TAPSim How-to

Version 1.0

Christian Oberdorfer*

February 2014

Institute of Materials Physics

University of Münster

Wilhelm-Klemm-Str. 10
48149 Münster, Germany

Atom Probe Tomography (APT) is an amazing analysis technique in materials science, which enables the analysis of local chemistry and atomic positioning in nanostructured materials. It offers outstanding spatial resolution in combination with sufficient mass resolving power for the determination of chemical species. The basic principle of this method is founded on the effect of controlled evaporation of single atoms from a field emitter. Due to the presence of extraordinary high field strengths in direct vicinity of the apex, single surface atoms can desorb and become ions. The needle shaped emitter represents at the same time the analysed sample and it acts as the most important part of the projective system for produced ions since it decisively effects the trajectories.

TAPSim is a simulation package which has been developed to better understand the effects which arise in APT measurements of samples with heterogenous evaporation properties. It allows the theoretical analysis of tips with arbitrary shape, with arbitrary atomic structure and well-defined chemical composition. In this regard, practically no limitations exist. Each tip atom is represented by a Wigner-Seitz cell.

The produced output by the simulation can be analysed with standard APT analysis techniques as they are applied to real experimental data, too. For this reason TAPSim enables a one on one comparison with a theoretical model of a sample and with direct measurements. From a theoretical point of view it offers the possibility for testing new concepts and procedures for APT

*oberdorc@uni-muenster.de

data treatment in a reliable and controlled way. Hopefully, this approach helps to improve the quality of the 3D reconstruction procedures as well as it will readily help to reveal and understand artifacts in present reconstructions of the measurement data.

The TAPSim package is available as open source software. Within the constraints of the GNU Public License (GPL) it may be freely distributed and modified.

1 Prerequisites

The TAPSim code is written in standard C++. It is entirely open source. Besides the C++ Standard Template Library (STL), external dependancies on Hang Si's *TetGen* package [Si06] and Shewchuck's "Fast and Robust Geometric Predicates" [She97] exist. Both packages are freely available. In order to avoid conflicts originating in different library versions at this point, appropriate versions of these packages — with some minor modifications — are included in the TAPSim distribution. Nevertheless, if the use of more recent versions as the included ones is needed, it remains an opportunity.

For multithreading the POSIX Threads API (implemented by the "thread" c-library) is used, which is the only hard prerequisite for compiling TAPSim. Parallelization is not essentially mandatory, but in order to ease the effort for maintenance currently compilation of TAPSim to a native single threaded version is offered.

The code should compile and run on any x86 based system, either 32 bit or 64 bit. However, the development was carried out on a x86-64 linux system using the GNU C++ compiler and nothing else has been tested so far. Usage of the mentioned or a similar 64 bit unix based platform is recommended.

2 Download and compilation

The distributed package comes in the form of a *bzip2* compressed archive. For the most recent version please visit the TAPSim homepage[1].

After download follow the instructions below for compilation. It is assumed that you are working within a unix environment and execute the commands from a terminal console:

- Decompress the obtained archive with *tar*. Switch to the directory in which the downloaded package has been placed and enter: `tar␣xfj␣tapsim_v*.tar.bz2`. Here the asterix is a placeholder for the actual name of the distributed version. Despite the file suffices, this name may incorporate additonal labels which mark the version in detail.
- When finished, the decompressed content should have been placed in a newly

[1] http://www.uni-muenster.de/physik.mp/schmitz/tapsim

created directory which has the same name as the archive but without the suffices. Switch to this directory (`cd␣...`).

- In the TAPSim-directory you find at least a licence file ("license.txt") containing the GNU GPL licence agreement and a makefile. A properly configured build system provided, the code may now be compiled, just by invoking the make command which by default processes the file "makefile": enter `make`. Hopefully no errors occur. Now, the final binaries of the compiled package should have been placed into "./bin" subdirectory.

Any documentation, as far as it is provided, will be available in the "./doc" subdirectory of the decompressed package.

3 Usage

The compiled package consists of two programs: one is a utility, simply called *MeshGen*. It uses the mesh generation abilities of *TetGen* in order to construct an adaptive mesh covering the mesoscale between a user provided sample structure and a preset simulation environment. The other program is the TAPSim binary itself, which executes the simulation. It computes the potential distribution for an input mesh and carries out the consecutive evaporation of atoms from a field emitter.

3.1 User provided files

In order to conduct a simulation, you need at least to provide the atomic structure of the field emitter in a separate file (called a "node file") with special format. This file basically consists of the coordinates of the atoms and supporting points. A full-scale mesh suitable for atom probe simulation can be constructed from this input with the support of the *MeshGen* tool. This approach will be described in section 3.2.

In addition to the structural node file information, a second "configuration" file must be provided. It contains information about distinguished nodal properties of the mesh. Both files are required to launch a simulation run. The respective file formats will be described in the following sections.

3.1.1 Node file format description

The first line of the nodefile is a text header. The header is delimited from a subsequent data part by a single newline character (`\n`). The format of the data content can be either text based or binary-coded. The header begins with a keyword, `BINARY` or `ASCII`, which defines the actual file mode. This is followed by a space separated list of parameters: one for the number of datasets expected to be present and two boolean flags (valid values are `0` for "false" or `1` for "true"). The flags control the presence of optional data fields.

The datapart itself has tabular structure. Each row represents a single dataset and is grouped into columns. The minimum number of columns is four — representing the three coordinates (x, y, z) and one id value which is used in order to distinguish different type of atoms with different properties. In text mode, the delimiter for each column is a tabstop and the rows are separated by a newline character from each other. In the case of binary data no delimiters are present.

Two optional columns may occur as declared in the header: one column to associate a unique number with each atom and another one for the potential. If any of the optional columns is not provided, the flag must be set to false. In this case, a unique number will be generated at each node. The potential will be initialized with the default value from the configuration file.

