PICLas Documentation

Institute for Aerodynamics and Gas Dynamics (University of Stuttgart)
Institute for Space Systems (University of Stuttgart)
boltzplatz - numerical plasma dynamics GmbH

Apr 22, 2024

USER GUIDE

PICLas is a three-dimensional simulation framework for Particle-in-Cell, Direct Simulation Monte Carlo and other particle methods that can be coupled for the simulation of collisional plasma flows. It features a high-order discontinuous Galerkin (DG) simulation module for the solution of the time-dependent Maxwell equations on unstructured hexahedral elements in three space dimensions. The code was specifically designed for very high order accurate simulations on massively parallel systems. It is licensed under GPLv3, written in Fortran and parallelized with MPI.

CHAPTER

ONE

USER GUIDE

PICLas comes with a vast variety of models and methods. Originally being centered around Particle-in-Cell (PIC) and Direct Simulation Monte Carlo (DSMC) methods, PICLas has been extended to other particle-based methods, namely Bhatnagar-Gross-Krook (BGK) and Fokker-Planck (FP) models. Each of these models, some of which can be combined, offer distinctive features such as

- Coupled Particle-in-Cell with Direct Simulation Monte Carlo methods
- Particle-based Bhatnagar-Gross-Krook (Ellipsoidal Statistical, Shakov, Unified) and Fokker-Planck (Cubic, Ellipsoidal Statistical) models for continuum gas flows
- Arbitrary order nodal polynomial tensor product basis using Gauss or Gauss Lobatto collocation points for electrostatic and electromagnetic solvers
- Matching high order curved mesh generation from external mesh formats (CGNS, GMSH) or simple analytic blocks via the open source preprocessor HOPR [1]
- Non-conforming interfaces [2] based on the mortar approach [3, 4] (electromagnetic solver)
- Non-reflecting boundary conditions via CFS-PMLs [5] (electromagnetic solver)
- Automatic domain decomposition for parallel simulations based on a space filling curve
- High order low-storage explicit Runge-Kutta time integration [6]
- I/O using the HDF5 library optimized for massively parallel jobs

1.1 Installation

The following chapter describes the installation procedure on a Linux machine requiring root access. This includes the installation of required prerequisites, setting up MPI and HDF5. Please note that high-performance clusters usually have a module environment, where you have to load the appropriate modules instead of compiling them yourself. The module configuration for some of the clusters used by the research group are given in Chapter *Cluster Guidelines*. In that case, you can jump directly to the description of the download and installation procedure of PICLas in Section *Obtaining the source*.

1.1.1 AppImage executable download

PICLas and its tools can be installed on a Linux machine without the need of compiling the source code. Currently, PICLas executables are only available as *AppImage* for Linux. The only requirements are that GNU C Library (glibc) and OpenMPI are pre-installed on the target system and available when running the AppImage executables. Static libraries for OpenMPI are not distributed within the AppImage package because of the system-dependent optimizations (e.g. specific InfiniBand settings) and the AppImage of piclas is built with version 4.1.0. Additional external libraries and versions that are used for compiling are gcc (GCC) 8.3.1 20190311 (Red Hat 8.3.1-3), HDF5 1.12.2 and PETSc 3.18.4. Other operating systems, such as Windows or MacOS might be supported in the future.

Download the pre-compiled (on Centos7) executables from the PICLas release tag assets. Note that versions prior to v3.0.0 are not supported for AppImage download. Unzip the files, switch into the directory an then and check their MD5 hashes by running

```
md5sum -c md5sum.txt
```

which should produce output looking like

```
piclasDSMC: OK
piclasLeapfrogHDG: OK
piclas2vtk: OK
superB: OK
```

indicating that everything is fine. After downloading the binary files, it has to be checked that all files are executable and if not simply run

```
chmod +x piclas*
```

for all files beginning with piclas (add the other files to the list if required) before they can be used. If problems occur when executing the AppImage, check the troubleshooting section for possible fixes.

1.1.2 Prerequisites

PICLas has been used on various Linux distributions in the past. This includes Ubuntu 16.04 LTS and 18.04 LTS, 20.04 LTS 20.10 and 21.04, OpenSUSE 42.1 and CentOS 7. For a list of tested library version combinations, see Section *Required Libraries*.

The suggested packages in this section can be replaced by self compiled versions. The required packages for the Ubuntu Linux distributions are listed in Table 1.1 and for CentOS Linux in Table 1.2. Under Ubuntu, they can be obtained using the apt environment:

```
sudo apt-get install git cmake
```

Table 1.1: Debian/Ubuntu packages. x: required, o: optional, -: not available

Package	Ubuntu 16.04/18.04	Ubuntu 20.04
git	x	x
cmake	x	x
cmake-curses-gui	o	o
liblapack3	x	x
liblapack-dev	x	x
gfortran	x	x
g++	x	x
mpi-default-dev	x	x
zlib1g-dev	x	x
exuberant-ctags	o	o
ninja	o	o
libmpfr-dev		x (GCC >= 9.3.0)
libmpc-dev		x (GCC >= 9.3.0)

and under CentOS via

```
sudo yum install git
```

For extra packages install EPEL and SCL

```
sudo yum install epel-release centos-release-scl
```

Table 1.2: Centos packages. x: required, o: optional, -: not available

Package	CentOS 7
git	x
cmake	x
cmake3	x
libtool	x
ncurses-devel	x
lapack-devel	x
openblas-devel	x
devtoolset-9	x
gcc	x
gcc-c++	x
zlib1g	x
zlib1g-devel	o
exuberant-ctags	o
numactl-devel	x
rdma-core-devel	o
binutils	x
tar	x

On some systems it may be necessary to increase the size of the stack (part of the memory used to store information about active subroutines) in order to execute **PICLas** correctly. This is done using the command

```
ulimit -s unlimited
```

from the command line. For convenience, you can add this line to your `.bashrc`.

1.1.3 Required Libraries

The following list contains the **recommended library combinations** for the Intel and GNU compiler in combination with HDF5, OpenMPI, CMake etc.

PICLas Version	Compiler	HDF5	MPI	CMake
2.8.0	gcc12.2.0	1.12.2	openmpi-4.1.4	3.24.2
2.3.0 - 2.7.0	gcc11.2.0	1.12.1	openmpi-4.1.1	3.21.3
2.0.0	intel19.1	1.10	impi2019	3.17
2.0.0 - 2.2.2	intel19.1	1.10	impi2019	3.17

and the **minimum requirements**

PICLas Version	Compiler	HDF5	MPI	CMake
2.3.0 - 2.8.0	gnu9.2.0	1.10.6	openmpi-3.1.6	3.17
2.0.0 - 2.2.2	intel18.1	1.10	impi2018	3.17

A full list of all previously tested combinations is found in Chapter *Appendix*. Alternative combinations might work as well, however, have not been tested.

If you are setting-up a fresh system for the simulation with PICLas, we recommend using a Module environment, which can be setup with the provided shell scripts in `piclas/tools/Setup_ModuleEnv`. A description is available here: `piclas/tools/Setup_ModuleEnv/README.md`. This allows you to install and switch between different compiler, MPI and HDF5 versions.

