort will increase not more than proportionally to the square of the number of points, which is reasonably slow.

Upon completion of the algorithm, a contiguity list and a vertex list is available for each tessell.

The described algorithm mainly relies on the fact that in 2-D it is possible to establish unique and ordered neighbour lists (e.g. by clockwise recording of neighbours). A generalization to the 3-D case is therefore not straightforward, however, there are other algorithms available in the literature based on vertex recording [3].

### 2.4 Data storage and retrieval

A common data storage and retrieval format is used for tesselation data and for the subsequently generated crack patterns.
2.4.1 Tessellation files format

The following example shows the lists for point number one and its neighbours in a given tessellation with 99 tessels (see Figure 2.2).

5 ELEMENTE IN ZELLE # 1

<table>
<thead>
<tr>
<th></th>
<th>60</th>
<th>-1</th>
<th>5</th>
<th>41</th>
<th>27</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2.0990241</td>
<td>2.0219145</td>
<td>3.6072218</td>
<td>2.9802236</td>
<td>2.3758535</td>
<td></td>
</tr>
<tr>
<td>9.6622007</td>
<td>2.6071079</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5 ELEMENTE IN ZELLE # 5

<table>
<thead>
<tr>
<th></th>
<th>41</th>
<th>-1</th>
<th>65</th>
<th>28</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8.1604141</td>
<td>8.9214689</td>
<td>10.0000000</td>
<td>10.0000000</td>
<td>8.2071629</td>
<td>7.9018173</td>
</tr>
<tr>
<td>3.2684964</td>
<td>2.9802236</td>
<td>3.6072218</td>
<td>4.6634964</td>
<td>4.110933</td>
<td>3.7389704</td>
</tr>
<tr>
<td>8.9637537</td>
<td>3.8085417</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4 ELEMENTE IN ZELLE # 27

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>41</th>
<th>73</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2.0990241</td>
<td>2.3758535</td>
<td>1.6146726</td>
<td>1.3804673</td>
<td></td>
</tr>
<tr>
<td>9.1825869</td>
<td>1.7378334</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For each tessell, there are 6 output lines.

The first line contains the number of entries in the list together with the number of the tessell.

In the second line, the number of each neighbour is shown (i.e. the contiguity list). Negative values indicate edges of the window.

The third line is for future use and will contain the marks for damaged facets (see below).

Lines four and five contain the vertex coordinates ECKX and ECKY of the first vertex of the facet (clockwise).

The last line contains the coordinates of the generating point of the tessell.

### 2.4.2 Crack pattern files format

There are two different options for crack pattern recording. First, crack patterns can be simply retrieved from tessellation files using the information in line three (see above) of each grain, which indicates whether the facet adjacent to the neighbouring grain given in line two of the same column is damaged (i.e. cracked) or not. In that case, there is no information available about the shape and neighbourhood of cracks. Therefore, a second option is provided where information on all facets of an isolated crack is combined. This information is collected in a crack pattern file where separate cracks are recorded according to the following scheme:

```
1 4 RISS-NR. MIT ANZAHL DER KANTEN
  0   1   2   0   0
  22  22  164  164  1178
  342 342  342  1178  342
  152 164  1178  950  771
  5.0973754  5.2855169  5.0374576  4.9151133  4.9325897
```
Each crack occupies 7 lines of information. In the first line, the number of the crack is given together with the number of facets it contains. The second line contains flags that indicate the shape of the crack and are important for the plotting subroutine as well as for the potential fracture mechanics description (0 - end point; 1 - kink point; 2 - branching point; 3 - closed loop point). Lines 3-5 contain the connection to the Dirichlet tesselation, namely, the numbers of the grains on the left and right side of the facet (looking from the starting point of the facet) as well as that ahead adjacent to the facet end point, and lines 6 and 7 contain the x- and y- coordinates of the starting point of the current crack facet.

There is no header in the crack pattern file because all information about the window and the contiguity lists of the frame is already contained in the corresponding tesselation file.

Generating crack pattern files and preserving grain boundary facet cracking information is the main difference to other codes dealing with different aspects of random mosaics that are available in the literature or on the internet. This was the main reason for developing an own code instead of simply adopting existing programs.
Programming considerations

3.1 General

The programming language used is FORTRAN 77. Variants of the program are running on IBM MVS 3090, under UNIX and also under LINUX. Variables are mainly communicated between different subroutines via COMMON blocks. Maximum array bounds are given in PARAMETER statements (see Table 3.1), which allows a flexible memory adjustment for test runs with a usually small number of tessels and production runs which may contain a very large number of tessels within one tessellation. The whole program is organized within separate libraries described below.

3.2 Dirichlet tessellation library

Most of the variables and arrays are transferred to the subroutines via the different COMMON-blocks which are given in Table 3.2, together with the bounds of the arrays. The COMMON blocks of Table 3.2 are compiled in a separate file which is included in the respective subroutines via the FORTRAN statement INCLUDE (COMTESS), where COMTESS is the name of the file.

The maximum bounds of the arrays can be adjusted by changing the PARAMETER statements which are given in Table 3.1; the actual bounds (i.e. the part of the array that is really used) depend on the number of points of the tessellation and have to be given in the input data set.

3.3 Damage simulation library

The variables described in Table 3.3 are related to a phenomenological way of introducing damage into the tessellation. The COMMON blocks of Table 3.3 are compiled in a separate file which is included in the respective subroutines via the FORTRAN statement INCLUDE (COMKAV), where

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>Maximum number of points for tessellation</td>
</tr>
<tr>
<td>NR</td>
<td>Maximum number of edges of the window</td>
</tr>
<tr>
<td>NMOD</td>
<td>Maximum number of entries (neighbours) in contiguity list of a point</td>
</tr>
<tr>
<td>NMODR</td>
<td>Maximum number of entries (neighbours) in contiguity list of an edge</td>
</tr>
</tbody>
</table>

Table 3.1: Variables defined in PARAMETER statements

Variables are used to adjust array dimensions in COMMON blocks in order to save memory.
### Table 3.2: Variables in COMMON blocks related to the Dirichlet tessellation

<table>
<thead>
<tr>
<th>COMMON block</th>
<th>Variable (bounds)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FACETT</td>
<td>ICAVIT (1:NP,-1:NM0D+1)</td>
<td>Cavitation list array for points</td>
</tr>
<tr>
<td>ZUFALL</td>
<td>SCHAED</td>
<td>Damage level of initial configuration</td>
</tr>
<tr>
<td></td>
<td>SCHINK</td>
<td>Damage level of succeeding configurations</td>
</tr>
<tr>
<td></td>
<td>ISEEDT</td>
<td>Random number generator seed for tessellation</td>
</tr>
<tr>
<td></td>
<td>ISEEDS</td>
<td>Random number generator seed for damage</td>
</tr>
<tr>
<td>PATH</td>
<td>IAC</td>
<td>Counter for number of cavitated facets</td>
</tr>
<tr>
<td></td>
<td>NAC</td>
<td>Counter for total number of facets</td>
</tr>
<tr>
<td></td>
<td>IC</td>
<td>Counter for weighted number of cavitated facets</td>
</tr>
<tr>
<td>ICAVI</td>
<td>NCAV</td>
<td>Counter for number of cavitated facets</td>
</tr>
<tr>
<td></td>
<td>NGES</td>
<td>Counter for total number of facets</td>
</tr>
<tr>
<td></td>
<td>DCAV</td>
<td>Fraction of cavitated facets contained in the tessellation</td>
</tr>
</tbody>
</table>

