TopologyTracer Documentation

Release 1.2

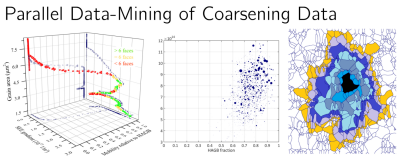
Markus Kuehbach

Dec 27, 2017

Contents

TopologyTracer is an MPI/OpenMP-parallelized datamining tool for conducting spatio-temporal analyses of microstructural element dynamics. Its purpose is the quantification of correlations — spatial and temporal — between individual microstructural elements and their higher-order neighboring elements. Currently, the software allows comprehensive studies of individual grains' volume evolution and set their growth history into relation to the evolution of their higher-order neighbors. The tool is unique in two ways: on the one hand in its ability to process these surveys in parallel and thereby handle hitherto intractable large datasets including millions of grains. On the other hand in its capability to explicitly take the spatial distribution of the long range neighborhood into account.

The source code was developed by Markus Kühbach during his PhD time with Luis A. Barrales-Mora and Günter Gottstein at the Institute of Physical Metallurgy and Metal Physics (IMM) with RWTH Aachen University. Being now with the Max-Planck-Institut fur Eisenforschung GmbH in Düsseldorf, I maintain the code, though at disregular intervals. Nonetheless, feel free to utilize the tool, do not hesitate contacting me for sharing thoughts, suggesting improvements, or reporting your experiences.



CHAPTER 1

In a nutshell

1.1 What is the TopologyTracer?

Currently, a software for the datamining of dump snapshots obtained from densely sampled instrumented grain coarsening simulations.

1.2 What are the user benefits?

Enabling studies with orders of magnitude more grains and thus higher statistical significance by parallelization

Unique capability of considering higher-order neighbors

1.3 What sets the TopologyTracer apart from other tools like DREAM3D or MTex?

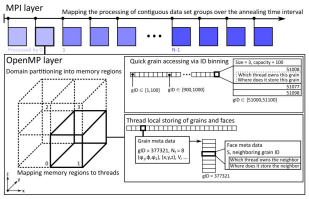
Spatio-temporal capabilities

While the aforementioned tools' strength is their ability to compute distributional descriptive statistics, like the grain size distribution or the orientation distribution function (ODF), they fall still short in their spatio-temporal and correlative capabilities. Additionally, they have much lower scalability as they do so far not employ parallelization. Thus, it remains the user's responsibility to add parallel functionality into the source code or set up batch processing queues via modifying the analysis scripts. Either way, this is not only a tedious task but in particular with respect to the scripting environment not straightforward to implement in a manner to advise the script interpreter exactly and in full control of the user were data should be stored to employ as much memory locality as possible: that is to assure that data which describe the microstructure in local regions are mapped as closely as possible into main memory such that their processing requires less costly memory queries.

Scalable efficient parallel design

By contrast, the TopologyTracer improves on this with its two-layer data parallelism. The individual datasets, each of which represents one time step, are elements in the coarser MPI-layer. Each MPI process hosts a contiguous set of all datasets over a particular time step interval. Additionally, each dataset is stored spatially partitioned into memory regions. These become mapped in such a manner as to reduce the number of remote memory accesses. It is exactly this data-locality-improvement-oriented design which suits potential thread-parallel execution with OpenMP for instance. Several pieces of this MPI/OpenMP-hybrid parallel scheme have already been implemented.

The figure illustrates this two-level parallelism.



In particular the logical and physical partitioning of the data in the second layer is fundamental to achieve scalable performance by improving data locality. This stands in clear contrast to many proprietary Materials Science software packages which still rely on traditional data structures commonly employed during sequential programming and their execution via scripting-language-based post-processing during which the complexity of data placement is opaque to the user. This opacity is the key to making perceivingly easily maintainable most Computational Materials Science scripting workflows. At the same time, though, it is the prime brake for achieving performance and thereby productive workflows - in particular for ever growing datasets. Admittedly, one could also employ thread parallelism without worrying about locality aspects. Beyond some problem-specific thread count threshold, though, this practice results in increasingly less performant execution as internally frequent remote memory accesses will be required. Exactly, these will then effectively saturate most contemporary memory infrastructures on not only super computers but also workstations and therefore prevent the quick delivering of data to the floating point processing units.

CHAPTER 2

References

2.1 The method

The method and the key to its implementation.

Kühbach, M.
On the Significance of the Long-Range Environment and Capillary Contributions for Nucleating Abnormal Grain Growth and Recrystallization
submitted to Acta Materialia, 2018

Kühbach, M., Mießen, C., Barrales-Mora, L. A., Gottstein, G.
Simulation and data-analytics of sub-grain growth with consideration of stored elastic energy and anisotropic grain boundary properties
Proceedings of the 6th International Conference on Recrystallization and Grain Growth 2016 (ReX & GG 2016) in the Omni William Penn Hotel in Pittsburgh, PA, U.S.
ISBN 978-1-119-32835-3

Kühbach M., Barrales-Mora L.A., Mießen C., Gottstein G.:
Ultrafast analysis of individual grain behavior during grain growth by parallel computing
Proceedings of the 36th Riso International Symposium on Materials Science
http://dx.doi.org/10.1088/1757-899X/89/1/012031

Further details are provided within my dissertation entitled

Kühbach, M.
Efficient Recrystallization Microstructure Modeling by Utilizing Parallel Computation
PhD thesis RWTH Aachen University
Successfully defended and accepted for publication in August, 2017

Finally open source published in January, 2018

2.2 Contact

Feel free to contact me and allow me to identify whether the TopologyTracer supplies functionalities which can cater also your analyses needs! Interested? Please contact me

2.3 Specifically about GraGLeS

Currently, the GraGLeS model is one of the very few grain coarsening simulation packages available that is capable of stressing the TopologyTracer.