If text mode is chosen as the data format, comments can be inserted everywhere after the header. A comment consists of a single line and is introduced by the hash symbol (`#`) as first character.

An exemplary file looks like this:

```
ASCII␣1447181␣0␣0\n
2.25e-08→ 0.00e+00→ 0.00e+00→ 5\n
    ⋮          ⋮          ⋮       ⋮
1.5e-09→  1.00e-09→ 1.00e-09→ 1\n
```

This file is in text format and does not contain the data column for unique numbers and the potential. — Anolog to the text based structure of the data part is the binary-coded representation. For the single data columns the following data types must be used:

column	data type	size
`x`	*float*	32 bit
`y`	*float*	32 bit
`z`	*float*	32 bit
`id`	*short*	16 bit
`number`	*unsigned int*	32 bit
`potential`	*float*	32 bit

According to the data types listed in the table, the maximum binary size of a row will amount to $4 \cdot \texttt{sizeof}(float) + \texttt{sizeof}(short) + \texttt{sizeof}(unsigned\ int) = 22\,\text{Byte}$. Be cautious: the total number of bytes present between the header and the end-of-file must match the number of expected datasets times the size of a single data row — taking regard to the optional content, respectively!

3.1.2 Configuration file format description

Similar to the node file format, the first line in the configuration file is again a header which consists of the `ASCII` keyword and a separating newline character at finish. Al-

though a binary coded structure of the configuration file exists, this description will focus on the text based version only. For normal usage the plain text format is sufficient. In this context the use is generally preferred because it is human readable, almost self-explaining and therefore easy to edit.

Subsequent to the header, definitions of properties grouped into packages follow. Each data package consists in a consecutive sequence of key value pairs, one per line, which define the properties for a distinct node type of the simulation mesh. The link between the mesh and the configuration entry is established by the value of the `ID` key. Separator for each data package is at least one empty line at the end. This last empty line is also needed subsequent to the last entry at the end of file! Comment lines are allowed and may appear everywhere after the introducing header.

Definition of a default temperature value to be used in the simulation is special. The respective definition consists of one key value pair plus the delimiting empty line. This information is optional. If it is ommitted, a default value of 0.0 K will be preset. However, it is possible to override the respective setting via a command-line parameter at runtime of the simulation.

Part of an exemplary configuration file is listed below. It includes the temperature data package and defines one node type which is labeled with the respective `ID` value. Delimiter for the key value pairs is a sequence of three characters ("␣=␣"), respectively:

```
ASCII\n
...
TEMPERATURE␣=␣30.0\n
\n
...
ID␣=␣0\n
NAME␣=␣Vacuum\n
CHARGE_DENSITY␣=␣0.0\n
DIELECTRICITY␣=␣1.0\n
REMOVABLE␣=␣0\n
NEUMANN_BOUNDARY␣=␣0\n
DIRICHLET_BOUNDARY␣=␣0\n
POTENTIAL␣=␣500.0\n
MASS␣=␣0.0\n
EVAPORATION_CHARGE_STATE␣=␣0\n
EVAPORATION_FIELD_STRENGTH␣=␣0.0\n
EVAPORATION_ACTIVATION_ENERGY␣=␣0.0\n
\n
...
```

Since the distinct properties of the nodes in the simulation mesh essentially control the simulation, it is important to fully understand them. The following list gives a detailed description for each value of the keys:

ID is an integer value in the range from "0" to "31". It is used to establish an unambiguous link between the definition of nodal properties and respective nodes in an associated mesh.

NAME is a character string. It denotes a speaking name for the set of properties defined by an `ID` value. This string may also be used as a substitute for the `ID` itself. For this reason, it is demanded to be unique within a configuration. Whitespace at the beginning and end of the string will be removed on input, intermediate spaces or special characters remain.

CHARGE_DENSITY is a floating point value and defines a local charge density, which is attributed to a nodal cell. The measurement unit is in C/m^3. — The actual local charge is computed by the product of the cell volume and the charge density, respectively.

DIELECTRICITY denotes a certain value of dielectricity attributed to the domain of the associated node. The expected representation is floating point.

REMOVABLE is a flag and may be either 1 ("true") or 0 ("false"). All the nodes which constitute the field emitter must have this property set. Other nodes don't.

NEUMANN_BOUNDARY is another flag which designates the Neumann Boundary condition beeing set. Generally it defines a fixed value for the derivative of the unknown in normal direction to the boundary. In this context the special Neumann Boundary condition is applied. The derivative of the potential — the electric field, respectively — is forced to zero. This means that incident field lines align normal to the boundary plane. The presence of internal Neumann boundaries is not implemented. Therefore this property is only important for nodes at the mesh boundary in contradiction to the Dirichlet type.

DIRICHLET_BOUNDARY is a flag which designates the Dirichlet Boundary condition. Its usage is competitive to the Neumann type. The Dirichlet Boundary condition defines a fixed potential value. Despite boundary nodes, this property is also applicable for nodes which are part of the field emitter in order to set a distinct emitter voltage.

POTENTIAL is a floating point value. At initialization the potential values at each node are preset to the value defined by this property and may change when the mesh is relaxed — except the Dirichlet Boundary condition is applied; then the potential will remain constant at this preset. The measurement unit is V.

MASS describes the mass which is used for computing ion trajectories of evaporated species. The expected data format is floating point. The measurement unit is in atomic mass units (u).

EVAPORATION_CHARGE_STATE is an integer value. It denotes the charge of evaporated ions which is measured in multiples of the unit charge.

EVAPORATION_FIELD_STRENGTH defines the value of the specific evaporation field strength which is taken into account, for determining the next atom in the evap-

oration sequence. The format is floating point. Appropriate measurement unit is V/m.