### Table 3.3: Variables in COMMON blocks related to damage simulation

<table>
<thead>
<tr>
<th>COMMON block</th>
<th>Variable (bounds)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMKAV</td>
<td></td>
<td>is the name of the file. Fracture mechanics variables which are necessary for a physically based damage simulation are not included in this report.</td>
</tr>
</tbody>
</table>

### 3.4 Crack extraction and facet characterization library

Damage is introduced into a tessellation facet by facet. Therefore, cracks (i.e. connected cavitated facets) have to be extracted from the tessellation in a convenient way. The extraction is done by the subroutine CRACK and consists in the determination of the nodes of the cracks and of their adjacent grains as well as in the characterization of the nodes (end nodes, middle nodes or branching nodes, respectively).

Subsequently, facet characterization is performed by the subroutine CHARAKT. The dimensions of the arrays can be adjusted by changing the PARAMETER statements which are given in Table 3.4; the related data blocks are given in Table 3.5. The COMMON blocks of Table 3.5 are compiled in a separate file which is included in the respective subroutines via the FORTRAN statement INCLUDE (COMRISS), where COMRISS is the name of the file.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NKMAX</td>
<td>Maximum number of nodes within one crack</td>
</tr>
<tr>
<td>NCMA</td>
<td>Maximum number of cracks within one tessellation</td>
</tr>
</tbody>
</table>

Table 3.4: Variables defined in PARAMETER statements
Variables are used to adjust array dimensions in COMMON blocks in order to save memory.

<table>
<thead>
<tr>
<th>COMMON block</th>
<th>Variable (bounds)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RISSE</td>
<td>IRCAV (1:NP,-1:NMOD+1)</td>
<td>Characterization list array for facets of Dirichlet tessellation</td>
</tr>
<tr>
<td>KNOTEN</td>
<td>NNODE (1:NCMAX,0:NKMAX)</td>
<td>Node characterization flags for each crack</td>
</tr>
<tr>
<td></td>
<td>NKR (1:NCMAX,0:NKMAX)</td>
<td>Right grain of a crack facet</td>
</tr>
<tr>
<td></td>
<td>NKL (1:NCMAX,0:NKMAX)</td>
<td>Left grain of a crack facet</td>
</tr>
<tr>
<td></td>
<td>NKM (1:NCMAX,0:NKMAX)</td>
<td>Grain between NKL and NKM</td>
</tr>
<tr>
<td></td>
<td>INODE (1:NCMAX)</td>
<td>Number of facets of one crack</td>
</tr>
<tr>
<td>KOORD</td>
<td>DDNODX (1:NCMAX,0:NKMAX)</td>
<td>x-coordinates of crack nodes</td>
</tr>
<tr>
<td></td>
<td>DDNODY (1:NCMAX,0:NKMAX)</td>
<td>y-coordinates of crack nodes</td>
</tr>
<tr>
<td>BEWERT</td>
<td>NRR1 (1:NP, 1:NMOD-1)</td>
<td>Auxiliary array for facet characterization</td>
</tr>
<tr>
<td></td>
<td>NRR2 (1:NP, 1:NMOD-1)</td>
<td>Auxiliary array for facet characterization</td>
</tr>
</tbody>
</table>

Table 3.5: Variables related to crack extraction and facet characterization.
3.5 Data flow subroutines

Construction of the Dirichlet tesselation without damage and introducing damage can be performed separately. This allows the use of 'model' tesselations with different amounts of damage. Therefore, data flow subroutines are supported for storage (subroutine TSTORE) and retrieval (subroutine TLOAD) of a given tesselation with or without damage.

Additionally, the resulting crack patterns which are generated by the crack extraction routines can be stored in files (subroutine RSTORE). Also a retrieval subroutine (RLOAD) exists for the crack patterns to be loaded e.g. for graphical presentation. Details are given in the corresponding paragraph.

3.6 Graphics

Graphics interfaces for both GKS (on MVS systems) and gnuplot (on UNIX/LINUX systems) are available, but not described in this report. For GKS, a GKS metafile is generated for further processing. For gnuplot, a set of two files is generated for each plot, the first containing plotting format specifications and the second containing the data.

Presentation of complete tesselations as well as crack patterns is possible. Crack patterns can be plotted from tesselation files using the information in array ICAVIT as well as from crack extraction files. In both cases, labeling is supported for better identification of tessel and cracks especially for demonstration purposes.

The graphics interfaces use the routines TLOAD and RLOAD, respectively, for data retrieval.
Applications

During the last years, a variety of possible application fields opened up because of the increasing interest in stochastic modelling of polycrystalline solids on a mesoscopic scale. A phenomenological damage model was applied for intergranular creep cavitation [4]. A fracture mechanics model for cracking under thermal shock loading was developed, leading to a largely sophisticated version of the subroutine DAMAGE for crack facet failure [5]. Failure due to creep-assisted intergranular stress corrosion cracking was also modelled [6]. The tessellation algorithm was used to model the spatial distribution of fibres in reinforced ceramic materials [7] together with their respective fibre volume fraction and to obtain an interpretation of results for the slice compression test experiments for this class of materials.

Current applications focus on modelling of fatigue crack growth for martensitic steels [8], indentation loading for ceramics [9] and domain characterization together with constitutive behaviour modelling for piezoelectric material [10].

Only recently, a number of papers were published by different authors showing the advantageous use of the tessellation approach in materials science. These papers covered a wide range of applications, such as the Voronoi cell-based finite element method for elastic analysis of heterogeneous structures [13], micro-shear banding in crystal plasticity [12], and creep and grain boundary sliding of polycrystals [14].
References


Appendix A

Dirichlet tessellation library

The Dirichlet tessellation library contains all subroutines which are necessary to obtain a Dirichlet tessellation in a convex window with a given number of edges and their respective vertices. The number of points in the window as well as the coordinates of the vertices of the window and some starting value for the random number generator have to be supplied by the input data set. The main program has to organize data input; subroutine TESSEL is then called to complete the construction of the Dirichlet tessellation, control is then returned to the calling main program. Data output or damage simulation may follow, if convenient.

The subroutines of the Dirichlet tessellation library shall now be described in detail.

A.1 Subroutine TESSEL

Description
This is the main program, organized as a subroutine. Its purpose is to construct the Dirichlet tessellation of a window containing a certain number points at prescribed random locations. The tessellation is performed iteratively. The tessellation containing only the first point comprises the complete window. Subsequently, the tessellation is updated pointwise until all points are recorded and their respective contiguity lists are completed.