Mießen, C. Velinov, N., Gottstein, G., Barrales-Mora, L. A.
A highly efficient 3D level-set grain growth algorithm tailored for ccNUMA architecture
Modelling and Simulation in Materials Science and Engineering
http://dx.doi.org/10.1088/1361-651x/aa8676

Mießen C., Liesenjohann M., Barrales-Mora L.A., Shvindlerman L.S., Gottstein G.
An advanced level set approach to grain growth – Accounting for grain boundary anisotropy and finite triple junction mobility
Acta Materialia, 2015, 99
http://dx.doi.org/10.1016/j.actamat.2015.07.040

Mießen, C.
A massive parallel simulation approach to 2d and 3d grain growth
PhD thesis RWTH Aachen University
Successfully defended in November, 2017

2.4 Third-party contributions

The TopologyTracer makes use of third-party contributions for some of its functionalities:

RapidXML: for the XML-based processing of control files and parameterization

Kalicinski, M.
http://rapidxml.sourceforge.net

Poly2Tri: for the triangularization of non-self-intersecting polygons

Liang, W.
http://sites-final.uclouvain.be/mema/Poly2Tri/

Robust predicates are required for numerical stable computational geometry. For these tasks Poly2Tri relies on

Shewchuk, J. R.
Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates

Computational Geometry, 1997, 18, p305
http://dx.doi.org/10.1007/PL00009321

Boost C++ library for probing the project folder structure to obtain meta data automatically

http://www.boost.org/

Eigen open-source library for performing various geometrical operations and the fitting of circles to point clouds for principal c

http://eigen.tuxfamily.org/

The detection of the intersection area between arbitrary triangles and a circle was inspired by the algorithm idea described in

http://stackoverflow.com/questions/540014/compute-the-area-of-intersection-between-a-circle-and-a-triangle/

Furthermore, I would like to acknowledge the work of S. Strobl et. al. who provided an open source algorithm to the numerical robust computation of the **intersection volume of an arbitrary tetrahedron and with a sphere**:

Strobl, S., Formella, A., Pöschel, T.
Exact calculation of the overlap volume of spheres and mesh elements
Journal of Computational Physics, Vol 311, 2016, p158
http://dx.doi.org/10.1016/j.jcp.2016.02.003

Additionally, I acknowledge the tetrahedralization project **TetGen** by Hang Si, which enabled me at least to probe in a substantiated manner the potential extension of the above-mentioned intersection problem in 3d:

Hang Si
TetGen A Quality Tetrahedral Mesh Generator and a 3D Delaunay Triangulator
http://wias-berlin.de/software/tetgen/

Its source code is not part of the TopologyTracer due to licencing issues but can be linked to the program, for which the interested user is referred to the source code.

CHAPTER 3

Setup

3.1 Which prerequisites are necessary?

The compilation of the program utilizes open source libraries as well as standard Linux tools.

- Check your Linux installation for a working installation of a **C/C++ build system** including **cmake** and **make**.
- Check your **C and C++ compiler**. We utilized successfully the Intel (v14.0 and newer) and the GNU (v4.8 and newer) compiler.
- You need a working installation of an MPI (Message Passing Interface) API library to be able to compile the program.
- We utilized successfully both the OpenMPI (v1.10.4 and newer) and the IntelMPI (2017.0.12) implementation.
- The minimum threading support level of the MPI implementation required is **MPI_THREAD_FUNNELED**.
- You need a working installation of the **Boost C++ libraries**. Details on where to get it and how to install are found here.
- The shipped-with default version of Boost of at least Ubuntu 16.04 provided to me all functionality required.
- You need the **Eigen libraries** that come along with the TopologyTracer package. Further information to this library is provided.

3.2 Which prerequisites are optional?

- It is well-known that the general purpose standard malloc(3) memory allocator class implementation does not assure that in particular many small allocations become locality-aware placed. Hence, the performance of the TopologyTracer can be improved by linking against an alternative memory allocator class, such as **Jason Evans jemalloc**. A documentation of how to obtain the library and how to compile it can be found online.
- In the future, an OpenMP-parallelization and HDF5 extension of the TopologyTracer is planned. So far however HDF5 is not required to run the TopologyTracer.

3.3 Where to get source code from?

TopologyTracer is available for Linux only. It is free software.