EVAPORATION_ACTIVATION_ENERGY denotes the energy barrier which must be overcome according to the image hump model for field evaporation. This property is only important if distinguished evaporation properties are computed in the context of a Monte Carlo scheme for field evaporation. Expected fomat is floating point. The measurement unit is eV.

3.2 Assisted mesh generation

This section describes the use of the *MeshGen* support application. This tool is able to create an adaptive mesh covering the mesoscale between a user provided tip structure and a full-scale simulation environment.

MeshGen relies on *TetGen*, an open source quality tetrahedral mesh generator written by Han Si [Si06]. The adaptive mesh can be constructed starting from an input of arbitrary shape. Only minor limitations apply:

- the input tip structure is demanded to be of convex shape,
- all coordinates are expected to be located inside a cylinder with restricted radius and height[2] — the cylinder is placed symmetric to the origin in the half space with $z \geq 0.0\,\mathrm{m}$; especially the input bottom nodes must be aligned in the plane with $z = 0.0\,\mathrm{m}$.

For reasons that are not described here, the output is assembled from a cascading set of submeshes. These are depicted by roman numbers in figure 1a. The patterned domain in the figure indicates the tip part provided by the user. The shape of the first surrounding submesh (I) is adaptive to the input, while in contradiction, the remaining submeshes (II-IV) are simple cylinders with prefixed size. All the submeshes fit into each other nicely and are stacked into each other.

In addition to mesh construction *MeshGen* takes care for correctly assigning standard values for the id information at each node. As depicted in figure 1a the following default values are used at the boundary nodes:

- "*one*" at the top,
- "*two*" at the bottom, and
- "*three*" at the sides.

The respective value for internal (vacuum) nodes is zero.

It might be a problem if the assigned node ids in the generated mesh get in conflict with the ones in the provided input. In order to avoid this situtation, it is recommended to use

[2] The exact dimensions depend on the defined settings. Although these may be changed, the use of the preset defaults is strongly recommended. Defaults provided, the maximum input radius will amount to about 2.66 µm and the respective height to about 90.90 nm.

ids with the numbering beginning at ten within the input mesh. Be aware: *MeshGen* will not modify the input mesh at all! On output, the provided data is just merged with the suitably generated mesh. For this reason the user is especially responsible for correctly assigning any ids to boundary nodes within the provided input! The placement is demonstrated in figure 1b: the encircled domains depict the bottom nodes with $z = 0.0\,\mathrm{m}$. Numbers (1) and (3) denote assignment of the Neumann Boundary Condition and (2) denotes assignment of the Dirichlet type boundary, respectively.

The syntax for running *MeshGen* is: `./meshgen␣input-file␣output-file`. The format for input and output files is of the node file type. If the program is started without any parameters, a listing with possible parameters will be output (enter: `./meshgen`). *MeshGen* asks to write an initialization file at first execution. The initialization file parameters allow detailed control of the program operation. The preset parameter values should be appropiate for almost all standard cases.

If in addtion to the created mesh file a configuration file template should be created, the respective command-line parameter must be added. The calling convention becomes

`./meshgen␣--create-config-template=config-file␣input-file␣output-file`.

in this case. A more detailed description of the comand-line and initialization file parameters is left out at this point.

At last, it should be mentioned that in some circumstances the mesh generation with *MeshGen* does not succeed. This occurs if the invoked *TetGen* subprocess fails and crashs the program. Unfortunately, as the reason for this is somewhere inside the *TetGen* part (at least the program should not crash due to *TetGen*), there is not much what one can do in this situation: If it occurs, look at which stage of program execution the crash happens (optionally with adding the `verbose` command-line parameter). Subsequently, try to modify the respective `*_INNER_CYLINDER_RESOLUTION` and/ or `*_OUTWARD_CYLINDER_RESOLUTION` entries in the initialization file — this effectively changes the number of inital boundary points in the respective sub-mesh and turned out to solve the problem in some cases.

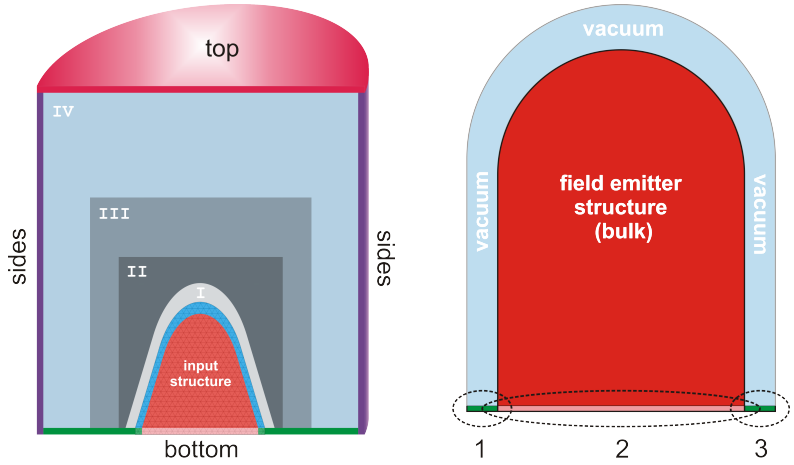
3.3 Running TAPSim

For running TAPSim it is sufficient to enter "`./tapsim`". The printed output will give instructions on how to go on. — This is basically intended to remind the (experienced) user which opportunities exist; so there would be no need to refer to any manual.

However, the basic syntax, in order to do something useful with the program, is:

`./tapsim␣mode␣[file(s)]␣[parameter(s)]`.

The arguments are a keyword, denoted by *mode*, possibly required files and optional command-line parameters, respectively. According to the applied keyword four program modes exist:



(a) Cascading mesh sheme

(b) Typical setup of the input tip structure

Figure 1

- **make-ini**,
- **relaxation**,
- **evaporation** and
- **resumption**.