Parameters In:
None

Parameters Out:
None

External Subroutines:

RAHMEN define window
PUNKTE generate randomly distributed points in the window
ANFANG construct the contiguity list of the first point in the window
UPDATE construct the contiguity list of one subsequent point
DRUCK generate printout of the contiguity list of one point
DRURA generate printout of the contiguity list of one edge
VOLL  determine the maximum number of neighbours in the contiguity list of all points of a
tessellation

**External Functions:**

None

**Local Variables:**

<table>
<thead>
<tr>
<th>IPUNKT</th>
<th>auxiliary variable (usually IPUNKT=1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP</td>
<td>counter for tessel which is presently being constructed</td>
</tr>
<tr>
<td>I</td>
<td>loop counter</td>
</tr>
</tbody>
</table>

**A.2 Subroutine RAHMEN**

**Description**

Generate contiguity list of all edges of the window. Edges are treated in a similar way as points, but with a negative sign and modified contiguity lists because of the larger number of possible neighbours.

**Parameters In:**

None

**Parameters Out:**

None

**External Subroutines:**

None

**External Functions:**

None

**Local Variables:**

<table>
<thead>
<tr>
<th>IP</th>
<th>loop counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPM, IPP</td>
<td>auxiliary variable</td>
</tr>
</tbody>
</table>
A.3 Subroutine PUNKTE

Description
Generate sample of random points within a predefined window.

Parameters In:
None

Parameters Out:
None

External Subroutines:
None

External Functions:
DRUNIF uniform random number generator (IMSL library)

Local Variables:
IZ loop counter
XSI, YSI auxiliary variables

A.4 Subroutine ANFANG

Description
Create the contiguity list of the first point in the window.

Parameters In:
None

Parameters Out:
None

External Subroutines:
None

External Functions:
MODP calculate modulus of an integer with respect to IMOD

Local Variables:
I, IP loop counters
I1, IP1 auxiliary variables
A.5 Subroutine UPDATE

Description
Update a given tesselation by adding a new point. Construct the contiguity list of the new point. Update the contiguity lists of the adjacent points.

Parameters In:

IPUNKT point for which contiguity list is currently being constructed

Parameters Out:
None

External Subroutines:

NACHB determine nearest neighbour of IPUNKT
NEXTT determine next neighbour to be inserted into the contiguity list of IPUNKT
CONLIA insert point IPUNKT into the contiguity list of the next neighbour, determined by NEXTT
CONLIN insert the next neighbour, determined in NEXTT, into the contiguity list of IPUNKT
NACHF determine the next neighbour to be inserted into the contiguity list of IPUNKT, if the tessel of IPUNKT lies adjacent to an edge of the window
COLIRA insert number IPUNKT into the contiguity list of an edge of the window
GNEU determine next neighbour to be inserted into the contiguity list of IPUNKT, if the tessel of IPUNKT lies adjacent to an edge of the window
DELCON delete members of the contiguity list of neighbours replaced by IPUNKT

External Functions:
None

Local Variables:

NNACHB nearest neighbour of IPUNKT
IUHRZ flag to determine search direction for subroutine NEXTT
NEXTP, NEUP, NEXPOS temporary variables for construction of contiguity list
ECKX, ECKY coordinates of next vertex to be inserted into contiguity list

A.6 Subroutine NACHB (IPUNKT, NNACHB)

Description
Find tessel in which the point IPUNKT which is to be added to the tesselation is situated. Starting from an initial point (which is arbitrarily set to 1) the search is performed along the contiguity list of successive points, jumping to the next point whenever its distance to IPUNKT becomes less than the distance of the present point to IPUNKT, until there is no such point in the complete contiguity list. This point is then the nearest neighbour of IPUNKT, i.e. IPUNKT is situated within its tessel.

Parameters In:
IPUNKT point for which contiguity list is currently being constructed

**Parameters Out:**

NNACHB nearest neighbour of IPUNKT

**External Subroutines:**

None

**External Functions:**

DIST calculate distance between two grain centres
MODP calculate modulus of an integer with respect to IMOD

**Local Variables:**

ISTART starting point for search (set to ISTART=1)
I, IZ loop counters
ABST0, ABST1, ICON auxiliary variables

### A.7 Subroutine NACHF (NEXTP, NEUP, NEXPOS)

**Description**

Find successor NEXPOS of NEXTP in the contiguity list of the edge NEXTP in a clockwise search direction. (NACHF and GNEU are called instead of NEXTT, if edge effects have to be considered)

**Parameters In:**

NEXTP point whose successor is searched
NEUP edge of the window whose contiguity list has to be checked for NEXPOS

**Parameters Out:**

NEXPOS successor of NEXTP in the contiguity list of the edge NEUP in clockwise direction

**External Subroutines:**

None

**External Functions:**

ISTART find first entry of contiguity list
IENDE find last entry of contiguity list
IFINDE find location of a certain point in the contiguity list of another point
MODR calculate modulus of an integer with respect to IMODR

**Local Variables:**

I, I1 temporary variables for loop counters
IBEG, IEND temporary variables for first and last entry of contiguity list
IFIND, IFIRA, INEXT temporary variables for entry points in contiguity lists
A.8 Subroutine NEXTT (IPUNKT,NEXTP,NEUP,EX,EY,IUHRZ)

Description

Find next point NEUP to be inserted into the contiguity list of the present point IPUNKT. Also the coordinates of the vertex of the edge between IPUNKT and NEUP are determined. The search for NEUP is performed by looking for the intersection point of a straight line starting from the midpoint of the line connecting IPUNKT and NEXTP in a prescribed search direction (i.e. clockwise with respect to IPUNKT) with the edges of the tessel containing the point NEXTP. The neighbour of NEXTP whose common edge with NEXTP contains the intersection point of the search line is the desired point NEUP.

Note: NEXTT is also called as auxiliary routine by the subroutine GNEU.

Parameters In:

IPUNKT point for which contiguity list is currently being constructed
NEXTP number of the last already found member of the contiguity list of IPUNKT
IUHRZ flag to determine search direction
  1 normal search in clockwise direction
  -1 search direction anticlockwise
  -2 search direction anticlockwise; perform search until NEUP is negative

Parameters Out:

NEUP next neighbour in the contiguity list of IPUNKT
EX, EY coordinates of vertex at the beginning of the edge between IPUNKT and NEUP to be added in contiguity list of IPUNKT
IUHRZ flag for GNEU to indicate failure of search for intersection point
  -5 no intersection point found

External Subroutines:

DKOMP auxiliary printing routine for debugging purposes

External Functions:

SGN determine sign of an integer
ISTART find first entry of contiguity list
IENDE find last entry of contiguity list
IFINDE find point in contiguity list of another point
MODP calculate modulus of an integer with respect to IMOD

Local Variables:

IPRINT control variable for printout
X, Y, U1, U2, VORZ temporary variables to determine unit vector of search direction
PXI, PYI, PXN, PMY temporary variables for point coordinates
A.9 Subroutine CONLIA (NEUP, IPUNKT, NEXTP, ECKX, ECKY)

Description
Insert the new point IPUNKT into the contiguity list of the neighbour NEUP. Insert the vertex coordinates of the edge between IPUNKT and NEUP into the vertex list of NEUP.