- Download the source from its git repository **https://github.com/mkuehbach/TopologyTracer**
- Eventually unpack the repository such that finally the following ends up in a single folder.
- This folder from now on will be called the **root** directory. You can give it any Linux-conformant name.
- Make sure in this root there is a **src** subdirectory with the cpp and the h source code files,
- a **build** directory for storing the executable and an exemplary **XML control file**.
- Additionally, check whether there is a **CMakeLists.txt** file in the root folder.
- Next, utilize the top section of this CMakeList.txt file to switch on and off the compiler (GNU or Intel)
- Next, open a console and dive into the **build** directory.
- If now its the first time you compile the TopologyTracer type **cmake ..**.
- This inspects your system and generates a customized makefile for you.
- Next use this makefile by typing **make** to compile the program.
- Upon success you should now have a **binary** within build called **topotracer2d3d**

3.4 Placement of files

The resulting executable expects the XML control file always in its current location folder!. Relative indexing is utilized. Other than that restriction, the executable can be renamed and relocated. The latter enables the scripting of TopologyTracer job queues and executing batch jobs.

3.5 Optimization

If desired, adjust the level of compiler optimization via the OPTLEVEL variable in the CMakeLists.txt upper section. OPTLEVEL "-O0" means no optimization and should be utilized for debugging purposes only, while "-O3" is the maximum and recommended level for production tasks. Improvements between the two extremes vary between a factor of 2 - 5 faster with maximum optimization compared to without.

3.6 Troubleshooting?!

If in between the compilation process unrecoverable errors occur, attempt first a **make clean** command. If this does not help: Delete everything in the build folder except for the **TopologyTracer2D3D_Parameter.xml** control file and start over with **cmake ..**.

Utilizing the GNU compiler? If the __int64 definition is unknown to the preprocessor and the code therefore does not complains, does not compile: define it as typedef long __int64 in the **TopologyTracer_Topology.cpp**.

CHAPTER 4

2. Creating input

4.1 What is TopologyTracer input?

A temporal sequence of microstructure dynamics snapshots. Currently, the TopologyTracer requests each of these snapshots as to be pairs of two binary files: a **Faces_*.bin** and a **Textures_*.bin** file. The asterisk placeholder (*) stands for a set of integer numbers separated by a constant offset, i.e. (1,2,3,4) or (10, 20, 30, 40) for instance, on an interval [SnapshotFirst,SnapshotOffset,SnapshotLast]. As the format of these files is generic, it is possible to feed either in-situ microstructural data from experiments or check pointing data from simulations. Currently, the TopologyTracer is implemented to work with input from the GraGLeS level-set code (https://github.com/GraGLeS). Feel free contacting me when there is interest in adding input routines for other tools. I am aware that conventions within our community differ so I am highly interested in offering such alternative entry to use my code.

4.2 Binary file format layout

The TopologyTracer utilizes binary data in order to provide accuracy, to minimize data storage costs, and to reduce I/O time via a one-file-per-process concept. During I/O operations all binary files are interpreted assuming

- LittleEndian byte order
- The default MPI view of the file is as of a linear byte stream
- Namely, displacement = 0, etype = MPI_BYTE, filetype = MPI_BYTE

In what follows a 64-bit C/C++ double (precision) floating point value reads as **d**, a 32-bit integer as **i**, a 32-bit unsigned integer as **ui**.

4.2.1 Grain boundary face data via Faces files

The **Faces_*.bin** file is headerless and contains all disjoint grain boundary faces. It reads as a contiguous block of as many MPI structs as faces exist. Each struct specifies the segment length and the two disjoint integer IDs A, B (with A > B >= 0). These IDs reference two grains that share the boundary face. As such, each boundary is stored only once. Currently, its description occupies 16B on a 64-bit machine. A single entry reads as follows:

```
d ui ui
```

4.2.2 Grain meta data via Texture files

The **Texture_*.bin** file is headerless as well. It contains the properties of all grains at the timestep, i.e. size, barycenter position to name but a few. The binary file encodes a contiguous block of as many MPI structs as grains remain. Each MPI struct encodes the properties of a disjoint grain. **The struct layout differs for 2D and 3D data in an effort to reduce the amount of data!**

In **2D** the order is as follows: Area, Perimeter, GBEnergy, Stored elastic energy, Bunge-Euler phi1, Bunge-Euler Psi, Bunge-Euler phi2, Barycenter X, Barycenter Y, GrainID, First-order neighbor count, Contact with boundary yes/or no, padding.

In **3D** the order is as follows: Area, Perimeter, GBEnergy, Stored elastic energy, Bunge-Euler phi1, Bunge-Euler Psi, Bunge-Euler phi2, Barycenter X, Barycenter Y, Barycenter Z, GrainID, First-order neighbor count, Contact with boundary yes/or no, padding.

The MPI struct encodes these pieces of information as follows:

```
d d d d d d d d ui ui ui ui

d d d d d d d d d ui ui ui ui
```

Consequently, a grain occupies 88B in 2D and 96B in 3D, respectively on a 64-bit architecture when utilizing the default byte order.

4.2.3 Illustrative example of expected datavolume

For a 3D polycrystal with 500,000 grains spanning a boundary network of 3,500,000 individual faces a single snapshot requires ((5.0e5 * 96B) + (3.5e6 * 16B)) / 1024 / 1024 KB/B MB/KB = 99.2MB disk space. For comparison, a practical peak bandwidth of a Lustre parallel file system for a one-file-per-process I/O scenario with a few MPI processes ranges typically between 1000-15000 MB/s.