The chosen mode effectively determines the files and filetypes which need to be supplied. Also, the possible command-line parameters depend on the mode. In general the command-line parameters are useful to force operation different from the default behaviour. Most settings are controlled via the internal/ external initialization mechanism, which will be described in the next subsection describing the "**make-ini**". However, default initialization settings may be ultimately overridden by respective parameters from the command-line when the program is called. Additionally, some settings have effect, whatever mode TAPSim is operating, and are exclusive to the command-line.

In the next subsections the distinct syntax for each mode and the intended usage is explained. But before, the general command-line parameters are described:

`--help`	Prints out the basic usage information of TAPSim as described at the beginning of this section. In combination with a *mode* argument the usage information for that mode is printed.
`--about`	Prints information about the program — especially the license information.

`--no-file-output`	Prohibits writing of any files to the disk at output. This may be useful for testing.
`--write-ascii`	Forces *all* output files to be written in an user-readable text based format.
`--write-binary`	Sets the output mode for *all* files to the binary coded fromat.
`--threads=?`	Determines the number of computing threads for the potential relaxation. The respective argument is an integer value. The default behaviour is to use as many threads as processor cores are present. Besides the computing threads additional threads exist. They process the asyncronous output of data to disk and are not effected by this setting.
`--ini-file=?`	Sets the path to an initialization file, which is the expected argument. This will prevent the use of the internal initialization settings and the reading of the default file from disk — if it would be present.

3.3.1 Make-ini mode

The calling convention is one of these two variants:

```
./tapsim␣make-ini
./tapsim␣make-ini␣filename
```

As its name suggests, the purpose of this mode is to generate an initialization file for TAPSim. In the first version an initialization file with a default name (`tapsim.ini`) is written in the current working directory. The second version allows to direct the output to a user defined path and filename.

In normal operation without any initialization file present, TAPSim relies on internal default settings for its parameters. If you want to revisit or change these parameters you can use the *make-ini* mode.

Then in adddition, if you run TAPSim and an initialization file with the default name is present in the current working directory, the contents of this file is used instead of the internal settings. The standard behaviour is to look for the default file in the current working directory first and then, if it does not exist or is not readable, to fall back on the

internal settings. Otherwise you can also force TAPSim to use a different initialization file with the "`--ini-file=?`" command-line parameter.

The structure of the initialization file is the key-value format. Purpose of the keys is described in the following. More details of the mentioned output file formats will be topic of section 4. Subsequently only the neccessary about the file formats is pointed out:

RESULTS_FILENAME is a character sequence. This sets the default file prefix for output of the main simulation results. The recorded data is analog to data you would obtain from a real atom probe measurement. The full filename is constructed from the given prefix and the running index of the first evaporation event which is written to it; e.g. "evaporation_data.000001" contains all events beginning with the first one, "evaporation_data.0000500" all the events beginning with the 500th, and so on. The maximum number of events is determined by the chunk size parameter (see below). If this filename is empty, any output is prohibited.

RESULTS_BINARY_OUTPUT is a boolean flag which determines the file format of the results file ("1" $\hat{=}$ binary coded output, "0" $\hat{=}$ text output).

RESULTS_CHUNK_SIZE is an integer. The chunk size determines the maximum number of evaporation events recorded per single results file. If the stated size is zero, output is written to a single file only.

TRAJECTORY_FILENAME is a character sequence. Analog to the results file, this entry holds the file prefix for the output of the full ion trajectory data. If it is an empty value, the respective output is prohibited.

TRAJECTORY_BINARY_OUTPUT is a boolean flag and determines the file format for trajectory data.

TRAJECTORY_CHUNK_SIZE is an integer. Sets the threshold for the maximum number of ion trajectories per file. If the value is zero, output will be constrained to a single file.

GRID_FILENAME is also a character sequence. A grid file contains a snapshot of the complete computing mesh including the potential and field values at each node. This entry determines the prefix for the grid output files. An empty entry prohibits any output.

GRID_BINARY_OUTPUT is a boolean flag, which sets the respective output format. Because of the possibly huge amount of data, binary output should be preffered at this point.

GRID_INTERVAL is an integer. It determines the period at which the grid snapshots are written to disk, e.g. every 100 or every 1000 evaporation events.

SURFACE_FILENAME is a character sequence. It holds the file prefix for surface data, which is the actual shape of the field emitter. In order to prohibit this kind of output set an empty string here.

SURFACE_BINARY_OUTPUT is boolean flag for the respective surface output format.

SURFACE_INTERVAL is an integer. Analog to the grid file output this is the snaphot interval for the shape of the field emitter.

GEOMETRY_FILENAME is a character sequence which denotes a full filename for the geometry data. Owing to the fact that the basic mesh geometry does not change at all during a simulation run, the geometry is recorded only once at the beginning of a simulation and is then saved to disk. This kind of data is needed in combination with the surface and grid data to reconstruct the full state afterwards.

GEOMETRY_BINARY_OUTPUT is a boolean flag. It determines the output format for geometry data.

GEOMETRY_CONTENTS is a hexadecimal coded bit sequence. Output of the full available geometry is reached by a value of '0xff'.

DUMP_FILENAME is a character sequence. Periodically the complete state of simulation is written to disk and backed up in a so called dump file. The respective file name is set via this entry. Output of a dump file is prohibited in the case of an empty string.

DUMP_INTERVAL is an integer and denotes the interval at which the state information is written to disk. Any previous file contents will get lost.

OUTPUT_DELAY_TIME is an integer. The value denotes the time in seconds at which buffered output information is written to disk. In fact this is the wake-up time for the internal disk-writer-threads which will do the work. Intermediately, data will be collected in memory. If the time is set to zero all data will be written immediately and without any buffering.

METHOD_EVAPORATION_PROBABILITY is an integer value. It determines the method which is used to calculate the field evaporation probability for each surface atom of the field emitter ($0 \mathrel{\hat{=}}$ "linear approach", $1 \mathrel{\hat{=}}$ "exponential (Boltzmann) approach").