Parameters In:

IPUNKT point for which contiguity list is currently being constructed
NEXTP number of the last already found member of the contiguity list of IPUNKT
NEUP point whose contiguity list is updated
ECKX, ECKY coordinates of the vertex of the edge between NEUP and IPUNKT

Parameters Out:
None

External Subroutines:
None

External Functions:

ISTART find first entry of contiguity list
IENDE find last entry of contiguity list
MODP calculate modulus of an integer with respect to IMOD

Local Variables:
I, J, J1 temporary variables for loop counters
IBEG, IEND temporary variables for first and last entry of contiguity list
ICON temporary variable for entry in contiguity list
IHILF, XHILF, YHILF, IHILF1, XHILF1, YHILF1 auxiliary variables
A.10 Subroutine CONLIN (IPUNKT, NEUP, ECKX, ECKY)

Description
Insert a new point NEUP into contiguity list of IPUNKT. Insert vertex coordinates of edge between IPUNKT and NEUP into the vertex list of IPUNKT.

Parameters In:

- IPUNKT: point for which contiguity list is currently being constructed
- NEUP: point whose contiguity list is updated
- ECKX, ECKY: coordinates of the vertex of the edge between IPUNKT and NEUP

Parameters Out:

- None

External Subroutines:

- None

External Functions:

- MODP: calculate modulus of an integer with respect to IMOD

Local Variables:

- I: loop counter
- ICON: temporary variable for entry in contiguity list

A.11 Subroutine COLIRA (NEUP, IPUNKT, NETPX)

Description
Version of subroutine CONLIA if NEUP is an edge of the window.

Parameters In:

- IPUNKT: point for which contiguity list is currently being constructed
- NEUX: number of the last already found member of the contiguity list of IPUNKT
- NEUP: edge whose contiguity list is updated

Parameters Out:

- None

External Subroutines:

- DKOMP: auxiliary printout routine for debugging purposes
External Functions:

ISTART find first entry of contiguity list
IENDE find last entry of contiguity list
MODR calculate modulus of an integer with respect to IMODR

Local Variables:

I, J, J1 temporary variables for loop counters
IBEG, IEND temporary variables for first and last entry of contiguity list
ICON, IHILF, IHILF1 temporary variables for entries in contiguity list

A.12 Subroutine GNEU (IPUNKT, NEXTP, NEUP, NEXPOS, ECKX, ECKY)

Description

GNEU is called by subroutine UPDATE instead of subroutine NEXTT, if NEXTP is not a point but an edge of the window.

Parameters In:

IPUNKT point for which contiguity list is currently being constructed
NEXTP number of the last already found member of the contiguity list of IPUNKT (NEXTP is an edge of the window)
NEXPOS next entry (clockwise) in the contiguity list of the edge NEXTP (this may be a point (if NEXPOS positive) or an edge of the window (if NEXPOS negative))

Parameters Out:

NEUP next neighbour in the contiguity list of IPUNKT
ECKX, ECKY coordinates of the next vertex for contiguity list

External Subroutines:

NEXTT determine next point and corresponding vertex coordinates on edge of the window
NACHF update of NEXPOS

External Functions:

None

Local Variables:

IUHRZ flag to determine search direction
ECKXR, ECKYR temporary variables for vertex coordinates
IRAND temporary variable for number of edge of the window
A.13 Subroutine DELCON (NEXTP, IPUNKT, NEUP, ECKX, ECKY)

Description
Remove those entries from the contiguity list of NEXTP which are replaced by IPUNKT. Also the respective vertices are removed from the vertex list and the new vertex coordinates of the edge between NEXTP and IPUNKT are inserted. All entries between IPUNKT and NEUP are removed.

Parameters In:

- **IPUNKT**: point for which contiguity list is currently being constructed
- **NEXTP**: number of the last already found member of the contiguity list of IPUNKT
- **NEUP**: next neighbour in the contiguity list of IPUNKT, determined by subroutine NEXXT or GNEU

Parameters Out:

- **None**

External Subroutines:

- **DECORA** is called if NEXTP is an edge of the window ('edge version' of DELCON)

External Functions:

- **ISTART**: find first entry of contiguity list
- **IENDE**: find last entry of contiguity list
- **IFINDE**: find entry of a certain point in contiguity list of another point
- **MODP**: calculate modulus of an integer with respect to IMOD

Local Variables:

- **I, I1, J, J1**: temporary variables for loop counters
- **IBEG, IEND**: temporary variables for first and last entry of contiguity list
- **ICON**: temporary variable for entry in contiguity list

A.14 Subroutine DECORA (NEXTP, IPUNKT, NEUP)

Description
Variant of DELCON if NEXTP is an edge of the window.

Parameters In:

- **IPUNKT**: point for which contiguity list is currently being constructed
- **NEXTP**: number of the last already found member of the contiguity list of IPUNKT (NEXTP is an edge of the window)
- **NEUP**: next neighbour in the contiguity list of IPUNKT, determined by subroutine NEXXT or GNEU
Parameters Out:

None

External Subroutines:

None

External Functions:

ISTART    find first entry of contiguity list
IENDE     find last entry of contiguity list
MODR      calculate modulus of an integer with respect to IMODR

Local Variables:

I, I1, J, J1 temporary variables for loop counters
IBEG, IEND temporary variables for first and last entry of contiguity list
ICON      temporary variable for entry in contiguity list

A.15 Integer function MODP

Description
Calculate the modulus of an integer with respect to IMOD. If necessary, the result is shifted by IMOD in order to be positive.

Parameters In:

I      integer variable

Parameters Out:

None

External Subroutines:

None

External Functions:

None

Local Variables:

None
A.16 Integer function MODR

Description
Calculate the modulus of an integer with respect to IMODR. If necessary, the result is shifted by IMODR in order to be positive.

Parameters In:
I integer variable

Parameters Out:
None

External Subroutines:
None

External Functions:
None

Local Variables:
None

A.17 Integer function ISTART

Description
Find first entry in the contiguity list of point NEXTP.

Parameters In:
NEXTP number of point

Parameters Out:
None

External Subroutines:
DKOMP printout routine (called if contiguity list is empty for debugging purposes)

External Functions:
None

Local Variables:
None
A.18 Integer function IENDE

Description
Find last element in the contiguity list of point IP.

Parameters In:
IP number of point

Parameters Out:
None

External Subroutines:
DKOMP printout routine (called if contiguity list is empty for debugging purposes)

External Functions:
None

Local Variables:
None

A.19 Integer function IFINDE

Description
Find entry of point IFIND in contiguity list of point ISUCH.