Installing GCC

You can install a specific version of the GCC compiler from scratch, if for example the compiler provided by the repositories of your operating systems is too old. If you already have a compiler installed, make sure the environment variables are set correctly as shown at the end of this section. For this purpose, download the GCC source code directly using a mirror website (detailed information and different mirrors can be found here: GCC mirror sites)

```
wget "ftp://ftp.fu-berlin.de/unix/languages/gcc/releases/gcc-11.2.0/gcc-11.2.0.tar.gz"
```

After a successful download, make sure to unpack the archive

```
tar -xzf gcc-11.2.0.tar.gz
```

Now you can enter the folder, create a build folder within and enter it

```
cd gcc-11.2.0 && mkdir build && cd build
```

Here, you can configure the installation and create the Makefile. Make sure to adapt your installation path (`--prefix=/home/user/gcc/11.2.0`) before continuing with the following command

```
../configure -v \
--prefix=/home/user/gcc/11.2.0 \
--enable-languages=c,c++,objc,obj-c++,fortran \
--enable-shared \
--disable-multilib \
--disable-bootstrap \
--enable-checking=release \
--with-sysroot=/ \
--with-system-zlib
```

After the Makefile has been generated, you can compile the compiler in parallel, e.g. with 4 cores

```
make -j4 2>&1 | tee make.out
```

and finally install it by

```
make install 2>&1 | tee install.out
```

If you encounter any difficulties, you can submit an issue on GitHub attaching the `make.out` and/or `install.out` files, where the console output is stored. To make sure that the installed compiler is also utilized by CMake, you have to set the environment variables, again making sure to use your installation folder and the correct version

```
export CC = /home/user/gcc/11.2.0/bin/gcc
export CXX = /home/user/gcc/11.2.0/bin/g++
export FC = /home/user/gcc/11.2.0/bin/gfortran
export PATH="/home/user/gcc/11.2.0/include/c++/11.2.0:$PATH"
export PATH="/home/user/gcc/11.2.0/bin:$PATH"
export LD_LIBRARY_PATH="/home/user/gcc/11.2.0/lib64:$LD_LIBRARY_PATH"
```

For convenience, you can add these lines to your `.bashrc`.

Installing OpenMPI

You can install a specific version of OpenMPI from scratch, using the same compiler that will be used for the code installation. If you already have OpenMPI installed, make sure the environment variables are set correctly as shown at the end of this section. First, download the OpenMPI source code either from the website https://www.open-mpi.org/ or directly using the following command:

```
wget "https://download.open-mpi.org/release/open-mpi/v4.1/openmpi-4.1.1.tar.gz"
```

Unpack it, enter the folder, create a build folder and enter it (procedure analogous to the GCC installation). For the configuration, utilize the following command, which specifies the compiler to use and the installation directory with `--prefix=/home/user/openmpi/4.1.1`

```
../configure --prefix=/home/user/openmpi/4.1.1 CC=$(which gcc) CXX=$(which g++) FC=
↪$(which gfortran)
```

After the Makefile has been generated, you can compile OpenMPI in parallel, e.g. with 4 cores

```
make -j4 2>&1 | tee make.out
```

and finally install it by

```
make install 2>&1 | tee install.out
```

If you encounter any difficulties, you can submit an issue on GitHub attaching the `make.out` and/or `install.out` files, where the console output is stored. To make sure that the installed OpenMPI is also utilized by CMake, you have to set the environment variables, again making sure to use your installation folder and the correct version

```
export MPI_DIR=/home/user/openmpi/4.1.1
export PATH="/home/user/openmpi/4.1.1/bin:$PATH"
export LD_LIBRARY_PATH="/home/user/openmpi/4.1.1/lib:$LD_LIBRARY_PATH"
```

For convenience, you can add these lines to your `.bashrc`.

Installing HDF5

An available installation of HDF5 can be utilized with PICLas. This requires properly setup environment variables and the compilation of HDF5 during the PICLas compilation has to be turned off (`LIBS_BUILD_HDF5 = OFF`, as per default). If this option is enabled, HDF5 will be downloaded and compiled. However, this means that every time a clean compilation of PICLas is performed, HDF5 will be recompiled. It is preferred to either install HDF5 on your system locally or utilize the packages provided on your cluster.

You can install a specific version of HDF5 from scratch, using the same compiler and MPI configuration that will be used for the code installation. If you already have HDF5 installed, make sure the environment variables are set correctly as shown at the end of this section. First, download HDF5 from HDFGroup (external website) or download it directly

```
wget "https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.12/hdf5-1.12.1/src/hdf5-1.12.
↪1.tar.gz"
```

and extract it, enter the folder, create a build folder and enter it

```
tar -xvf hdf5-1.12.1.tar.gz && cd hdf5-1.12.1 && mkdir -p build && cd build
```

Afterwards, configure HDF5 and specify the installation directory with `--prefix=/home/user/hdf5/1.12.1`

```
../configure --prefix=/home/user/hdf5/1.12.1 --with-pic --enable-fortran --enable-
↪fortran2003 --disable-shared --enable-parallel CC=$(which mpicc) CXX=$(which mpicxx)␣
↪FC=$(which mpifort)
```

After the Makefile has been generated, you can compile HDF5 in parallel, e.g. with 4 cores

```
make -j4 2>&1 | tee make.out
```

and finally install it by

```
make install 2>&1 | tee install.out
```

If you encounter any difficulties, you can submit an issue on GitHub attaching the `make.out` and/or `install.out` files, where the console output is stored. To make sure that the installed HDF5 is also utilized by CMake, you have to set the environment variables, again making sure to use your installation folder and the correct version

```
export HDF5_DIR = /home/user/hdf5/1.12.1
export HDF5_ROOT = /home/user/hdf5/1.12.1
export PATH="/home/user/hdf5/1.12.1/include:$PATH"
export PATH="/home/user/hdf5/1.12.1/bin:$PATH"
export LD_LIBRARY_PATH="/home/user/hdf5/1.12.1/lib:$LD_LIBRARY_PATH"
```

For convenience, you can add these lines to your `.bashrc`.

Installing PETSc

The following list contains the **recommended/working library versions** for PETSc and PICLas

PICLas Version	3.18	3.17
3.0.0	yes	yes
2.9.0	no	yes

Local machine

Download PETSc from the git repository

```
git clone -b release-3.17 https://gitlab.com/petsc/petsc.git petsc
```

Configure and install PETSc (MPI and BLAS/LAPACK have to be installed)

```
./configure PETSC_ARCH=arch-linux --with-mpi-dir=/opt/openmpi/3.1.6/gcc/9.2.0/ --with-
↪debugging=0 COPTFLAGS='-O3 -march=native -mtune=native' CXXOPTFLAGS='-O3 -march=native␣
↪-mtune=native' FOPTFLAGS='-O3 -march=native -mtune=native' --download-hypre --download-
↪mumps --download-scalapack
make all
```

Set the required environment variables

```
export PETSC_DIR=/home/user/petsc
export PETSC_ARCH=arch-linux
```

Set the PETSc flag to ON in during cmake configuration of PICLas

```
PICLAS_PETSC   ON
```

Cluster (HLRS) with restricted internet access

Download PETSc to your local PC and copy to HLRS via ssh

```
wget https://ftp.mcs.anl.gov/pub/petsc/release-snapshots/petsc-3.17.0.tar.gz
scp petsc-3.17.0.tar.gz hawk:~/.
ssh hawk
tar xf petsc-<version number>.tar.gz
```

Load the modules on hawk (note that it was not possible to compile PETSc with mpt/2.23), only OpenMPI due to an MPI variable error

Configuration for the old software stack (Attention: THIS IS OLD):

```
module load  cmake/3.16.4  gcc/9.2.0  hdf5/1.10.5  blis/2.1  openmpi/4.0.4  libflame/2.1␣
↪ aocl/2.1.0
```