CHAPTER 5

3. TopologyTracing

TopologyTracing requires at least to have

1. A collection of ID contiguous binary data file pairs, i.e. Faces_*.bin and Texture_*.bin binary files,
2. The topotracer2d3d executable,
3. A XML control file all in the same folder.

Several analysis routines require additional files as it is explained for many in the input section.

With these prerequisites, TopologyTracing, i.e. the datamining of simulation data, requires two steps:

1. Setup the control file
2. Specify the post-processing tasks

Both tasks are accomplished via the XML control file:

5.1 XML Control File Settings

The entire datamining is controlled by only one control parameter settings file — the **Topology-Tracer2D3D_Parameter.xml** file. All angular values are expected in degrees!

5.1.1 Analysis mode

AnalysisMode

Determines which execution model and fundamental tasks should be executed.

Option (1) (the default) is for tracing data in parallel. It distributes an ensemble of Face/Texture data on the MPI processes.

Option (2) tracking sequentially by identifying first all grains close to boundaries, thereafter loading successively the files for processing.

Make sure to populate GrainBoundaryContact with a file to be generate/or existent that stores the deadtimes of the grains.

You should switch on **ProbeBoundaryContact** to generate such file and switch it off once done and the file exists.

Option (3) reads sequentially a TargetGrainIDs file and a KNNFile to correlate the evolution of a grain with a prognosis of their evolution based on the k-th neighborhood in the initial structure (SnapshotFirst). The functionality is UNDOCUMENTED.

Option (4) reads binary GBContourPoints_<fid>.bin files to compute unbiased quantitative metrics for each nucleus to its long-range environment in 2d. The functionality is UNDOCUMENTED.

Option (5) is the currently only MonteCarlo discretely implemented extension of this in 3D. The functionality is UNDOCUMENTED.

Option (6) reads binary GBContourPoints_<fid>.bin files to approximate for each grain the capillary driving force by evaluating the turning angles at the piecewise segmented grain boundary contour. The functionality works at the moment only for 2d contours.

Option (7) is UNDOCUMENTED.

5.1.2 Fundamental settings

SnapshotFirst

Specifies the integer index of the first dataset to analyze.

SnapshotOffset

Specifies the constant integer increment of datasets desired,

i.e. tracing is performed on the interval [First, Last] in increments of Offset.

SnapshotLast

Specifies the integer index of the last dataset to analyze.

All indices are integer, >= 0. First and Last can be set to the same value to analyze only one dataset namely the SnapshotFirst.

Dimensionality

Is the dataset describing 2d (2) or 3d (3) simulations?

LargestGrainID

The user has to specify the maximum grain ID in the entire dataset.

ProbeBoundaryContact

You should switch this on only when in mode 2 to analyze once the dead times of the grains into a file named **GrainBoundaryContact**.

5.1.3 Physical properties

HAGBMobility

Specifies the **maximum** mobility of all grain boundaries (m^4/Js).

HAGBEnergy

Specifies the **maximum** specific energy of all grain boundaries (J/m^2).

DislocEnPerM

Specifies the product 0.5Gbb,i.e the dislocation line energy per unit length dislocation (J/m).

PhysicalDomainEdgeLength

Specifies the real distance which the simulation domain with InitialDomainEdgeLength represents (m).

InitialDomainEdgeLength

Specifies the integer edge length of the simulation domain to the beginning of the simulation (square in 2D, cube in 3D).

Mind that all these settings have to meet those in the coarsening simulation!
Especially, LargestGrainID is required as the largest unsigned integer grainID
This is not necessarily the total number of grains in the simulation!

5.1.4 Datamining operations

The following options define which analyses should be executed (1) or not (0).

AnalyzeGrainSizeQuantiles
Computes descriptive means of the grain size (area, vol) and the volume-average quantiles for each Snapshot.

AnalyzeSEE
Approximates the CDF of the stored elastic energy in the grains in an area/volume averaged manner.

AnalyzeMODF
Approximates the CDF of the MODF (disorientation angles in boundary network (face segment length/area averaged)
This is not the MacKenzie plot (probability density of occurrence over disorientation angle) but its integrated form!
Only considers boundaries between grains which both (instantaneously) do not make simulation domain boundary contact.
GraGLeS outputs also an MODF, via an file MODF_* ASCII file. Numerical differences exist.

AnalyzeGSD
Approximated a histogram of the population arithmetic average normalized distribution of grain sizes (area, volume).

AnalyzeMaxSizeGainForward
Computes for all grains the maximum size (area/volume) they obtain in all Snapshots.

AnalyzeApproxRXFraction
Determines two populations of grains: one with all grains which never touched the simulation domain boundaries
and another from those remaining in SnapshotLast (targets).
Therefrom it is computed the instantaneous coverage ratio (area, volume) of these targets with the population.

AnalyzeTrajectoriesBackward
Extracts a list of those grains in the last timestep (SnapshotLast) which not made contact with the simulation domain boundaries.
These grains are then followed backwards in time. This is the suggested mode of operation to follow the evolution of persistent grains.