METHOD_EVAPORATION_CHOICE is an integer value. This denotes the method which is applied to select an atom for field evaporation from the emitter surface ($0 \mathrel{\hat{=}}$ "maximum probability", $1 \mathrel{\hat{=}}$ "probability weighted randomization").

VACUUM_CELL_IDENTIFIER is a character sequence. This string depicts the vacuum nodes within the mesh (the unique name which is given in the respective configuration file). This information is needed in order to obtain the nodes which are part of the emitter surface. The string defaults to "Vacuum".

TRAJECTORY_INTEGRATOR_TYPE is an integer value. It sets the type of integration method which is used to compute the ion trajectory ($0 \mathrel{\hat{=}} \mathcal{O}(1)$-integrator, $1 \mathrel{\hat{=}} \mathcal{O}(5)$-integrator).

TRAJECTORY_STEPPER_TYPE is an integer value. This value controls the conditions under which a proposed integration step of the trajectory becomes successful

($0 \hat{=}$ every step succeeds, $1 \hat{=}$ restriction by preset numerical error threshold, $2 \hat{=}$ step is restricted by local mesh geometry, $3 \hat{=}$ restriction by local mesh geometry an predefined error threshold applies).

TRAJECTORY_INITIAL_TIME_STEP is a floating point. It denotes the initial value of time discretization for integration (e.g. 1×10^{-14} s).

TRAJECTORY_MINIMUM_TIME_STEP is a floating point and sets a lower threshold for time discretization (the value defaults to 1×10^{-16} s).

TRAJECTORY_ERROR_THRESHOLD is a floating point. This depicts the numerical error threshold for computed trajectories.

TRAJECTORY_NON_RANDOM_START_POSITION is a boolean flag. If set to "false", a blurred starting position in order to mimic the Brownian motion of the emitter atoms is enabled. Otherwise, if set to "true", the lattice position is used.

TRAJECTORY_NO_INITIAL_VELOCITY is a boolean flag. An initial random velocity for evaporated atoms is applied if this key equals "false". In the opposite case the velocity defaults to zero.

VOLTAGE_QUEUE_SIZE is an integer value. For each event a representative tip voltage is computed taking into account the preset evaporation field, the local potential, and previously computed voltage values of the preceeding events for averaging. The queue size denotes the number of events over which is actually averaged.

LOCAL_RELAXATION_ERROR_THRESHOLD is a floating point value. It defines the error threshold which applies to the maximum computed potential changes in local relaxation mode.

LOCAL_RELAXATION_SHELL_SIZE is an integer value. It determines the order of next neighbour nodes which are taken into account for local relaxation, e.g. the default is three, which means that the environment is constrained by the third order next neighbours.

LOCAL_RELAXATION_CYCLE_SIZE is an integer value. It sets the number of subsequent local relaxation cycles which are processed without testing whether the error threshold has been reached.

LOCAL_RELAXATION_QUEUE_SIZE is an integer value. If the preset error threshold for the local relaxation of potentials cannot be reached due to the presence of numerical inaccurracies, relaxation will stop anyway. The number of preceeding cycles which are taken into account for detection of this situation, is defined by this setting.

GLOBAL_RELAXATION_STEPS is an integer value. This sets the number of global relaxation cycles subsequently processed to any local relaxation. The default is one.

REFRESH_RELAXATION_INTERVAL is an integer value. This value defines the number of evaporation events after which an error controlled relaxation of the potentials

will be conducted. In the case this value is set to zero any error controlled relaxation is skipped. (The described procedure takes part in addition to the global relaxation subsequent to any localized one).

REFRESH_RELAXATION_ERROR_THRESHOLD is a floating point value. It denotes the error threshold which is applied to the global potential relaxation step (the initial one before the evaporation sequence starts and any subsequent intermediate steps).

REFRESH_RELAXATION_CYCLE_SIZE is an integer value used for global relaxation processing. The meaning is analog to the description given for the "LOCAL_RELAXATION_CYCLE_S entry.

REFRESH_RELAXATION_QUEUE_SIZE is an integer value. It applies to the global relaxation of the potentials. Again the meaning is analog to the given at the "LOCAL_RELAXATION_QUEUE_SIZE" entry.

3.3.2 Evaporation mode

The evaporation mode is the main operation mode of TAPSim. The calling syntax is:

```
./tapsim evaporation iconfig inode [results]
```

or

```
./tapsim evaporation --input-dump=? iconfig [results]
```

In the first form a configuration filename followed by a node filename and optionally a results filename for output is expected. Calling TAPSim this way implies the following list of tasks which will get processed:

- At beginning, the input files for the configuration and node data are read.
- Then the Delaunay tetrahedralization from the nodal coordinates is obtained and the computing mesh is constructed.
- Subsequently, needed coupling factors for the iterative finding of the potential solution are computed. This process is called synchronizing due to the linking of geometric properties with electrostatic ones.
- The initial potential distribution is obtained and,
- finally, the consecutive evaporation of atoms from the field emitter is simulated. In the cyclic procedure the potential is continously recomputated and the respective trajectory and event data are recorded for each event. Within preset intervals, additional data is saved in order to track the evolution of the emitter shape and the state of the computation grid. A dump file is written to disk repeatedly, which will allow resumption of the simulation if the program execution is interrupted for some reason.

The second form of the prescribed syntax is used in the case a dump file with an relaxed mesh (see section 3.3.4) is initially present. The provided information from the dump is

then combined with the input data from the configuration file and processing starts with the syncronization task. The optional *results* argument in both syntaxes is a filename prefix for the output of results data. If it is present, the default setting by the initialization data will be overridden.