Configuration for the new software stack (Attention: THIS IS NEW):

```
module load  cmake/3.15.2 gcc/10.2.0  hdf5/1.10.5  blis/2.1  openmpi/4.1.4
```

Configure PETSc (MPI and BLAS/LAPACK have to be installed), which might fail due to the firewall that is active on hawk and prevents direct internet access. Follow the hints in the error message or simply copy the required external packages from your local system to hawk. If you configure on a local machine, the external packages are downloaded to `petsc-3.17.0/arch-linux/externalpackages/`. These must be copied to hawk, e.g., via

```
cd petsc-3.17.0/arch-linux/externalpackages/
rsync -avPe ssh git.hypre hawk:~/petsc-3.17.0/arch-linux/externalpackages/.
rsync -avPe ssh git.mumps hawk:~/petsc-3.17.0/arch-linux/externalpackages/.
rsync -avPe ssh git.scalapack hawk:~/petsc-3.17.0/arch-linux/externalpackages/.
```

and then the configuration can be done on hawk

```
cd petsc-3.17.0
```

Configuration for the old software stack (Attention: THIS IS OLD):

Note that gcc/9.2.0 and openmpi/4.0.4 must be loaded

```
./configure PETSC_ARCH=arch-linux --with-mpi-dir=/opt/hlrs/non-spack/mpi/openmpi/4.0.4-
↪gcc-9.2.0/ --with-debugging=0 COPTFLAGS='-O3 -march=native -mtune=native' CXXOPTFLAGS=
↪'-O3 -march=native -mtune=native' FOPTFLAGS='-O3 -march=native -mtune=native' --
↪download-hypre --download-mumps --download-scalapack
```

Configuration for the new software stack (Attention: THIS IS NEW without installing fblaslapack):

Note that gcc/10.2.0 and openmpi/4.1.4 must be loaded

```
./configure PETSC_ARCH=arch-linux --with-mpi-dir=/opt/hlrs/non-spack/rev-009_2022-09-01/
↪mpi/openmpi/4.1.4-gcc-10.2.0/ --with-debugging=0 COPTFLAGS='-O3 -march=native -
↪mtune=native' CXXOPTFLAGS='-O3 -march=native -mtune=native' FOPTFLAGS='-O3 -
↪march=native -mtune=native' --download-hypre --download-mumps --download-scalapack --
↪with-blas-lib=/sw/hawk-rh8/hlrs/spack/rev-009_2022-09-01/blis/2.1-gcc-10.2.0-g6f3pga5
```

Configuration for the new software stack (Attention: THIS IS NEW + includes installing fblaslapack):

```
./configure PETSC_ARCH=arch-linux --with-mpi-dir=/opt/hlrs/non-spack/rev-009_2022-09-01/
↪mpi/openmpi/4.1.4-gcc-10.2.0/ --with-debugging=0 COPTFLAGS='-O3 -march=native -
↪mtune=native' CXXOPTFLAGS='-O3 -march=native -mtune=native' FOPTFLAGS='-O3 -
↪march=native -mtune=native' --download-hypre --download-mumps --download-scalapack --
↪download-fblaslapack=1
```

that requires

```
cd petsc-3.17.0/arch-linux/externalpackages/
rsync -avPe ssh git.fblaslapack hawk:~/petsc-3.17.0/arch-linux/externalpackages/.
```

and after configuration, run make

```
make all
```

Set the environment variables

```
export PETSC_DIR=/home/user/petsc
export PETSC_ARCH=arch-linux
```

Set the PETSc flag to ON in during cmake configuration of PICLas

```
PICLAS_PETSC   ON
```

Load the paths and modules in the submit.sh script on hawk by adding the following lines to the submit.sh script

```
module load cmake/3.16.4  gcc/9.2.0  hdf5/1.10.5  libflame/2.1  openmpi/4.0.4  aocl/2.1.
↪0  blis/2.1
export PETSC_DIR=/zhome/academic/HLRS/irs/iagcopp/petsc-3.17.0
export PETSC_ARCH=arch-linux
```

1.1.4 Obtaining the source

The **PICLas** repository is available at GitHub. To obtain the most recent version you have two possibilities:

- Clone the **PICLas** repository from Github

  ```
  git clone https://github.com/piclas-framework/piclas.git
  ```

- Download **PICLas** from Github:

  ```
  wget https://github.com/piclas-framework/piclas/archive/master.tar.gz
  tar xzf master.tar.gz
  ```

Note that cloning **PICLas** from GitHub may not be possible on some machines, as e.g. the HLRS at the University of Stuttgart restricts internet access. Please refer to section *Cloning with the SSH protocol* of this user guide.

1.1.5 Compiling the code

To compile the code, open a terminal, change into the **PICLas** directory and create a new subdirectory. Use the CMake curses interface (requires the `cmake-curses-gui` package on Ubuntu) to configure and generate the Makefile

```
mkdir build && cd build
ccmake ..
```

Here you will be presented with the graphical user interface, where you can navigate with the arrow keys. Before you can generate a Makefile, you will have to configure with `c`, until the generate option `g` appears at the bottom of the terminal. Alternatively, you can configure the code directly by supplying the compiler flags

```
cmake -DPICLAS_TIMEDISCMETHOD=DSMC ..
```

For a list of all compiler options visit Section *Compiler options*. PICLas supports Unix Makefiles (default) and Ninja as generators. To select ninja either export the environment variable `export CMAKE_GENERATOR=Ninja` or add the cmake option `-GNinja`, e.g.

```
cmake -GNinja ..
ccmake -GNinja ..
```

After a successful generation of the Makefile (or ninja build files), compile the code in parallel (e.g. with 4 cores) using

```
make -j4 2>&1 | tee piclas_make.out
```

or the corresponding Ninja command

```
ninja -j4 2>&1 | tee piclas_make.out
```

To use all available cores for parallel compilation use either `make -j` or `ninja -j0`. Finally, the executables **piclas** and **piclas2vtk** are contained in your **PICLas** directory in `build/bin/`. If you encounter any difficulties, you can submit an issue on GitHub attaching the `piclas_make.out` file, where the console output is stored.

Directory paths

In the following, we write `$PICLASROOT` as a substitute for the path to the **PICLas** repository. Please replace `$PICLASROOT` in all following commands with the path to your **PICLas** repository *or* add an environment variable `$PICLASROOT`.

Furthermore, the path to executables is omitted in the following, so for example, we write `piclas` instead of `$PICLASROOT/build/bin/piclas`.

In order to execute a file, you have to enter the full path to it in the terminal. There are two different ways to enable typing `piclas` instead of the whole path (do not use both at the same time!)

1. You can add an alias for the path to your executable. Add a command of the form

```
alias piclas='$PICLASROOT/build/bin/piclas'
```

to the bottom of the file `~/.bashrc`. Source your `~/.bashrc` afterwards with

```
. ~/.bashrc
```

2. You can add the **PICLas** binary directory to your `$PATH` environment variable by adding

```
export PATH=$PATH:$PICLASROOT/build/bin
```

to the bottom of the file `~/.bashrc` and sourcing your `~/.bashrc` afterwards.

Now you are ready for the utilization of **PICLas**.