AnalyzeTrajectoriesForward
Extracts first all grains which never during their existence, i.e. (size > 0) made contact with simulation domain boundaries if any.
Then, their properties are tracked forward in time. If a grain disappeared earlier than SnapshotLast,
all succeeding result values for within the column vector are set to zero.
Mind that the resulting output can be very large as a total of 10 million grains initially tracked for 1000 snapshots will result in a matrix of 10 billion doubles!

AnalyzeAbnormalGrains
Determines three populations of grains, all of which never touched simulation domain boundaries.
One from SnapshotLast (the possible AGG candidate grains).
One from all grains.
One of all the latter grains but excluding the candidates (the matrix).

It is evaluated for each time step the (spherical equivalent radius ratio) of the candidate grains to the matrix.

AnalyzeKNN

Determines the k-th order (nearest) neighbors to a target of the grains. Each grain of the dataset is inspected independently

and resulting in a successful detection only if none of the neighbors made contact with domain boundaries to prevent bias.

During this neighborhood analysis also the region boundary length is computed.

Between the 0-th and the 1-th order neighbor this is the grain boundary perimeter of the target itself.

Also the fraction of high-angle grain boundaries is computed. WORKFLOW IS UNDOCUMENTED!

The following functions are currently UNDOCUMENTED

AnalyzeDrivingForceSEE
AnalyzeMeanDrivingForceSEEForward
AnalyzeSizeGainVsMatrixBackward
AnalyzeSizeGainVsMatrixForward
AnalyzeClassicalNucModels
AnalyzeTopologyDifferenceForward

5.1.5 Datamining operation specific settings

Datamining is performed on the timestep interval [SnapshotFirst, SnapshotLast] in increments of SnapshotOffset. Grains with contact to the simulation domain boundaries (special grain ID 0) are expelled from the analyses in every case by default. The code allows for the implementation of utilizing results from simulations that were conducted under periodic boundary conditions to enable higher statistical significance. The latter is of particular interest for 3D simulations and **in particular when quantifying the higher-order neighbors**. The results are either stored in ASCII files (***.csv**) or as binary matrices 2D (***.bin**). The latter have no header but a **speaking filename** (<whatever>.F.<SnapshotFirst>.O.<SnapshotOffset>.L.<SnapshotLast>.NC.<ncols>.NR<nrows>.bin) which specifies the number of columns (NC) and rows (NR), respectively. In general, columns are column vectors, each of which encoding all grains at one time step, while rows are row vectors which specify properties of one grain along all time steps. With these definitions the binary results file read as a 2D matrix (grains-time) in implicit form by aligning entire column vectors according to their row ID contiguously in memory.

AnalyzeTrajectoriesForwardMode

Modifies AnalyzeTrajectoriesForward, when unset (0) all grains are tracked, when set (1) the TopologyTracer requires TargetGrainIDs to be set in order to perform tracking only on these included IDs. Then an additional file needs to be supplied (**TargetGrainIDS**).

MaximumNumberOfKShells

Specifies the maximum order of (long-range) neighbors during the analysis AnalyzeKNN.

HAGBDetectionThreshold

When is a grain boundary considered as to be of high-angle character?

Sensible default in accordance with Read-Shockley theory is 15 degrees.

DisoriAngleBinningMin
DisoriAngleBinningMax
DisoriAngleBinningWidth

The values specify the binning of the MODF.

StoredElasticEnergyBinningMin
StoredElasticEnergyBinningMax
StoredElasticEnergyBinningWidth
The equivalent binning specification for the stored elastic energy density.
GSDBinningMin
GSDBinningMax
GSDBinningWidth
The equivalent binning specification for the mean-normalized size histogram.

PersistenceRadiusMin
PersistenceRadiusIncr
PersistenceRadiusMax
Radius (normalized to RVE domain edge length) of the fixed size reference window about each target for which the intruding neighboring is computed. The incrementer allows to setup a basic loop to sample differently sized reference windows within [Min,Min+Incr,Min+Incr+Incr,...,Max].
UDSFile
A uds file specifying the properties of all the grains in the synthetic structure. Is written by the microstructure generator.

ComputeCurvatureAlsoAtTJP
Flag to overwrite the default behavior that for supporting points close to junctions the curvature is not computed into a scalar average of the grain. See supplementary material to the long-range environment reference paper for more details.
TranslateBinary2GNU
Does what it reads at the cost of potentially significant disk space.

CapillaryActivityTargets
Allows to reduce list of target grains to specific user-defined set of IDs rather than to operate on grains of all IDs in the input data.
CapillaryActivityMode
Without digging into source code, leave with 0.

5.1.6 Auxiliary files

GrainBoundaryContact
See explanation for analysis mode 2 and ProbeBoundaryContact.
TargetGrainIDs
Enables to supply a headerless ASCII file of grainIDs (only positive 32-bit unsigned int IDs on interval [1,LargestGrainID] are interpreted) in order to restrict analyses to specific grains.
Specifically, when AnalyzeTrajectoriesForward is chosen (1)
and AnalyzeTrajectoriesForwardMode set to tracking forward (1) in time the forward tracking is performed only for
these targets and not the entire population to reduce the overall size of the output.
KNNFile
enables to supply a single line header-equipped ASCII file with the layout grainID, x, y

coordinate for a currently UNDOCUMENTED option.