Parameters In:
ISUCH point whose contiguity list is checked for IFIND
IFIND neighbour whose entry is to be found in the contiguity list of ISUCH

Parameters Out:
None

External Subroutines:
None

External Functions:
ISTART find first entry of contiguity list
IENDE find last entry of contiguity list
MODP calculate modulus of an integer with respect to IMOD
MODR calculate modulus of an integer with respect to IMODR

Local Variables:
I, I1 temporary variables for loop counters
IBEG, IEND temporary variables for first and last entry of contiguity list
ICON temporary variable for entry in contiguity list
A.20 Subroutine DRUCK (IP)

Description
Generate printout of the contiguity list of the point IP

Parameters In:
IP point whose contiguity list is to be printed

Parameters Out:
None

External Subroutines:
DRURA generate printout if IP is an edge of the window

External Functions:
ISTART find first entry of contiguity list

Local Variables:
KSTART, KEND auxiliary variables indicating first and last entry of contiguity list

A.21 Subroutine DRURA (IP)

Description
Generate printout of the contiguity list of the edge IP of the window.

Parameters In:
IP number of the edge to be printed

Parameters Out:
None

External Subroutines:
None

External Functions:
ISTART find first entry of contiguity list

Local Variables:
KSTART, KEND auxiliary variables indicating first and last entry of contiguity list
A.22 Subroutine DKOMP (IP)

Description
Generate printout of the contiguity list of one point in compressed format; auxiliary routine for debugging purposes.

Parameters In:
IP point whose contiguity list is to be printed

Parameters Out:
None

External Subroutines:
None

External Functions:
None

Local Variables:
KSTART, KEND auxiliary variables indicating first and last entry of contiguity list

A.23 Subroutine VOLL (IP)

Description
Determine maximum number of entries in the contiguity list of points 1 to IP.

Parameters In:
IP number of points

Parameters Out:
None

External Subroutines:
None

External Functions:
None

Local Variables:
I loop counter
IMAX auxiliary variable indicating number of entries of contiguity list
IPMAX, IRMAX auxiliary variables indicating point or edge with maximum of entries
A.24 Double precision function SGN (IARG)

Description
Calculate sign of an integer.

Parameters In:
IARG Integer number

Parameters Out:
None

External Subroutines:
None

External Functions:
None

Local Variables:
None

A.25 Double precision function DIST (IP1,IP2)

Description
Calculate the distance between two points in the window.

Parameters In:
IP1, IP2 Points for which distance is to be calculated.

Parameters Out:
None

External Subroutines:
None

External Functions:
None

Local Variables:
DX, DY auxiliary variables
A.26 Subroutine EINGAB

Description

Initialize all arrays and read input quantities for the construction of the Dirichlet tessellation from file on unit 11. The structure of the input file is given in the following example together with comments on the meaning of the quantities:

```
15   400   NMOD, NMODR
10   NPUNKT
4    NRAHM
10.  0.   0.   10.  ECKEX(*)
10.  10.  0.   0.   ECKEX(*)
123456789  ISEEDT
1    NKONF
.0D0  .0D0  ENDE DER KONFIGURATION  <-- this line reserved for future use
```

Free field format is used by subroutine EINGAB. NKONF tessellations can be obtained in a single run by an appropriate modification of the input file (the lines shown above have to be repeated NKONF times). This allows NKONF tessellations to be generated with the option of e.g. different starting values ISEEDT for the IMSL random number generator or different number of points NPUNKT.

Parameters In:

None

Parameters Out:

None

External Subroutines:

None

External Functions:

None

Local Variables:

I, J loop counters

A.27 Subroutine AUSGAB

Description

Print echo of input quantities read by subroutine EINGAB. Perform consistency check of input quantities.

Parameters In:
None

**Parameters Out:**
None

**External Subroutines:**
None

**External Functions:**
None

**Local Variables:**
I   loop counter
INER  (not used)
Appendix B

Damage simulation library

The damage simulation library contains subroutines which are used to complete the array ICAVIT, which indicates whether a facet is damaged (ICA VIT=1) or not (ICA VIT=0). Subroutine CAVIT is used for previously undamaged tessellations, whereas subroutine CA VNEU allows additional cavitated facets to be introduced into previously existing pre-damaged tessellations. Different damage models can be introduced, which may reflect the surroundings of previously cavitated facets in a different manner. This may be done with the help of subroutines DAMAGE and MODELL.

The subroutines described in this section provide a framework for introducing more sophisticated fracture mechanics-based damage models, the description of which is beyond the scope of this report. The fracture mechanics background and damage simulation results can be found in Refs. [5], [6] and [15].

B.1 Subroutine CAVIT

Description
Subroutine controlling the simulation of damage of the facets of the tessellation.

Parameters In:

None

Parameters Out:

None

External Subroutines:

Damage control damage of the facet between two given facets
DRUCK printout of the contiguity list of one point

External Functions:

MODP calculate modulus of an integer with respect to IMOD
IFINDE find entry of a certain point in the contiguity list of another point

Local Variables:
I, J loop counters
NCAV, NGES counters for cavitated and total number of facets, respectively
IWW flag indicating whether interaction of cavitated facets is to be modelled
JSTART, JEND temporary variables for first and last entry of contiguity list
JMOD, NPOS auxiliary variables

B.2 Subroutine CAVNEU

Description
Subroutine controlling the damage evaluation of a tessellation with damaged facets by introducing additional cavitated facets. CAVNEU is essentially identical with CAVIT.

Parameters In:
None

Parameters Out:
None

External Subroutines:

DAMAGE control damage of the facet between two given facets
DRUCK printout of the contiguity list of one point

External Functions:

MODP calculate modulus of an integer with respect to IMOD
IFINDE find entry of a certain point in contiguity list of another point

Local Variables:

ICNEU temporary array for storing new values of ICAVIT. ICAVIT is set to ICNEU upon the end of the subroutine.
I, J loop counters
NCAV, NGES counters for cavitated and total number of facets, respectively
IWW flag indicating whether interaction of cavitated facets is to be modelled
JSTART, JEND temporary variables for first and last entry of contiguity list
JMOD, NPOS auxiliary variables
B.3 Subroutine PFAD (IRAND, EXALT, EYALT, UX, UY)

Description
Determine fraction of cavitated facets hit by a line starting from the point (EXALT, EYALT) at one edge of the window and crossing the window in the direction selected by the direction vector (UX, UY) which can be selected arbitrarily. Subroutine PFAD provides a damage parameter that is used for creep damage by grain boundary cavitation [4].