1.2 Mesh Generation

PICLas utilizes computational meshes from the high order preprocessor **HOPR** in the HDF5 format. It is available under GPLv3 at https://github.com/hopr-framework/hopr and is included as part of the PICLas installation. To compile HOPR together with PICLas, you can set the following CMake variable in the CMake GUI:

```
LIBS_BUILD_HOPR          ON
```

or add the following variable to the console command:

```
cmake -DPICLAS_TIMEDISCMETHOD=DSMC -DLIBS_BUILD_HOPR=ON ..
```

The source code will be downloaded and built automatically, afterwards the executable `hopr` will be placed in the `build/bin/` folder next to the other executables. If you would like to install HOPR directly from the repository by yourself (e.g. if you would like to join the development), please refer to the HOPR documentation.

The design philosophy is that all tasks related to mesh organization, different input formats and the construction of high order geometrical mappings are separated from the *parallel* simulation code. These tasks are implemented most efficiently in a *serial* environment. The employed mesh format is designed to make the parallel read-in process as simple and fast as possible. For details concerning the mesh format please refer to the HOPR HDF5 Curved Mesh Format Documentation.

The basic command for either mesh generation or conversion of an external mesh is

```
hopr hopr.ini
```

Note that the path to the **HOPR** executable, which depends on whether HOPR was installed together with PICLas or separately, is omitted in the command. For the former, the path set in Section *Directory paths* for PICLas applies for the HOPR executable as well.

1.2.1 Mesh generation with HOPR

Using **HOPR**, simple, structured meshes can be directly created using an in-built mesh generator. A number of strategies to create curved boundaries are also included in HOPR.

1.2.2 Mesh conversion with HOPR

More complex geometries can be treated by importing meshes generated by external mesh generators in CGNS or GMSH format (Example parameter file). Detailed instructions for some mesh generators that we have experience with (Gmsh, Coreform Cubit, OMNIS/HEXPRESS, GridPro, CENTAUR, MeshGems within SALOME) are given below.

Mesh generation with Gmsh

Gmsh is an open-source mesh generator with a built-in CAD engine. As of Gmsh 4.11.0 (October 2022), meshes can be exported in the `.msh` format. For the mesh conversion with HOPR, it is required to enable the option `Save all elements` and chose the ASCII code for the file export. The mesh file can be converted with HOPR, using the corresponding mode:

```
Mode = 5
```

Tetrahedral or mixed meshes are created by default. Hexahedral meshes can be generated directly in gmsh by using the subdivision algorithm or by using the following option in HOPR for a tetrahedral mesh:

```
SplitToHex = TRUE
```

Boundary conditions have to be defined as physical groups. A tutorial for the mesh generation with gmsh can be found in the tutorial *Hypersonic Flow around the 70° Cone (DSMC) - 3D Mesh with Gmsh*, which requires hopr version v.1.1.0 or newer.

Mesh generation with Coreform Cubit

As of Coreform Cubit 2021.2 (November 2021), the CGNS export is not implemented for unstructured meshes. The Gambit file format can be utilized instead. Boundary conditions have to be defined as side sets first with the names as they will be used in the simulation. After the side set definitions, additional CFD BCs have to be mapped to the side sets (type is irrelevant, no name required: `System-assigned ID`). The exported Gambit file (`*.neu`) can be converted with HOPR, using the corresponding mode:

```
Mode = 2
```

Mesh generation with OMNIS/HEXPRESS

As of OMNIS Version 5.2 (November 2021), the CGNS export is not suitable for boundaries with more than one surface (even if they have been grouped together correctly in OMNIS). Instead, the mesh should be exported in the `.sph` format. The following script can be utilized to merge the surfaces

```
#####################################################
# Script to merge surfaces and CGNS export
#####################################################
# Make sure to insert the correct file name
READMESH merge.sph
# First, all surfaces within "WALL_IN_OMNIS" BC (as defined in OMNIS) are summarized␣
↪under the new BC "BC_WALL" (the name to use in HOPR)
UNITESELECTIONS BC_WALL 1 WALL_IN_OMNIS*
# Second, the original BC group and its surfaces should be deleted. These two commands␣
↪can be repeated for multiple boundary conditions within a single script.
DELETESELECTION WALL_IN_OMNIS*
# Export the mesh as CGNS for the conversion with HOPR
WRITEMESH mesh.cgns
END
```

The above script can be saved in a text file (e.g. `omnis_mergeBCs.conf`) and executed using the datamapper tool

```
hexpressdatamapper101 omnis_mergeBCs.conf
```

The resulting CGNS file can be converted with HOPR, using the corresponding mode:

```
Mode = 3
```

Mesh generation with GridPro

GridPro is a proprietary conforming multi-block mesh generator with hexahedral elements. However, a free academic version limited to 250 blocks is available.

After mesh generation, and before naming the boundaries in the *Property Setter*, you should set the output format to STARCD. Make sure to define not only labels but also different properties for the boundaries. Then export as STARCD and you will get four output files. During the export GridPro loses the label information, thus the boundary names have to be set again in the *.inp file. An example of a correct *.inp is given below:

```
TITLE
Converted from GridPro v4.1
CTAB 1 FLUI
CTNA 1 ZONE_VOL
RDEF 1 WALL $ $ $ $ $ $ $ $
RNAM 1 BC_WALL
RDEF 2 CYCL $ $ $ $ $ $ $ $
RNAM 2 BC_CYCL
VREAD,/home/user/mesh/test.vrt,,1,43770,CODE
CREAD,/home/user/mesh/test.cel,,1,21504,MODI,CODE
BREAD,/home/user/mesh/test.bnd,,1,43768,MODI,CODE
```

HOPR can then read-in the mesh with following mode option:

```
Mode = 4
```

More recent versions of GridPro also support a CGNS output. Here, the option *Export -> Grid -> CGNS -> Elementary* should be chosen. For different boundary labels, different property types have to be defined (Note: The property type *Wall* might be causing problems during the HOPR read-in and should be avoided). The following errors can be ignored as long as HOPR finishes successfully and a mesh file is written out

```
ERROR: number of zones in inifile does not correspond to number of zones in meshfile(s)
ERROR - Could not find corresponding boundary definition of ws.Interblck
```

Mesh generation with CENTAUR

A conventional tetrahedral mesh can be generated with CENTAUR. Boundaries have to be set in CENTAUR and accordingly in the HOPR parameter file. During the export as CGNS the following options are required:

- Check "Only write out boundary faces"
- Check "Write out boundary faces grouped by panels"

Read-in and convert with HOPR using the following options:

```
Mode = 3
BugFix_ANSA_CGNS = TRUE
SplitToHex = TRUE
```

Should problems occur, try to set SpaceQuandt to a higher value, e.g. 100. During the pre-processing step, every tetrahedron will be converted to 4 hexahedra resulting in increased number of elements.

Mesh generation with MeshGems/SALOME

Note: This tutorial was last updated/used: June 2016

MeshGems-Hexa is proprietary automated all-hex mesh generator. The algorithm can be used a plug-in within the open-source platform SALOME. Currently, we do not have a MeshGems-Hexa license and cannot give any support on mesh generation.