GNUFile

A gnu 2D grain boundary contour file. The functionality is UNDOCUMENTED and DEPRECATED.

5.1.7 Additional settings

OnlyProbeTheWorkPartitioning

In mode 1 this option allows to execute only the dataset partitioning planning without reading the heavy data.

This allows to plan the allocated load partitioning and to verify that all

processes get snapshot data volume as equally as possible distributed.

LocalDatabaseMaximumSize

Controls the maximum size of binary data (Faces, Texture) which a single MPI process accepts to store and process in AnalysisModes 1 and 2.

At the moment the workpartitioning works as follows: i) it computes the total size of all binary file pairs, ii) then it partitions on the MPI processes.

These do usually **but not necessarily handle an equal number of Faces,Texture binary file pairs!**

In every case the processes neither handle incomplete snapshots nor do they share or duplicate snapshot data!

Namely, either they handle as many datasets (pairs Faces/Texture) as their capacity allows. If their capacity is exceeded,

the next process accumulates snapshots, until either there are no more snapshots to distribute and the processing can start or the program has to exit in a controlled manner because all processes have already been exhausted and should no longer load snapshot data. Hence, the value must be chosen with care. The idea behind this concept is to enable the user to set an upper bound on how much memory the MPI processes have to supply at least to load snapshot data and thus prevent the program from flooding memory or running in above hard memory limits when running in batch queues.

The design implies that the more data per process are set the more processes with the higher rank IDs will idle as the partitioning

of snapshots proceeds in the order of the MPI process ID (rank). On the contrary, if the value is too low such as that they do not

allow to partition the entire ensemble, the post-processing terminates.**

MPIReadBlockLength

Sensible default of the MPI I/O buffer size.

MaxIDRange

Sensible default for how many contiguous grain IDs are stored per IDBucket. The smaller this value is chosen,

the more buckets are required but also the proportionally lower it is the time to find a specific grain.

Clearly, the value is performance relevant, as the larger the value is set the proportionally more grains have to be

scanned on average before finding a specific. On the contrary, a very small value will reduce the total

number of checks but eventually may not end up in pointing to aligned pieces of memory, unless additional measures of

improving contiguity (with for instance the jemalloc allocator class) are enforced.

Hence, the value should be in the order of the size of single cache block times size(unsigned int). 100 is a sensible default.

The choice to go for a concept of IDBuckets rather than a hashfield of pointer is to avoid the

storing of many NULL pointer for non-existent IDs for those snapshots in which only few IDs remain.

MemRegionsX

MemRegionsY

MemRegionsZ

Required to be 1

An integer spatial partitioning of the domain into regions. All grains whose barycenter are in the region are handled by an independent memory handler that is a local memory region class object. This improves memory locality for a future

OpenMP extension of the code, as then shared memory accessess within the region are more likely found in local caches.

DeveloperMode

With this option enabled (1) each binary file is translated into an ASCII file.

Make sure to disable this option (0) whenever running production jobs!

5.2 Program execution

5.2.1 MPI only

All set? Excellent! Then, the tracing is executed via one command line call:

```
mpiexec -n <nprocesses> <topotracer> <simid> <TopologyTracer2D3D_Parameter.xml>
↪<optional: arguments>
```

Please note that the angle brackets must not be typed into the command line as they only mark the input parameter!

In its current version the following input arguments are required:

- <nprocesses> How many MPI processes to utilize?
- <topotracer> The name of the executable.
- <simid> JobID, a positive integer to distinguish the datamining results from runs with other settings but the same raw data.
- <TopologyTracer2D3D_Parameter.xml> a properly formatted XML control file. The name can be changed as long as the file remains a properly formatted XML file.

Be careful: if the <simid> value is set to the same value during subsequent runs in the same folder, data will be overwritten without prompting!

Below is a typical example call which executes the program with 10 MPI processes, reads from MySpecialSettings what should be done and tags all output with a consistent run ID, 1000 in this case:

```
mpiexec -n 10 topotracer2d3d 1000 MySpecialSettings.xml
```

5.2.2 MPI/OpenMP

Core functionalities within the analyses modes 4, 5, and 6 are additionally OpenMP thread parallelized. For this, the OpenMP environment variable OMP_NUM_THREADS must be set to the desired number of threads spawned at most per MPI process. Otherwise the program call is as above:

```
export OMP_NUM_THREADS=10
```

When utilizing exemplarily the Intel compiler it is usually worthwhile to reconsider the thread placement on the computing cores. One can guide this operating system decision by for instance the KMP AFFINITY environment.

5.3 Compile a growth history for individual grains

Consider the Matlab scripts in the *scripts/* folder.

Please note that these scripts provide only the key functionalities to import TopologyTracer results data into MATLAB. Hence, they require adaptation to the analysis target desired.