Parameters In:
IRAND number of the edge of the window containing the starting point
EXALT, EYALT coordinates of the starting point (modified upon completion)
UX, UY line direction vector

Parameters Out:
None

External Subroutines:
DRUCK auxiliary printing routine for debugging purposes

External Functions:
ISTART find first entry of contiguity list
IENDE find last entry of contiguity list
IFINDE find location of a point in contiguity list of another point
MODP calculate modulus of an integer with respect to IMOD
MODR calculate modulus of an integer with respect to IMODR
IWICHT calculate weighting factor for cavitated facets

Local Variables:
IPRINT control variable for printing output
U1, U2 line direction unit vector
DNORM length of line direction vector
IBEG, IEND, IBEGP, IENDP temporary variables for first and last entry of contiguity list
IFIND, IFIND1 temporary variables for entry of a point in contiguity list of another point
IZIEL auxiliary variable for loop range
I, I1, IECK, IECK1 auxiliary variables for loop
D, DL, DM, DLAM, DMUE, DMIN, DLAMX, DLAMY temporary variables
AX, AY, BX, BY, EX, EY, ECKY, ECKX, ECKX1, ECKY1 temporary variables for vertex coordinates
IERR error flag
IAC, IC, NAC counter for cavitated facets
B.4 Subroutine DAMAGE (I, NACHB, IWW, KAPUTT)

Description
Simulate the damage of a single facet of the tessellation.

Parameters In:
I, NACHB neighbouring tessel of the considered facet
IWW flag indicating whether interaction of cavitated facets is to be modelled

Parameters Out:
KAPUTT flag indicating whether the facet is cavitated (1) or not (0)

External Subroutines:
MODELL supply different models for the interaction of damaged facets

External Functions:
IWCAV determine flag for configuration of adjacent cavitated facets
DRNUNF uniform random number generator (IMSL library)

Local Variables:
XCAV, IWC, XSI auxiliary variables

B.5 Integer function IWCAV (NEXTP, NEUP)

Description
Determine flag for adjacent cavitated facets of a non-cavitated facet. IWCAV is set to

0 if no cavitated facet adjacent
1 if one cavitated facet adjacent
2 if two cavitated facets adjacent; one at each side of the facet
-2 if two cavitated facets adjacent at one side of the facet
-3 if three cavitated facets adjacent; i.e. two at one side of the facet and one at the other side of the facet
-4 if four cavitated facets adjacent; two at each side of the facet.

Parameters In:
NEXTP, NEUP neighbouring tessel of the considered facet

Parameters Out:
None

External Subroutines:

None

External Functions:

ISTART find first entry of contiguity list
IENDE find last entry of contiguity list
IFINDE find location of a point in contiguity list of another point
MODP calculate modulus of an integer with respect to IMOD

Local Variables:

NXP, NUP, NXPN, NUPN temporary variables for first and last entry of contiguity list
IBNXP, IBNUP, IENXP, IENUP temporary variables for first and last entry of contiguity list
IXUPN, IUNXP temporary variables for entries in contiguity list
ICXPN, ICUPX temporary variables for cavitated facets in contiguity list
IWC1, IWC2 temporary variables for adjacent cavitated facets at each side
IPP, IPM auxiliary variables

B.6 Integer function IWICHT (NEXTP, NEUP)

Description
Determine number of connected cavitated facets containing the facet between the tessel NEXTP and NEUP.

Parameters In:

NEXTP, NEUP tessel adjacent to the cavitated facet

Parameters Out:

None

External Subroutines:

None

External Functions:

ISTART find first entry of contiguity list
IENDE find last entry of contiguity list
IFINDE find location of a point in contiguity list of another point
MODP calculate modulus of an integer with respect to IMOD
**Local Variables:**

NXP, NUP, NXPN, NUPN temporary variables for first and last entry of contiguity list
IBNXP, IBNUP, IENXP, IENUP temporary variables for first and last entry of contiguity list
IXUPN, IUNXP temporary variables for entries in contiguity list
ICXPN, ICUPX temporary variables for cavitated facets in contiguity list
IPP, IPM auxiliary variables

**B.7 Subroutine MODELL (KONF, IWW, IWC, VSTERK)**

**Description**

Supply different models for the interaction of damaged facets. Calculate enhancement factor VSTERK according to the selected model. VSTERK depends on the configuration of the adjacent damaged facets, which is given by the parameter IWC. Subroutine MODELL provides a very rough way of interaction modelling, but can be taken as an interface for the introduction of fracture mechanics-based models.

**Parameters In:**

KONF control flag for the selection of the appropriate damage model
IWW flag indicating whether interaction of cavitated facets is to be modelled
IWC flag indicating configuration of cavitated facets

**Parameters Out:**

VSTERK enhancement factor for the calculation of the damage probability of a facet in subroutine DAMAGE.

**External Subroutines:**

None

**External Functions:**

None

**Local Variables:**

None
Appendix C

Crack extraction and facet characterization library

C.1 Subroutine CRACKI (NCRACK)

Description
Subroutine controlling the extraction of cracks from a Dirichlet tesselation containing damaged facets. The algorithm is as follows: For each tessel, the facets of the tessel are checked for damage. If so, the number of adjacent damaged facets IWCAV is determined. If IWCAV ≠ 1, the facet is skipped. Otherwise, this facet is a facet located at the end of a crack, whose facets are then recorded by the subroutine CRACK. This is continued until all tessel are checked.

Parameters In:
None

Parameters Out:
NCRACK number of cracks in the tesselation

External Subroutines:
DRUCK printout of the contiguity list of one point
CRACK perform extraction of one isolated crack from the tesselation

External Functions:
IFINDE find location of a point in contiguity list of another point
MODP calculate modulus of an integer with respect to IMOD
IWCAV determine number of adjacent damaged facets

Local Variables:
I, J loop counters
NCFAC counter for cavitated number of facets per tessel
JSTART, JEND temporary variables for first and last entry of contiguity list
JMODO, NPOS auxiliary variables
NACHB, NEXTP, NEUP auxiliary variables
ICR auxiliary variable
C.2 Subroutine CRACK (NEXTP, NEUP, NCRACK)

Description

Subroutine for the extraction of one isolated crack from a tesselation containing cavitated facets. CRACK is called by CRACKI.

The extraction starts from the facet between NEXTP and NEUP, which is a facet at the end of a crack, and is continued node by node until all facets of the crack are reached. At each branching node, the right branch is selected, the node is marked incomplete and recording of the facets continues until an end node is reached. Execution then continues with the left branch of the last incomplete branching node. If an end node is reached and no branching node is left incomplete, the crack is recorded completely.

Note: If closed cracks occur, a warning message is issued. These cracks are, however, still recorded completely. As this occurs only at the very final stage of damage modelling, where the physical basis of the model is breaking down, no effort was made to allow 'closed-loop-cracks' to be recorded.