- Import geometry as STEP or IGS
- Create groups in Geometry module for the boundary conditions
  - Right-click on the imported file in Object Browser -> "Create Group"
  - Select faces as "Share Type" (2D object)
  - Choose appropriate name for the BC
  - Select one or many surfaces by (shift-) clicking on the surfaces
  - Click "Add" when finished for a single BC
  - "Apply" to save the BC and repeat the process (or "Apply and Close" at the end)
- Create mesh in Mesh module: "Mesh" -> "Create mesh"
  - Choose the imported geometry in "Geometry" (if not already chosen, when geometry was highlighted previously)
  - Choose MG-Hexa for the 3D algorithm
  - Choose MG-CADsurf for the 2D algorithm

- Assign boundary conditions to mesh
  - Right-click on created mesh
  - "Create groups from Geometry"
    - Click on the BC in "Object Browser" in the "Geometry"-tree
    - "Apply" and repeat with the next BC
    - "Apply and close" with the last BC
- Export mesh as CGNS
  - Right-click on mesh, choose "Export" -> "CGNS file"
  - Import the exported CGNS file
  - Check the names of boundary conditions
    - Activate the BC by clicking on the visibility icon next to the name
    - Change the name to correspond to the entry in the preproc.ini
  - Export the modified mesh again as CGNS
- Convert CGNS mesh from hdf to adf format Install cgns-tools ("sudo apt-get install cgns-tools") "hdf2adf filename"

Read-in and convert with HOPR and the following options:

```
Mode = 3
BugFix_ANSA_CGNS = TRUE
```

1.3 Workflow

In this chapter, the complete process of setting up a simulation with **PICLas** is detailed.

1.3.1 Compiler options

This section describes the main configuration options which can be set when building **PICLas** using CMake. Some options are dependent on others being enabled (or disabled), so the available ones may change.

The first set of options describe general CMake behaviour:

- `CMAKE_BUILD_TYPE`: This statically specifies what build type (configuration) will be built in this build tree. Possible values are
  - Release: "Normal" execution.
  - Profile: Performance profiling using gprof.
  - Debug: Debug compiler for detailed error messages during code development.
  - Sanitize: Sanitizer compiler for even more detailed error messages during code development.
  - Nitro: Fast compiler option `-Ofast` for even more speed but at the cost of accuracy.
- `CMAKE_HOSTNAME`: This will display the host name of the machine you are compiling on.
- `CMAKE_INSTALL_PREFIX`: If "make install" is invoked or INSTALL is built, this directory is prepended onto all install directories This variable defaults to /usr/local on UNIX.

For some external libraries and programs that **PICLas** uses, the following options apply:

- CTAGS_PATH: This variable specifies the Ctags install directory, an optional program used to jump between tags in the source file.
- LIBS_BUILD_HOPR: Enable the compilation of the mesh pre-processor HOPR during the PICLas compilation. The executable `hopr` will be placed in the build/bin/ folder next to the other executables. For more details, on the utilization of HOPR, see *Mesh Generation*.
- LIBS_DOWNLOAD_HOPR: Enable downloading the mesh pre-processor HOPR during the PICLas compilation from GitHub. The executable `hopr` will be linked in the build/bin/ folder next to the other executables. For more details, on the utilization of HOPR, see *Mesh Generation*.
- LIBS_BUILD_HDF5: This will be set to ON if no pre-built HDF5 installation was found on your machine. In this case a HDF5 version will be built and used instead. For a detailed description of the installation of HDF5, please refer to Section *Installing HDF5*.
- HDF5_DIR: If you want to use a pre-built HDF5 library that has been built using the CMake system, this directory should contain the CMake configuration file for HDF5 (optional).
- PICLAS_MEASURE_MPI_WAIT: Measure the time that is spent in MPI_WAIT() and output info to MPIW8Time.csv and MPIW8TimeProc.csv
- PICLAS_BUILD_POSTI: Enables the compilation of additional tools and activates the following options:
  - POSTI_BUILD_SUPERB: Enables the compilation of **superB**, which is allows the computation of magnetic fields based on an input of coils and permanent magnets, see Section *superB*
  - POSTI_BUILD_VISU: Enables the compilation of the post-processing tool **piclas2vtk**, which enables the conversion of output files into the VTK format
  - POSTI_USE_PARAVIEW: Enables the compilation of the ParaView plugin, which enables the direct read-in of output files within ParaView
- PICLAS_SHARED_MEMORY: Split type for creating new communicators based on colors and keys (requires MPI 3 or higher). Options with the prefix OMPI_ are specific to Open MPI.
  - MPI_COMM_TYPE_SHARED: creates one shared memory domain per physical node (default)
  - OMPI_COMM_TYPE_CORE: creates one shared memory domain per MPI thread
  - PICLAS_COMM_TYPE_NODE: creates one shared memory domain per X numbers of MPI threads defined by PICLAS_SHARED_MEMORY_CORES
    - PICLAS_SHARED_MEMORY_CORES: Number of MPI threads per virtual node (default is 2). Assumes that all MPI threads run on the same physical node.

Some settings are not shown in the graphical user interface, but can be changed via command line

- PICLAS_INSTRUCTION: Processor instruction settings (mainly depending on the hardware on which the compilation process is performed or the target hardware where piclas will be executed). This variable is set automatically depending on the machine where piclas is compiled. CMake prints the value of this parameter during configuration

```
-- Compiling Nitro/Release/Profile with [GNU] (v12.2.0) fortran compiler using␣
↪PICLAS_INSTRUCTION [-march=native] instructions.
```

  When compiling piclas on one machine and executing the code on a different one, the instruction setting should be set to `generic`. This can be accomplished by running

```
cmake -DPICLAS_INSTRUCTION=-mtune=generic
```

  To reset the instruction settings, run cmake again but with

```
-DPICLAS_INSTRUCTION=
```

which resorts to using the automatic determination depending on the detected machine.

1.3.2 Solver settings

Before setting up a simulation, the code must be compiled with the desired parameters. The most important compiler options to be set are:

- `PICLAS_TIMEDISCMETHOD`: Time integration method
  - Leapfrog: 2nd order when only electric fields are relevant (poisson solver)
  - Boris-Leapfrog: 2nd order for electric and magnetic fields (poisson solver)
  - Higuera-Cary: 2nd order for electric and magnetic fields (poisson solver)
  - RK3: Runge-Kutta 3rd order in time
  - RK4: Runge-Kutta 4th order in time
  - RK14: Low storage Runge-Kutta 4, 14 stages version - Niegemann et al 2012
  - DSMC: Direct Simulation Monte Carlo, Section *Direct Simulation Monte Carlo*
  - FP-Flow: Fokker-Planck-based collision operator, Section *Fokker-Planck Collision Operator*
  - BGK-Flow: Bhatnagar-Gross-Krook collision operator, Section *Bhatnagar-Gross-Krook Collision Operator*
- `PICLAS_EQNSYSNAME`: Equation system to be solved
  - maxwell:
  - poisson:
- `PICLAS_POLYNOMIAL_DEGREE`: Defines the polynomial degree of the solution. The order of convergence follows as $N+1$. Each grid cell contains $(N+1)^3$ collocation points to represent the solution.
- `PICLAS_NODETYPE`: The nodal collocation points used during the simulation
  - GAUSS: Legendre-Gauss distributed nodes
  - GAUSS-LOBATTO: Legendre-Gauss-Lobatto distributed nodes
- `PICLAS_INTKIND8`: Enables simulations with particle numbers above 2 147 483 647
- `PICLAS_READIN_CONSTANTS`: Enables user-defined natural constants for the speed of light *c0*, permittivity *eps* and permeability *mu* of vacuum, which must then be supplied in the parameter file. The default if *OFF* and the values for the speed of light c0=299792458.0 [m/s], permittivity eps=8.8541878176e-12 [F/m] and permeability mu=1.2566370614e-6 [H/m] are hard-coded.

The options EQNSYSNAME, POLYNOMIAL_DEGREE and NODETYPE can be ignored for a DSMC simulation. For parallel computation the following flags should be configured:

- `LIBS_USE_MPI`: Enabling parallel computation. For a detailed description of the installation of MPI, please refer to refer to Section *Installing OpenMPI*.
- `PICLAS_LOADBALANCE`: Enable timer-based load-balancing by automatic determination of workload weights for each simulation element.