These scripts are the originals I used for post-processing the long-range effects paper results:

> Kühbach, M.
> *On the Significance of the Long-Range Environment and Capillary Contributions for Nucleating Abnormal Grain Growth and Recrystallization*
> submitted to Acta Materialia, 2018

5.4 Output data format specification

The TopologyTracer generates a number of binary results file which can be visualized and processed with Matlab. These files do not include the simulated or scaled real physical time. **That one has to be supported manually as a part of the Matlab analysis!**

In what follows, a 64-bit C/C++ double precision floating point value reads as **d**, a 32-bit integer as **i**, and a 32-bit unsigned integer as **ui**. The byte order is read and written assuming LittleEndian. MPI I/O views by default all files with displacement = 0, etype = MPI_TYPE and filetype = MPI_BYTE. The TopologyTracer follows this default. All files encode as 2D matrices of an elemental datatype (either d,i, or ui, or an MPI struct) as a number of nc columns of a contiguous block (row) of size nr each. All means are arithmetic mean values.

The results filenames are speaking with NR = nr and NC = nc!

Mind that in all what follows the statement all grains means explicitly all grains filtered for the analysis. This number is not necessarily equal to the number of grains in the population, because if the simulation is performed with open boundary conditions the grains touching the domain boundary have been expelled from the analysis! In general the TopologyTracer assumes that the grain with ID 0 is a special grain, namely a proxy ID for the RVE domain itself. Equivalently, we can state that the TopologyTracer utilizes Fortran ID labeling for grains and faces.

5.4.1 analyze_vol_quantiles()

Writes two files **VolQuantiles** which specifies the n = STATISTICS_VOL_NQUANTILES (currently 100) (1/n,2/n,...,n/n) quantile of the distribution of grain size (area (2D), volume (3D)). Specifically, nr = n and nc = (SnapshotLast - SnapshotFirst)/SnapshotOffset + 1. The elemental datatype is d.

The file **VolMeta** provides descriptive statistics with nr = 1 and nc as above. The elemental type is a contiguous MPI struct of five d. They encode

- The total area covered by all grains considered
- The mean size
- The variance
- The total number of grains in the snapshot
- The total number of grains considered (for mean and var)

5.4.2 analyze_vol_forward()

Writes four 2D matrices, all with the same layout. Except for **FW.NF**, whose elemental datatype is ui, all others have d.

- **FW.NF** is the number of neighbors
- **FW.VOL** is the size of the grains (micron^Dimensionality)
- **FW.HAGBFRAC** gives fraction of boundary (length (2D), area (3D) with disorientation of target and neighbor >= HAGBDetectionThreshold
- **FW.MOBDSEE** gives area-weighted product of mobility * (stored elastic energy nbor - stored elastic energy target) (m/s)

Values for no longer existing grains get filled up with zeros!

5.4.3 analyze_vol_backward()

Monitor for all targets (grains which survived the coarsening step up to SnapshotLast) backwards in time for the evolution of their faces, size, HAGB fraction, and instantaneous biased velocity. **Even though tracking is performed backwards, the organization of the matrix is positive in time, i.e. higher column vector indices mean later in time.**

The layout and content is the same as for analyze_vol_forward(). The only differences are the file name suffixes.

- **BK.NF**
- **BK.VOL**
- **BK.HAGBFRAC**
- **BK.MOBDSEE**

5.4.4 analyze_sizegain_vs_bk()

Idea: for all survivors identify in each snapshot how their size is relative to the mean of the matrix. The matrix constitutes of all grains excluding the survivors. Writes one 2D matrix of elemental type d. Each first entry in each row describes the number of matrix grains remaining, while the second entry the mean size of the matrix grains.

- **SIZEGAINBK**

5.4.5 analyze_approx_rxfraction()

Calculate approximate recrystallized area/volume fraction assuming the following. Matrix grains constitute all grains. The recrystallized grains are all those which survive the coarsening, i.e. all remaining up to SnapshotLast. A 2D matrix with nr = 1 is written. The elemental datatype is a contiguous MPI_struct of d. These detail

- The total size of all grains considered in the analysis
- The total size covered by the targets, i.e. the considered as recrystallized
- The recrystallized coverage as TotalSizeTargets/TotalSizeAllGrains
- The total number of all grains considered in the analysis but were still detectable in the snapshot
- The total number of rxgrains

5.4.6 analyze_modf()

Writes a 2D matrix of doubles (d) which encodes the disorientation distribution function (MDF) with equiangular adjustable binning, via nr = largest positive integer (MaxDisoriAngle/DisoriAngleBinWidth) and nc = 1 + ((SnapshotLast-SnapshotFirst)/SnapshotOffset+1). The first row of the matrix gives the bin ends in degrees. Thereafter, the MDF for each snapshot follows.

5.4.7 analyze_see()

Same layout as analyze_modf. First row gives bin ends. Thereafter, each snapshot.

5.4.8 analyze_gsd()

Equal in mentality and data output structure to SEE and MODF.