Finally, one remark to the process of finding the potential: computation can be executed globally including all nodes in the mesh or it may be restricted to a local environment of some order of next neighbour shells in vicinity to an arbitrary center node. The second variant is applied in the case the potential needs to be recomputed subsequent to an evaporation event. Whatever variant is conducted, controlling parameters remain basically the same:

- The potential relaxation is stopped if a preset threshold value has been reached. Therefore the maximum potential difference at any processed node before and after one relaxation step is obtained and compared with this user-defined preset.
- In order to avoid the overhead to compute the actual value for comparison in each relaxation cycle, this value is determined periodically. The respective number of pure relaxation steps without comparison is controlled by the cycle-size parameter type.
- A third parameter affects the detection of a deadlock-situation when the preset threshold is not reachable for some reason: The queue-size controls the length of a fifo-queue. This queue is filled with the reference values which have been computed for threshold comparison. The deadlock-sitation is detected if the slope of the linefit for the values in the queue becomes zero. The relaxation is stopped in this case — however the threshold has been reached or not.

In evaporation mode the following command-line parameters may be invoked.

`--skip-relaxation`	Skips the initial relaxation step for the potential.
`--init-potentials` `--reset-potentials`	Forces initialization of potentials prior to any relaxation. Vacuum nodes are set to a random potential, while all other nodes become initialized according to the respective configuration data.

`--threshold=?`
`--cycle-size=?`
`--queue-size=?`
	Set the parameters for global relaxation.

`--no-trajectories`	Disable the output of trajectory data.
`--write-trajectories=?`	Set the filename prefix for trajectory data.
`--no-grids`	Disable the output of grid data.
`--write-grids=?`	Set the filename prefix for grid data.
`--no-surfaces`	Disable the output of surface data.

`--write-surfaces=?`	Set the filename prefix for surface data.
`--no-chunks`	Disable splitting of the output into several files (applies to results, trajectory data, respectively).
`--chunk-size=?`	Set the maximum number of event data per results or trajectory file at output.
`--snapshot-interval=?`	Set the period of evaporation events for writing the grid and surface data.
`--no-dump`	Disable writing of a dump file.
`--write-dump=?`	Set the filename for the dump file.
`--dump-interval=?`	Set the period of evaporation events after which a dump file is written.
`--vacuum-identifier=?`	Set the unique name for identifying vacuum nodes.
`--trajectory-error-threshold=?`	Set the numerical error threshold which is applied for the trajectory computation.
`--shrink-limit=?`	Set the maximum shrink size of the field emitter after which the simulation is stopped. Zero is equivalent to infinite size, e.g. complete removal of all atoms from the emitter.
`--event-limit=?`	Set the maximum number of atoms which should be removed by the simulation. Zero is equivalent to infinite number.
`--voltage-queue-size=?`	Set the length of the voltage queue which is used for smoothing of calculated evaporation voltages.

`--evap-threshold=?`
`--evap-cycle-size=?`
`--evap-shell-size=?`
`--evap-queue-size=?`
	Override the parameters which are used for the local relaxation step which applied after each removal of an atom from the field emitter.

`--evap-global-cycles=?`	Set the number of global relaxation cycles after each evaporation step.

`--refresh-interval=?`
`--refresh-threshold=?`
`--refresh-cycle-size=?`
`--refresh-queue-size=?`
	Set the parameters which are used for intermediate relaxation of the potential during the evaporation cycle. The refresh interval defines the number of events after which refreshing takes place. If it is set to zero, the procedure is skipped at all.

3.3.3 Resumption mode

If you have got a dump file which was previously generated by TAPSim the resumption mode allows to resume the simulation from the information in this file. The syntax is:

```
./tapsim␣resumption␣idump␣[--last-event=?]␣[results]
```

The first parameter is the input dump file. Optional parameters are an initialization value for the event index, denoted by *last-event* and a user specific filename for the results data, respectively. If the event index parameter is omitted, the value from within the dump information will be used by default. As the simulation is resumed from a dump, all needed information is directly available and the action immediately starts with the processing of the evaporation cycle.

Additional parameters as they have been explained in the preceeding for the evaporation mode may be supplied. Except the ones which force the initialization/ reset of the potential since this would make no sense in this mode.

3.3.4 Relaxation mode

This mode is basically useful to only computing the potential distribution in a mesh. The syntax which applies to the relaxation mode is as follows:

```
./tapsim␣relaxation␣--read-dump=?␣oconfig␣onode
```

or

```
./tapsim␣relaxation␣iconfig␣inode␣[onode]
```

The parameters with *config/ node* denote a configuration and a node file at input or output. Argument for the `--read-dump` parameter is a dump filename, respectively.

If TAPSim is called in the way it is stated in the first line, a dump file will be read, the included mesh with the potential values will be relaxed and on output a configuration file and a node file will be written.

The version presented in the second line is very similar. But instead of a dump file a configuration and a node file is read. After the relaxation the output is eiter saved in the input node file, which gets overridden or it is written to an optionally provided output node file.

In addition the following command-line parameters are available in this mode:

`--init-potentials` `--reset-potentials`	Forces initialization of potentials prior to any relaxation. Vacuum nodes are set to a random potential, while all other nodes become initialized according to the respective configuration data.

`--threshold=?`
`--cycle-size=?`
`--queue-size=?`
	Overwrite the default parameters for global relaxation.

`--write-dump=?`	Forces output of a dump file with the given name after relaxation.

3.4 Example: Simulated evaporation of a tungsten field emitter

In this section a step by step instruction for the simulated evaporation of a bcc ⟨110⟩ specimen is given. The sample shape is a cylinder with half-spherical apex. The diameter amounts ≈ 30 nm and the height to ≈ 60 nm. A node file containing the respective geometry is provided in the "`./examples`" subdirectory of the TAPSim package. The file is called "bcc_110_5.00e-10.bin". In the following it is assumed that the binaries have already been compiled as described in section 2 and reside in the "`./bin`" directory below the root of the package.