All other options are set in the parameter file.

1.3.3 Setup of parameter file(s)

The settings of the simulation are controlled through parameter files, which are given as arguments to the binary. In the case of PIC simulations the input of a single parameter file (e.g. *parameter.ini*) is sufficient, while the DSMC method requires the input of a species parameter file (e.g. *DSMC.ini*). The most recent list of parameters can be found by invoking the help in the console:

```
piclas --help
```

General parameters such the name of project (used for filenames), the mesh file (as produced by HOPR), end time of the simulation (in seconds) and the time step, at which the particle data is written out (in seconds), are:

```
ProjectName    = TestCase
MeshFile       = test_mesh.h5
TEnd           = 1e-3
Analyze_dt     = 1e-4
ManualTimeStep = 1e-4 (over-rides the automatic time step calculation in the Maxwell␣
↪solver)
```

Generally following types are used:

```
INTEGER = 1
REAL    = 1.23456
REAL    = 1.23E12
LOGICAL = T         ! True
LOGICAL = F         ! False
STRING  = PICLAS
VECTOR  = (/1.0,2.0,3.0/)
```

The concept of the parameter file is described as followed:

- Each single line is saved and examined for specific variable names
- The examination is case-insensitive
- Comments can be set with symbol "!" in front of the text
- Numbers can also be set by using "pi"

```
   vector = (/1,2Pi,3Pi/)
```

- The order of defined variables is irrelevant, except for the special case when redefining boundaries. However, it is preferable to group similar variables together.

The options and underlying models are discussed in Chapter *Features & Models*, while the available output options are given in Chapter *Visualization & Output*. Due to the sheer number of parameters available, it is advisable to build upon an existing parameter file from one of the tutorials in Chapter *Tutorials*.

1.3.4 Simulation

After the mesh generation, compilation of the binary and setup of the parameter files, the code can be executed by

```
piclas parameter.ini [DSMC.ini]
```

The simulation may be restarted from an existing state file

```
piclas parameter.ini [DSMC.ini] [restart_file.h5]
```

The state file , e.g., TestCase_State_000.5000000000000000.h5, contains all the required information to continue the simulation from this point in time

```
piclas parameter.ini DSMC.ini TestCase_State_000.5000000000000000.h5
```

A state file is generated at the end of the simulation and also at every time step defined by `Analyze_dt`. **Note:** When restarting from an earlier time (or zero), all later state files possibly contained in your directory are deleted!

After a successful simulation, state files will be written out in the HDF5 format preceded by the project name, file type (e.g. State, DSMCState, DSMCSurfState) and the time stamp:

```
TestCase_State_001.5000000000000000.h5
TestCase_DSMCState_001.5000000000000000.h5
```

The format and floating point length of the time stamp *001.5000000000000000* can be adjusted with the parameter

```
TimeStampLength = 21
```

where the floating format with length of *F21.14* is used as default value.

Parallel execution

The simulation code is specifically designed for (massively) parallel execution using the MPI library. For parallel runs, the code must be compiled with `PICLAS_MPI=ON`. Parallel execution is then controlled using `mpirun`

```
mpirun -np [no. processors] piclas parameter.ini [DSMC.ini] [restart_file.h5]
```

The grid elements are organized along a space-filling curved, which gives a unique one-dimensional element list. In a parallel run, the mesh is simply divided into parts along the space filling curve. Thus, domain decomposition is done *fully automatic* and is not limited by e.g. an integer factor between the number of cores and elements. The only limitation is that the number of cores may not exceed the number of elements.

Profile-guided optimization (PGO)

To further increase performance for production runs, profile-guided optimization can be utilized with the GNU compiler. This requires the execution of a representative simulation run with PICLas compiled using profiling instrumentation. For this purpose, the code has to be configured and compiled using the following additional settings and the `Profile` build type:

```
-DPICLAS_PERFORMANCE=ON -DUSE_PGO=ON -DCMAKE_BUILD_TYPE=Profile
```

A short representative simulation has to be performed, where additional files with the profiling information will be stored. Note that the test run should be relatively short as the code will be substantially slower than the regular `Release` build type. Afterwards, the code can be configured and compiled again for the production runs, using the `Release` build type:

```
-DPICLAS_PERFORMANCE=ON -DUSE_PGO=ON -DCMAKE_BUILD_TYPE=Release
```

Warnings regarding missing profiling files (`-Wmissing-profile`) can be ignored, if they concern modules not relevant for the current simulation method (e.g. `bgk_colloperator.f90` will be missing profile information if only a DSMC simulation has been performed).

1.3.5 Post-processing

PICLas comes with a tool for visualization. The piclas2vtk tool converts the HDF5 files generated by **PICLas** to the binary VTK format, readable by many visualization tools like ParaView and VisIt. The tool is executed by

```
piclas2vtk [posti.ini] output.h5
```

Multiple HDF5 files can be passed to the piclas2vtk tool at once. The (optional) runtime parameters to be set in `posti.ini` are given in Chapter *Visualization & Output*.

1.4 Features & Models

1.4.1 Particle Tracking

Three different particle tracking methods are implemented in PICLas and are selected via

```
TrackingMethod = triatracking ! Define Method that is used for tracking of
                              ! particles:
                              ! refmapping (1): reference mapping of particle
                              ! position with (bi-)linear and bezier (curved)
                              ! description of sides.
                              ! tracing (2): tracing of particle path with
                              ! (bi-)linear and bezier (curved) description of
                              ! sides.
                              ! triatracking (3): tracing of particle path with
                              ! triangle-aproximation of (bi-)linear sides.
```

For conventional computations on (bi-, tri-) linear meshes, the following tracking algorithm is recommended:

```
TrackingMethod = triatracking
```

Following options are available to get more information about the tracking, e.g. number of lost particles:

Option	Values	Notes
DisplayLostParticles	F/T	Display position, velocity, species and host element of particles lost during particle tracking (TrackingMethod = triatracking, tracing) in the std.out
CountNbrOfLostParts	T/F	Count number of lost particles due to tolerance issues. This number is a global number, summed over the full simulation duration and includes particles lost during the restart. The lost particles are output in a separate `*_PartStateLost*.h5` file.

The two alternative tracking routines and their options are described in the following.

DoRefMapping

```
TrackingMethod = refmapping
```

This method is the slowest implemented method for linear grids and large particle displacements. A particle is mapped into a element to compute the particle position in the reference space. This test determines in which element a particle is located. Each element has a slightly larger reference space due to tolerance. Starting from reference values >=1. the best element is found and used for the hosting element. In order to take boundary interactions into account, all BC faces in the halo vicinity of the element are checked for boundary interactions and a boundary condition is performed accordingly. This algorithm has an inherent self-check. If a boundary condition is not detected, the particle position is located outside of all elements. A fall-back algorithm is then used to recompute the position and boundary interaction. Periodic domains are only possible for Cartesian meshes. The particle position is used for periodic displacements.

Option	Values	Notes
CartesianPeriodic	T/F	If a fully periodic box (all 6 sides) is computed, the intersections do not have to be computed. Instead, each particle can be simply shifted by the periodic vector.
FastPeriodic	T/F	Moves particle the whole periodic distance once, which can be several times the mesh size in this direction.