Parameters In:

NEXTP, NEUP tessel adjacent to the first facet of the crack
NCRACK number of crack extracted from the tesselation

Parameters Out:

None

External Subroutines:

DIRECT determine left and right tessel of the first facet of a crack in the direction of the following facets of the crack

External Functions:

ISTART find first entry of contiguity list
IENDE find last entry of contiguity list
IFINDE find location of a point in contiguity list of another point
MODP calculate modulus of an integer with respect to IMOD

Local Variables:

I, J loop counters
KNODE, LNODEx counters for nodes of a crack
IVZW counter for incomplete branching nodes of a crack
NXP, NUP, NRP, NLP, NRPN, NLPN, KMO temporary variables for adjacent tessel of a crack
IBNLP, IENLP, IBNRP, IENRP, IENLP, IENL, IPNLP temporary variables for the beginning, end and entries of contiguity lists
ICLPN, ICRPN temporary variables for damaged facets
IPP, IPM, ISTOP auxiliary variables
NODE, KL, KR, KM temporary arrays for node marks and adjacent tessel
DNODEX, DNODEY temporary arrays for coordinates of crack nodes
IV counter for branching nodes
KSD flag for recording branching nodes of a crack

42
C.3 Subroutine DIRECT (NXP, NUP, NRP, NLP, ILNRP, KM0)

Description

Determine left and right tessel of the first facet of a crack in direction of the following facets of the crack.

Parameters In:

NXP, NUP tessel adjacent to the first facet of the crack

Parameters Out:

NRP, NLP tessel adjacent to the first facet of the crack ordered in a way that NRP is at the right side and NLP is at the left side of the crack.

ILNRP entry of NRP in contiguity list of NLP

KM0 tessel between NLP and NRP in opposite direction (stored in NKM(0)).

External Subroutines:

None

External Functions:

ISTART find first entry of contiguity list
IENDE find last entry of contiguity list
IFINDE find location of a point in contiguity list of another point
MODP calculate modulus of an integer with respect to IMOD

Local Variables:

IBNXP, IENXP, IBNUP, IENUP, IUNXP, IUNUP temporary variables for beginning, end and entries of contiguity lists
NXPN, NUPN, NUPV temporary variables for adjacent tessel of a crack
ICXPN, ICUPN temporary variables for damaged facets
IPP, IPM auxiliary variables

C.4 Subroutine CHARAK (NCRACK)

Description

Characterize undamaged facets of a tesselation according to the number of damaged facets in its surroundings.

CHARAK determines the array IRCAV which, for every facet of a tessel, is set to one, if the facet is cavitated; i.e. belongs to an existing crack. For facets which are undamaged, IRCAV attains the following values:

0 if undamaged facet adjacent to no crack
2 if undamaged facet adjacent to end node of one crack
3 if undamaged facet adjacent to middle node of one crack
4 if undamaged facet adjacent to 2 end nodes of 2 different cracks
-4 if undamaged facet adjacent to 2 end nodes of 1 single crack
5 if undamaged facet adjacent to 1 end and 1 middle node of 2 different cracks
-5 if undamaged facet adjacent to 1 end and 1 middle node of 1 single crack
6 if undamaged facet adjacent to 2 middle nodes of 2 different cracks
-6 if undamaged facet adjacent to 2 middle nodes of 1 single crack

which means that positive values of \text{IRCAV} denote different cracks (if any), whereas negative values denote different nodes of identical cracks being connected to the undamaged facet.

In a first step, two auxiliary arrays \text{NRR1} and \text{NRR2} are calculated, which contain the cracks adjacent to each facet of each tessel and \text{IRCAV} is set to \text{ICAVIT} for all facets belonging to cracks. The second step, where \text{IRCAV} is determined for all remaining facets of the tesselation (i.e. those not belonging to cracks), completes the characterization of the facets.

\textbf{Parameters In:}

\text{NCRACK} \hspace{1cm} \text{Number of cracks}

\textbf{Parameters Out:}

\text{None}

\textbf{External Subroutines:}

\text{BNRR0} calculate auxiliary arrays \text{NRR1} and \text{NRR2} for end nodes of a crack
\text{BNRR1} calculate auxiliary arrays \text{NRR1} and \text{NRR2} for middle nodes of a crack
\text{SETCAV} set \text{ICAVIT} = 1 for facets belonging to a crack
\text{DRUNRR} auxiliary printing routine for debugging purposes only

\textbf{External Functions:}

\text{KM0} determine \text{KM}(0), if not already available

\textbf{Local Variables:}

\text{I, J, ICRAK, IK} loop counters
\text{JSTART, JEND} temporary variables for first and last entry of contiguity list
\text{IKN} temporary variable for number of nodes
\text{KR, KL, KM} temporary variables for adjacent tessel
C.5 Subroutine BNRRO (KR, KL, KM, ICRACK)

Description
Calculate values of the auxiliary arrays NRR1 and NRR2 for undamaged facets adjacent to end nodes of a crack. If NRR1 is equal to zero, NRR1 is set to ICRACK, otherwise NRR1 is left unchanged and NRR2 is set to ICRACK.

Parameters In:
KR, KL, KM tessels adjacent to the facet of the crack
ICRACK crack presently being recorded

Parameters Out:
None

External Subroutines:
None

External Functions:
IFINDE find location of a point in contiguity list of another point

Local Variables:
IFMR, IFRM, IFML, IFLM auxiliary variables for entries of contiguity lists

C.6 Subroutine BNRRL (KD, KM, ICRACK)

Description
Calculate values of the auxiliary arrays NRR1 and NRR2 for undamaged facets adjacent to middle nodes of a crack. If NRR1 is equal to zero, NRR1 is set to ICRACK, otherwise NRR1 is left unchanged and NRR2 is set to ICRACK.

Parameters In:
KD, KM tessels adjacent to the facet of the crack
ICRACK crack presently being recorded

Parameters Out:
None

External Subroutines:
None

External Functions:
IFINDE find location of a point in contiguity list of another point

Local Variables:
IFMD, IFDM auxiliary variables for entries of contiguity lists
C.7 Subroutine SETCAV (K1,K2)

Description
Set ICAVIT = 1 for damaged facets lying between two tessels K1 and K2.

Parameters In:
K1, K2 number of adjacent tessels

Parameters Out:
None

External Subroutines:
None

External Functions:
IFINDE find location of a point in contiguity list of another point

Local Variables:
IF12, IF21 auxiliary variables for entries of contiguity lists

C.8 Subroutine DRUNRR

Description
Auxiliary printing routine for printout of NRR1 and NRR2 together with IRCAV for every facet of the tessellation. Used only for debugging purposes.

Parameters In:
None

Parameters Out:
None

External Subroutines:
None

External Functions:
IFINDE find location of a point in contiguity list of another point
MODP calculate modulus of an integer with respect to IMOD

Local Variables:
IP loop counter
KSTART, KEND auxiliary variables
C.9 Integer function KM0 (KR,KL)

Description

Determine $\text{KM}(0)$, if not already available in a stored crack pattern.