The first step is the setup of the full-scale simulation mesh. For this *MeshGen* is used. Switch to the directory with the provided sample and enter:

```
../bin/meshgen␣bcc_110_5.00e-10.bin␣sampleMesh.bin␣
--create-config-template=sampleMesh.cfg
```

If no initialization file exists in the current directory, *MeshGen* asks to create a default one. Proceed writing the file and continue program execution. The mesh generation process takes place. It completes within seven stages and will probably consume a few minutes. Maybe you will get a message that some points have been removed from the *generated* mesh due to limitations of the floating point arithmetic. You can ignore this warning and go on. — At finish, the supplied *create-config-template* parameter lets *MeshGen* write an extra configuration file, named `sampleMesh.cfg` in this case.

This file is now to be edited in the next step. Use your preferred text editor (e.g. *vi*) and change the subsequent entries in the description for which the `ID` is set equal to 10 (this information should have been placed somewhere near at the end of the file):

- `NAME␣=␣Tungsten`
- `MASS␣=␣183.85`
- `EVAPORATION_CHARGE_STATE␣=␣3`
- `EVAPORATION_FIELD_STRENGTH␣=␣57.1e-9`

Afterwards TAPSim is started, enter:

```
../bin/tapsim evaporation sampleMesh.cfg sampleMesh.bin
```

The first steps of the simulation will need a lot of time before the evaporation of single atomes is really computed. The generated output files are explained in the next section and must be analysed with custom made tools. — However, if not interrupted, the program will continue execution until all atoms have been removed from the emitter. Have a lot of fun!

4 Output data and file formats

As it is the case for the node file and configuration file formats, output of TAPSim is basically recorded in a plain text format or binary coded. The encoding is marked by a respective keyword at the beginning of the files, maybe subsequent to some lines which are introduced with the hash symbol.

The data, which is in direct relationship to a single evaporation event, is written to the results type file and a trajectory file. The results file is at most analog to an atom probe measurement file. Both files are streaming formats. They can hold an arbitrary number of event data within one file or split it over multiple files of that types.

Contrary, information about the computing mesh as a whole is accessible by grid type data files. In the case you are only interested in the information in direct vicinity of the tip surface, you may refer to the surface data type file. Because these file formats decisively rely on the distinct mesh geometry, which will not change at all during a simulation run, the detailed geometric information is saved in a separate but common geometry type file. This allows to save a huge amount of disk space and in addition processing time for the destillation of the geometric data (typical size of the geometry data amounts to some gigabytes, while the grid data needs only about some tenth megabytes or a few megabytes for the surfce information of space, respectively).

4.1 Results data

As already mentioned, results data is the main output format of TAPSim. It contains information analog to the measurement files in atom probe and additional data which is only accessible due to the simulation approach. The basic format is a tabular structure with the data of each evaporation event in one row. The complete dataset looks like this; the data types stated in the second column are valid for the binary-coded format only:

index	*int*	running index of the evaporation event
id	*int*	identifier value of the atom type
number	*unsigned int*	unique number of the original atom node

voltage	*float*	computed measurement voltage in volts (this value is computed by rescaling the local potential with the evaporation field strength of the atoms at the field emitter; do not get confused: it is *not* the potential in the computing mesh!)
startX **startY** **startZ**	*float*	starting position of the ion trajectory/ position at the emitter surface in meters
stopX **stopY** **stopZ**	*float*	stopping position of the computed ion trajectory in meters.
tof	*float*	time-of-flight in seconds (this value is computed according to the measurement voltage)
probability	*float*	the evaporation probability for the event (evaporation probabilities of all surface atoms of the field emitter are always normalized)
potentialBefore	*float*	the potential at the atom position prior evaporation in volts
fieldBeforeX **fieldBeforeY** **fieldBeforeZ**	*float*	electric field vector at the atom position prior evaporation in volts per meter
potentialAfter	*float*	potential at the atom position subsequent to its removal and local relaxation in volts
fieldAfterX **fieldAfterY** **fieldAfterZ**	*float*	electric field vector at the atom position subsequent to the evaporation and local relaxation in volts per meter
normalX **normalY** **normalZ**	*float*	unit normal vector of the local emitter surface at the atom position (prior removal)
apexX **apexY** **apexZ**	*float*	coordinates of the emitter apex in meters (The apex atom is defined by the position at the emitter surface with the maximum z coordinate.)

4.2 Trajectory data

The trajectory data is used to store the complete information accompanied with the computation of ion trajectories. The data in the file is grouped into buckets. Each bucket contains the information attributed to a single trajectory plus additional content which is embedded in comment lines before and subsequent to the main data part. Basically, a trajectory is described as a sequence of points in the $(t, \vec{r}, \vec{v})$ phase space; t denoting time, $\vec{r}$ the location and $\vec{v}$ the velocity, respectively. In addition to this neccessary information an index to the associated tetrahedron of the Delaunay mesh, in which the respective position is located, is also recorded.

Part of a trajectory file presenting the information within a single bucket is listed below:

```
# index = 14\n
# unique number = 0\n
# time scale = 3.5e-01\n
# integrator status = 5 => SYSTEM_LIMIT_PASSED\n
# charge [C] / mass [kg] = 1.6e-19 / 8.3e-26\n
ASCII 61\n
0.0e+00→ -8.5e-09→ 0.0+00→  2.1e-08→ 0.0e+00→ 0.0e+00→ 0.0e+00→ 562\n
⋮        ⋮         ⋮         ⋮         ⋮         ⋮         ⋮         ⋮
1.6e-12→ -3.6e-08→ 0.0e+00→ 6.8e-08→ -2.8e+04→ 0.0e+00→ 4.5e+04→ 908\n
1.9e-12→ -4.4e-08→ 0.0e+00→ 8.1e-08→ -2.8e+04→ 0.0e+00→ 4.5e+04→ -1\n
# final error estimate (position) = (1.2e-14/0.0e+00/1.8e-14)\n
# final error estimate (velocity) = (4.3e-02/0.0e+00/6.5e-02)\n
\n
```

The keyword introducing the main data part is followed by an integer. The value denotes the number of expected rows. The meaning of additional information in the comment lines is as follows:

Index is the index of the evaporation event. This entry provides a link to results data file.