Tracing

```
TrackingMethod = tracing
```

This method traces the particle trajectory throughout the domain. The initial element is determined by computing the intersection between the particle-element-origin vector and each element face. If none of the six element faces are hit, the particle is located inside of this element. Next, the particle trajectory is traced throughout the domain. Hence, each face is checked for an intersection and a particle assigned accordingly to neighbor elements or the interaction with boundary conditions occur. This algorithm has no inherent self-consistency check. For critical intersections (beginning or end of a particle path or if a particle is located close to the edges of element faces) an additional safety check is performed by recomputing the element check and if it fails a re-localization of the particle is required. Particles

traveling parallel to element faces are in an undefined state and are currently removed from the computation. This leads to a warning message.

Parameters for DoRefMapping and Tracing (NEEDS UPDATING)

Following parameters can be used for both schemes.

Option	Values	Notes
MeasureTrackTime	T/F	Measure the time required for tracking and init local.
RefMappingGuess	1-4	Prediction of particle position in reference space:
	1	Assumption of a linear element coord system.
	2	Gauss point which is closest to the particle.
	3	CL point which is closest to the particle.
	4	Trivial guess: element origin
RefMappingEps	1e-4	Tolerance of the Newton algorithm for mapping in ref. space. It is the L2 norm of the delta Xi in ref space.
BezierElevation	0-50	Increase polynomial degree of BezierControlPoints to construct a tighter bounding box for each side.
BezierSampleN	NGeo	Polynomial degree to sample sides for SurfaceFlux and Sampling of DSMC surface data.
BezierNewtonAngle	$< PI/2$	Angle to switch between Clipping and a Newton algorithm.
BezierClipTolerance	1e-8	Tolerance of Bezier-Clipping and Bezier-Newton
BezierClipHit	1e-6	Tolerance to increase sides and path during Bezier-Algo.
BezierSplitLimit	0.6	Minimum degrees of side during clipping. A larger surface is spit in two to increase convergence rate and predict several intersections.
BezierClipMaxIntersec	2*NGeo	Maximum number of roots for curvilinear faces.
epsilontol	100*epsM	Tolerance for linear and bilinear algorithm.

Rotating Frame of Reference

Beside the described methods of particle tracking, it is important to define the frame of reference for the correct simulation of particle trajectories. Per default the resting frame of reference is used and no further settings are required. The rotating reference frame can be used to represent rotating geometries like e.g. turbine blades, since rotating/changing meshes are currently not supported. The corresponding rotational wall velocity has to be defined for the boundary as well, as shown in Section *Wall movement (Linear & rotational)*. The distinction between a resting and rotating frame of reference is only important for the particle movement step. Here particles are not moving on a straight line due to the pseudo forces, i.e. the centrifugal and the Coriolis force. This means that particles follow a circular path towards a stationary boundary that represents a rotating geometry. The usage of the rotating reference frame is enabled by

```
Part-UseRotationalReferenceFrame = T
```

Additionally, the rotational axis (x-, y- or z-axis) and frequency have to be defiend by

```
Part-RotRefFrame-Axis = 1          ! x=1, y=2, z=3
Part-RotRefFrame-Frequency = -100  ! [Hz, 1/s]
```

The sign of the frequency (+/-) defines the direction of rotation according to the right-hand rule. It is also possible to use both reference frames within a single simulation. For this purpose, regions can be defined in which the rotating frame of reference is to be used. First, the number of rotating regions is defined by

```
Part-nRefFrameRegions = 2
```

Afterwards the minimum and maximum coordinates must be defined for each region. Both values refer to the coordinates on the rotational axis, since the boundary surfaces of these regions can only be defined perpendicular to the rotation axis:

```
Part-RefFrameRegion1-MIN = 10
Part-RefFrameRegion1-MAX = 20
Part-RefFrameRegion2-MIN = 100
Part-RefFrameRegion2-MAX = 110
```

This allows to model systems of rotating and stationary geometries (e.g. pumps with stator and rotor blades) within a single simulation. For rotationally symmetric cases, the simulation domain can be reduced using the rotationally periodic boundary condition (as shown in Section *Rotational Periodicity*). Examples are provided in the regression test directory, e.g. `regressioncheck/CHE_DSMC/Rotational_Reference_Frame` and `regressioncheck/CHE_DSMC/Rotational_Reference_Frame_Regions`.

Time step subcycling for rotating frame of reference

In PICLas, an explicit time stepping scheme is used for the DSMC method, with both collision and motion operators altering the particle distribution function within each iteration. This leads to changes in the particle positions, momentum, and energy due to motion and collisions. Operators can be sequentially executed through operator splitting, adjusting the particle positions based on velocities first, followed by collisions within a time step. It is crucial for the time step to resolve collision frequency adequately. External forces (i.e. the centrifugal and the Coriolis force in the case of a rotating reference frame) may require additional consideration for the time step determination, especially when particle acceleration needs to be modeled. To ensure that the existing time step requirement in DSMC, dictated by collisions, remains unaffected, a subcycling algorithm only for the particle motion can be used. This algorithm divides the motion and thus the modeling of external forces into smaller subtimesteps. Consequently, the time step can be chosen based on collision frequency, while the motion can be more finely resolved through subcycling. The usage of the subcycling algorithm is enabled by

```
Part-RotRefFrame-UseSubCycling = T
```

Additionally, the number of the subcycling steps can be defined by

```
Part-RotRefFrame-SubCyclingSteps = 10 ! Default = 10 steps
```

1.4.2 Field Solver - Poisson Equation

To numerically solve electrostatic problems, PICLas offers a solver for Poisson's equation

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0}$$

where the charge density ρ is given by the charged particles within the system. To enable a simulation using the Poisson field solver, the code must be compiled using the following setting

```
PICLAS_EQNSYSNAME     = poisson
PICLAS_TIMEDISCMETHOD = Euler-Explicit, Leapfrog, Boris-Leapfrog, Higuera-Cary, RK3, RK4
```

where one of the available time discretization methods for the particle evolution must be chosen for `PICLAS_TIMEDISCMETHOD`, which are also referred to as particle pushers. There are different options available that yield different functionalities.

PICLAS_TIMEDISCMETHOD	Electric field	Magnetic field	Order of Convergence
Euler-Explicit	yes	no	1
Leapfrog	yes	no	2
Boris-Leapfrog	yes	yes	2
Higuera-Cary	yes	yes	2
RK3	yes	yes	3
RK4	yes	yes	4

Note that high-order time discretization methods in general allow for a larger time step and are usually more costly per time step. To see the available parameter input file options, simply run

```
./bin/piclas --help HDG
```

CG Solver

The default numerical method for solving the resulting system of linear equations, is the Conjugate Gradient Method. The following input parameter can be set to control the simulation

```
epsCG     = 1e-6 ! default value for the abort residual
MaxIterCG = 500  ! default value for the number of CG solver iterations
```

where `epsCG` is the residual of the CG solver and `MaxIterCG` are the maximum number of iteration performed in one time step to solve the system. Furthermore, the residual can be either set absolute (default) or relative via

```
useRelativeAbortCrit = F ! default
```

PETSc Solver

A multitude of different numerical methods to solve the resulting system of linear equations is given by the implemented PETSc library [7], [8], [9]. For detailed installation steps of PETSc within PICLas, see Section *Installing PETSc*. To use PETSc, another flag must be set during the compilation of PICLas

```
PICLAS_PETSC = ON
```

and the parameter input file for the simulation requires setting

```
PrecondType = 2 ! default
```

where the following options are possible

PrecondType	Iterative or Direct	Method
1	iterative	Krylov subspace
2	iterative	Krylov subspace
3	iterative	Krylov subspace
10	direct	

Note that the same parameter setting for `epsCG` will result in a smaller residual with PETSc as compared with the default CG solver without using the PETSc library.