Parameters In:

$\text{KR}, \text{KL}$ tessels adjacent to the first facet of the crack; $\text{KR}$ is at the right side and $\text{KL}$ is at the left side of the crack

Parameters Out:

None

External Subroutines:

None

External Functions:

$\text{ISTART}$ find first entry of contiguity list
$\text{IENDE}$ find last entry of contiguity list
$\text{IFINDE}$ find location of a point in contiguity list of another point
$\text{MODP}$ calculate modulus of an integer with respect to $\text{IMOD}$

Local Variables:

$\text{IBKR, IEKR, IFRL}$ temporary variables for beginning, end, and entries of contiguity lists
$\text{IPM, KMR}$ auxiliary variables
Appendix D

Data flow subroutines

Storage and retrieval of Dirichlet tessellations with or without damaged facets is performed with the help of the subroutines TSTORE and TLOAD, respectively.

Storage and retrieval of the corresponding crack patterns is performed with the help of the subroutines RSTORE and RLOAD, respectively.

Files containing the Dirichlet tessellation are organized in the following way:

1st line:
- seed for random number generator and value for damage level

2nd line:
- Number of points in the Dirichlet tessellation (NPUNKT), number of edges of the window (NRAHM), current bounds of the contiguity list of a point in the Dirichlet tessellation (IMOD) (must not exceed NMOD), current bounds of the contiguity list of an edge of the window (IMODR) (must not exceed NMODR)

3rd line:
- number of entries in the contiguity list of the edge # 1 of the window

4th line:
- contiguity list of the edge # 1 of the window

5th line:
- vertex coordinates of the edge # 1 of the window

The lines 3 - 5 are repeated for all NRAHM edges of the window. Line 3 * NRAHM + 3 contains the 1st line for point # 1 of the Dirichlet tessellation. For each point of the Dirichlet tessellation, there are 6 lines in the data file. An example is given in the introduction.

The first line contains the number of entries in the list together with the number of the tessel.

In the second line, the number of each neighbour (i.e. the contiguity list) is shown. Negative values indicate edges of the window.

The third line is for future use and will contain the marks for damaged facets (see below).

Lines four and five contain the vertex coordinates ECKX and ECKY of the first vertex of the facet (clockwise).

The last line contains the coordinates of the generating point of the tessel.

Files containing crack patterns are organized in the following way: For each crack, there are 7 lines in the data set.
1st line: 
   number of crack and number of crack facets
2nd line: 
   flag characterizing kind of nodes
3rd line: 
   number of grain located at the right side of the crack facet (NKR)
4th line: 
   number of grain located at the left side of the crack facet (NKL)
5th line: 
   number of grain located between NKR and NKL (NKM)
6th line: 
   x-coordinates of the crack nodes (DDNODX)
7th line: 
   y-coordinates of the crack nodes (DDNODY)

A consistency check is performed by the subroutines RLOAD and TLOAD to ensure that the maximum allowable array bounds are not exceeded.

D.1 Subroutine TLOAD

Description
Initialize all arrays and load a Dirichlet tessellation from file on unit 40.

Parameters In:
   None

Parameters Out:
   None

External Subroutines:
   None

External Functions:
   None

Local Variables:
   I, J, K loop counters
   JBEG temporary variable indicating the beginning of contiguity list (set to 0)
   JP, JR temporary variables indicating number of entries in contiguity list
   IPM, IPP auxiliary variables
D.2 Subroutine TSTORE

Description
Store a Dirichlet tessellation in file on unit 30.

Parameters In:
None

Parameters Out:
None

External Subroutines:
None

External Functions:
ISTART find first entry of contiguity list
MODP calculate modulus of an integer with respect to IMOD
MODR calculate modulus of an integer with respect to IMODR

Local Variables:
I, K loop counters
JBEG, JR, JP auxiliary variables

D.3 Subroutine RLOAD (NCRACK)

Description
Initialize all arrays and load a crack pattern from file on unit 50.

Parameters In:
None

Parameters Out:
NCRACK number of cracks contained in the crack pattern

External Subroutines:
None

External Functions:
None

Local Variables:
IUNIT FORTRAN file unit (set to 50)
I, IC, IK loop counters
IHILF temporary variable indicating number of facets of a crack
IERR error flag
D.4 Subroutine RSTORE (NCRACK, IUNIT)

Description
Store a crack pattern in file on unit IUNIT.

Parameters In:

NCRACK  number of cracks in pattern
IUNIT  FORTRAN file unit (set to 33, if IUNIT = 0)

Parameters Out:

IUNIT  (see above)

External Subroutines:

None

External Functions:

None

Local Variables:

I, ICRACK loop counters
IERR  error flag
IHILF  auxiliary variable
Appendix E

Auxiliary subroutines

Several auxiliary subroutines are available, e.g. for the calculation of the area of the grains of a Dirichlet tessellation (subroutine AREA).

Handling of crack interaction in the fracture mechanics description of crack patterns is done with help of auxiliary tesselations generated by subroutine ERSDZ which is described below.

E.1 Subroutine ERSDZ (NCRACK,ICONR,PXR,PYR)

Description
Generate auxiliary tessellation with the centres of gravity of the NCRACK cracks of an existing crack pattern as generating points. Provide list of neighbouring cracks for interaction effects. Subroutine ERSDZ utilizes the same arrays as the original algorithm. Auxiliary arrays therefore have to be provided to preserve the contiguity list and coordinate arrays of the original tessellation and to restore these after completion of the auxiliary tessellation.

Parameters In:

NCRACK number of cracks in the given crack pattern

Parameters Out:

ICONR list of neighbouring cracks
PXR array of x-coordinates for centres of gravity
PYR array of y-coordinates for centres of gravity

External Subroutines:

RAHMEN define window
ANFANG construct the contiguity list of the first point in the window
UPDATE construct the contiguity list of one subsequent point
DRUCK generate printout of the contiguity list of one point
DRURA generate printout of the contiguity list of one edge
VOLL determine the maximum number of neighbours in the contiguity list of all points of a tessellation
TSTORE Store a Dirichlet tessellation in file on unit 30

External Functions:
None

Local Variables:
I, J loop counters
IP counter for tessel which is presently being constructed
NPKTH temporary variable to preserve NPUNT
ICONH, ICORH temporary array to preserve ICONLI, ICORA
ECKEXH temporary array to preserve ECKEX
ECKEYH temporary array to preserve ECKEY
PXH, PYH temporary arrays to preserve PX, PY
NKI auxiliary variable (number of facets of one crack)

E.2 Subroutine AREA

Description
Provide list of area values for the grains of a given tesselation. The area of grains is calculated by summing up contributions from triangles given by centre and 2 consecutive points in the contiguity list.

Parameters In:
None

Parameters Out:
None

External Subroutines:
None

External Functions:
DIST calculate distance between two grain centres

Local Variables:
I, K loop counters
JP temporary variable indicating number of entries in contiguity list
JBEG temporary variable indicating beginning of contiguity list
GESAMT temporary variable for grain area summation
ZX, ZY coordinates of tip of triangle
L, L1 counters for base line points of triangle
NACHB neighbouring grain at base line of triangle
HOEHE height of triangle
U1, U2 auxiliary variables for height calculation
DNORM auxiliary variable for height calculation
BASIS auxiliary variable for height calculation