Unique number is the number which was assigned to nodal cell at the origin of the ion.

Time scale denotes the scaling factor which was used to compute realistic time-of-flight values if a certain measurement voltage was taken into account.

Integrator status indicates the status at which integration was stopped.

Charge/ mass denotes the respective ion charge and its mass.

Final error estimates denote the total numerical error estimates from the integration, for the position and velocity, respectively.

In the binary coded version all data types are floating points, except the index for the tetrahedra which is an integer.

4.3 Geometry data

The geometry data is quite complex. For this reason only a rough overview will be presented here and for detailed information a reference to the TAPSim source code is given.[3]

The first line of the header is the keyword for format encoding. Then a series of different keywords follows. Each of these keywords mark a different kind of data with a distinct structure of the subsequent information:

- The first data to be present is introduced with `NODES`. This is for the coordinates of the vertices in the mesh and an additional boundary marker. The boundary marker indicates if a vertex is on the boundary or not, respectively.
- Then the information about the Delaunay tetrahedralization follows, marked with the `TETRAHEDRA` keyword. For each tetrahedron the indices of the four constituting vertices are written and in addition, another four indices its neighbouring tetrahedra.
- Subsequent information about the `VORONOI_CELLS` follows. This basically consists in the definition of `VORONOI_FACETS`. The union of all facets incident to a common node forms the closed domain of the respective Vornoi cell. Each Voronoi facet is a convex polygon. The list with the coordinates of all appearing `VORONOI_VERTICES` completes the geometry data.

The coordinates of a Voronoi vertex are computed via the circumcenters of the tetrahedra. Here, use is made of the geometric duality of the Delaunay tetrahadralization and the partition of space into Voronoi cells. Each vertex of a tetrahedron is linked to a distinct Voronoi cell. Vice versa each tetrahedron is linked to a a distinct Voronoi vertex.

4.4 Grid data

The grid data format is one big table. For each node of the mesh

- the id value (*short*, 16 bit),
- unique number (*unsigned int*),
- local charge (*float*),
- potential (*float*) and
- the electric field vector (three *floats*)

are recorded at certain a state of the simulation. Like a snaphot.

First line in the file is the header with the keyword for signaling the encoding and the number of nodes, equal to the number of datarows, of the respective mesh. For coordinate information of the nodes et cetera it is referrred to the geometry data file

[3]The respective source file is `file_io.cpp` and the function `File_Io::writeGeometry()` therein.

which is provided in there. If binary encoding is used, the respective datatypes are as stated in parenthesis above.

4.5 Surface data

The contents of the surface data file are comparable to a grid file. But in contradiction, the surface file only contains a subset of the data provided in the grid file: only information which depends on the surface of the field emitter (the single atoms of the field emitter, and their neighbouring nodes, e.g. local vacuum nodes) are recorded. As it is the case for the grid format, the according data is basically organized in tabular form. Subsequent to the header line with the identifier for the data format, two tabular data parts follow.

The first datapart is introduced by a single line with the `CELLS` keyword and a number value. It contains the neccessary data of the surface cells of the field emitter. Each dataset in a row describes a single surface site. It consists of four columns (data types for binary coding are in parenthesis):

- a node index (*int*),
- the type/ id (*short*, 16 bit),
- the unique number (*unsigned int*) and
- the computed evaporation probability (*float*).

In addition a fifth column with a variable number of indices to neighbouring nodes in the mesh is appended. In order to be able to reconstruct the surface shape and draw the respective Voronoi cell this additional information of the neighbouring nodes at each surface site is needed. The first value in the fifth column is an integer with the number of neighbours, followed by the index values for the neighbours, respectively (all *int*s). Since some surface nodes are neighbours to each other, part of this information is redundant. Nevertheless it is completely recorded. Also be aware that some of these neighbour nodes may not be part of the field emitter. Instead they are "vacuum" nodes.

The second data part is introduced by a separate line with the `NODES` keyword. After the keyword the number of the expected node data that follows is printed. There is one row for each occurring node index in the `CELLS` part above — the indices of neighbouring nodes included. The first column in this data part contains

- the respective node index (*int*),
- the local potential (*float*),
- the electric field vector (three *floats*) and
- the unit normal vector (three *floats*)

at the local emitter surface. In the case the respective node is not a surface node, either outside the surface in the vacuum or within the bulk of the emitter, the surface normal vector is zero.

References

[Obe14] Christian Oberdorfer. "Numeric Simulation of Atom Probe Tomography". PhD thesis. Institute of Materials Physics, University of Münster, 2014. URL: nbn-resolving.de/urn:nbn:de:hbz:6-72369452077.

[OES13] Christian Oberdorfer, Sebastian Manuel Eich, and Guido Schmitz. "A full-scale simulation approach for atom probe tomography". In: *Ultramicroscopy* 128 (2013), pp. 55–67. DOI: 10.1016/j.ultramic.2013.01.005. URL: www.sciencedirect.com/science/article/pii/S0304399113000144.

[She97] Jonathan Richard Shewchuck. "Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates". In: *Discrete & Computational Geometry* 18.3 (Oct. 1997), pp. 305–363. URL: www.cs.cmu.edu/~quake/robust.html.

[Si06] Hang Si. *TetGen — A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator*. Version 1.4. Weierstrass Institute for Applied Analysis and Stochastics (WIAS). Mohrenstr. 39, 10117 Berlin, Germany, Jan. 2006. URL: www.wias-berlin.de/software/tetgen.