5.4.9 analyze_abnormal_graingrowth()

Writes a 2D matrix of structs containing 13 doubles (d) each which encode the following

The total size of all grains in the list of

- survivors
- matrix
- matrix exclusive survivors

The equivalent radius of

- survivors
- matrix
- matrix exclusive survivors

The remaining number of grains in

- survivors
- matrix
- matrix exclusive survivors

The number of abnormal large grains, i.e. whose equivalent radius is larger than 3.0 the average of the individual populations

- matrix exclusive survivors abnormal when testing against mean of matrix
- matrix exclusive survivors abnormal when testing against mean of matrix exclusive survivors
- survivors abnormal when testing against mean of matrix
- survivors abnormal when testing against mean of matrix exclusive survivors

Mind that when approaching SnapshotLast the population of matrix grains ceases because this understood as the matrix into which the survivors grow.

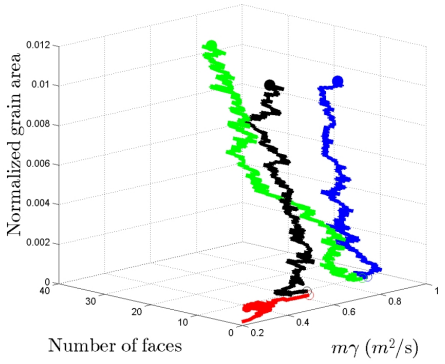
5.4.10 Currently undocumented

- analyze_grainsize_quantiles()
- analyze_drivingforce_see()
- analyze_topologydifference_forward()
- analyze_classical_nucmodels()
- analyze_knn_naive()

CHAPTER 6

4. Visualizing results

In its current state, the TopologyTracer does not contain a GUI. Instead, plain MPI I/O raw files and ASCII files are generated which require post-processing with for instance Matlab scripts available in the **scripts** folder. In general one can visualize a growth path by the plot3(X,Y,Z) command and access a distribution of values via the histogram function. Therein X, Y, Z specify each a row vector of length NC for a particular grain ID in row r.



CHAPTER 7

Funding

The authors gratefully acknowledge the support from the DFG in the frame of the Reinhart Koselleck project (GO 335/44-1).

Furthermore, we acknowledge the support from the FZJülich and RWTH Aachen University within the JARAHPC research alliance.

In addition I would like to acknowledge the provision of computing resources by the Max Planck-Institut für Eisenforschung GmbH.

CHAPTER 8

Version history

v1.2

Implementation of tracking mode 2 which does not require any longer the entire dataset to be loaded in main memory, but
instead reads successively files for analyses. This patch makes the program also applicable to low main memory workstations.
Implementation of 3D extension to unbiased longer range environment measures via Monte Carlo sampling,
curvature estimation via binary contour line data single-grain-resolved for arbitrary number of time steps,
binning of stored elastic energy in grain population, tracking of maximum size of grain during its existence.
Bug fixing, implementation of allocation error handling, performance updates in particular beneficial for handling
hundred thousand grain ID long lists of candidate grains for population categorization.
Clean-up of source code debris and documentation of cmake script.
Clean-up of and provision of Matlab-based analysis scripts for post-processing and their core documentation.

v1.0

Successful switch from ASCII- to binary file-based workflow, locality-aware data organization, and tree-based
data storage improved raw data I/O by orders of magnitude. Furthermore, it made collecting random grain properties
orders of magnitude more efficient. Novel algorithm to identify k-th order neighbors makes now possible the tracking
of arbitrary k-th order neighbors. Novel analysis routines make possible the comparison to classical theory of abnormal grain
growth and classical nucleation criteria (Bailey-Hirsch, sub-grain size distribution spreading to name but a few).
Additionally, the code supports now functionality to compute analytically the triangle-circle intersection area.

v0.1

Implementation of first set of functionalities to obtain temporal trajectories in topology/mobility/energy-phase space for the Riso 2015 conference with simple neighbor correlations and disorientation.

CHAPTER 9

Licence

9.1 The project is licenced under the GNU v3.0

TopologyTracer is an MPI-parallel datamining tool for the conducting of spatio-temporal analyses of microstructural element dynamics. Its purpose is the quantification of correlations — spatial and temporal — between individual microstructural elements and their higher-order neighboring elements. Specifically, its current functionalities allow to study the volume evolution of individual grains over time and set their growth history into relation to the evolution of the higher-order neighbors. The tool is unique insofar as it allows processing these individual surveys in a parallelized manner. Thus, enabling the post-processing of so far intractable large datasets.

The source code was developed by Markus Kühbach during his PhD time with Luis A. Barrales-Mora and Günter Gottstein at the Institute of Physical Metallurgy and Metal Physics with RWTH Aachen University. Being now with the Max-Planck-Institut fur Eisenforschung GmbH in Dusseldorf, I maintain the code, though at disregular intervals. Nonetheless, feel free to utilize the tool, do not hesitate contacting me for sharing thoughts, suggesting improvements, or reporting your experiences. markus.kuehbach at rwth-aachen.de and m.kuehbach at mpie.de

The authors gratefully acknowledge the financial support from the Deutsche Forschungsgemeinschaft (DFG) within the Reinhart Koselleck-Project (GO 335/44-1) and computing time grants kindly provided by RWTH Aachen University and the FZ Jülich within the scope of the JARAHPC project JARA0076.

CHAPTER 10

Questions, contributions

Just let me know or contact *m.kuehbach@mpie.de*

CHAPTER 11

Future plans

v1.3

Documentation, 3D curvature computation, triple line tracking