1.4.3 Boundary Conditions - Field Solver

Boundary conditions are defined in the mesh creation step in the hopr.ini file and can be modified when running PICLas in the corresponding *parameter.ini* file. In the *hopr.ini* file, which is read by the *hopr* executable, a boundary is defined by

```
BoundaryName = BC_Inflow   ! BC index 1 (from  position in the parameter file)
BoundaryType = (/4,0,0,0/) ! (/ Type, curveIndex, State, alpha /)
```

where the name of the boundary is directly followed by its type definition, which contains information on the BC type, curving, state and periodicity. This can be modified in the *parameter.ini* file, which is read by the *piclas* executable via

```
BoundaryName = BC_Inflow ! BC name in the mesh.h5 file
BoundaryType = (/5,0/)   ! (/ Type, State /)
```

In this case the boundary type is changed from 4 (in the mesh file) to 5 in the simulation.

Maxwell's Equations

The boundary conditions used for Maxwell's equations are defined by the first integer value in the *BoundaryType* vector (consisting of the *Type* and *State*) and include, periodic, Dirichlet, Silver-Mueller, perfectly conducting, symmetry and reference state boundaries as detailed in the following table.

(/Type,State/)	Type	State
(/1,1/)	periodic	1: positive direction of the 1st periodicity vector
(/1,-1/)	periodic	-1: negative (opposite) direction of the 1st periodicity vector
(/2,2/)	Dirichlet	2: Coaxial waveguide
(/2,22/)	Dirichlet	22: Coaxial waveguide BC (boundary condition or exact flux)
(/2,3/)	Dirichlet	3: Resonator
(/2,4/)	Dirichlet	4: Electromagnetic dipole (implemented via RHS source terms and shape function deposition)
(/2,40/)	Dirichlet	40: Electromagnetic dipole without initial condition (implemented via RHS source terms and shape fu
(/2,41/)	Dirichlet	41: Pulsed Electromagnetic dipole (implemented via RHS source terms and shape function deposition)
(/2,5/)	Dirichlet	5: Transversal Electric (TE) plane wave in a circular waveguide
(/2,7/)	Dirichlet	7: Special manufactured Solution

continu

Table 1.3 – continued from previous page

(/Type,State/)	Type	State
(/2,10/)	Dirichlet	10: Issautier 3D test case with source (Stock et al., div. correction paper), domain [0;1]^3
(/2,12/)	Dirichlet	12: Plane wave
(/2,121/)	Dirichlet	121: Pulsed plane wave (infinite spot size) and temporal Gaussian
(/2,14/)	Dirichlet	14: Gaussian pulse is initialized inside the domain (usually used as initial condition and not BC)
(/2,15/)	Dirichlet	15: Gaussian pulse with optional delay time *tDelayTime*
(/2,16/)	Dirichlet	16: Gaussian pulse which is initialized in the domain and used as a boundary condition for t>0
(/2,50/)	Dirichlet	50: Initialization and BC Gyrotron - including derivatives
(/2,51/)	Dirichlet	51: Initialization and BC Gyrotron - including derivatives (nothing is set for z>eps)
(/3,0/)	SM	1st order absorbing BC (Silver-Mueller) - Munz et al. 2000 / Computer Physics Communication 130, 8
		of div. correction field for low B-fields that only set the correction fields when ABS(B)>1e-10
(/5,0/)	SM	1st order absorbing BC (Silver-Mueller) - Munz et al. 2000 / Computer Physics Communication 130, 8
(/6,0/)	SM	1st order absorbing BC (Silver-Mueller) - Munz et al. 2000 / Computer Physics Communication 130, 8
		of div. correction field for low B-fields that only set the correction fields when B is significantly large c
(/4,0/)	PEC	Perfectly electric conducting surface (Munz, Omnes, Schneider 2000, pp. 97-98)
(/10,0/)	Symmetry	Symmetry BC (perfect MAGNETIC conductor, PMC)
(/20,0/)	Ref	Use state that is read from .h5 file and interpolated to the BC

Dielectric -> type 100?

Poisson's Equation

The boundary conditions used for Maxwell's equations are defined by the first integer value in the *BoundaryType* vector (consisting of the *Type* and *State*) and include, periodic, Dirichlet (via pre-defined function, zero-potential or *RefState*), Neumann and reference state boundaries as detailed in the following table.

(/Type,State/)	Type	State
(/1,1/)	periodic	1: positive direction of the 1st periodicity vector
(/1,-1/)	periodic	-1: negative (opposite) direction of the 1st periodicity vector
(/2,0/)	Dirichlet	0: Phi=0
(/2,2/)	Dirichlet	2: Automatic adjustment for Phi to meet const. input power, see *Power control*
(/2,1001/)	Dirichlet	1001: linear potential y-z via Phi = 2340y + 2340z
(/2,101/)	Dirichlet	101: linear in z-direction: z=-1: 0, z=1, 1000
(/2,103/)	Dirichlet	103: dipole
(/2,104/)	Dirichlet	104: solution to Laplace's equation: Phi_xx + Phi_yy + Phi_zz = 0
		$\Phi = (COS(x) + SIN(x))(COS(y) + SIN(y))(COSH(SQRT(2.0)z) + SINH(SQRT(2.0)z))$
(/2,200/)	Dirichlet	200: Dielectric Sphere of Radius R in constant electric field E_0 from book: John David Jackson, Class
(/2,300/)	Dirichlet	300: Dielectric Slab in z-direction of half width R in constant electric field E_0:
		adjusted from CASE(200)
(/2,301/)	Dirichlet	301: like CASE=300, but only in positive z-direction the dielectric region is assumed
(/2,400/)	Dirichlet	400: Point Source in Dielectric Region with
		epsR_1 = 1 for x $<$ 0 (vacuum)
		epsR_2 != 1 for x $>$ 0 (dielectric region)

cont

Table 1.4 – continued from previous page

(/Type,State/)	Type	State
(/4,0/)	Dirichlet	zero-potential (Phi=0)
(/5,1/)	Dirichlet	1: use RefState Nbr 1 and $\cos(\omega t)$ function (for details see *RefState boundaries*)
(/6,1/)	Dirichlet	1: use RefState Nbr 1 and $\cos(\omega t)+1$ function that does not switch sign (for details see *RefState bound*
(/7,1/)	Dirichlet	1: use LinState Nbr 1, linear function for Phi, see *Linear potential function*
(/8,1/)	Dirichlet	8: Assign BC to EPC group nbr. 1 (different BCs can be assigned the same EPC), see *Electric potential*
(/10,0/)	Neumann	zero-gradient (dPhi/dn=0)
(/11,0/)	Neumann	q*n=1
(/20,1/)	FPC	1: Assign BC to FPC group nbr. 1 (different BCs can be assigned the same FPC), see *Floating boundar*
(/50,0/)	Dirichlet	*Bias Voltage DC* (0: this number has no meaning)
(/51,1/)	Dirichlet	*Bias Voltage AC* (1: use RefState Nbr 1, see *RefState boundaries*)
(/52,1/)	Dirichlet	*Bias Voltage AC and Fixed coupled power* (1: use RefState Nbr 1, see *RefState boundaries*) and *Power*
(/60,1/)	Dirichlet	*Power control* for $\Phi = A\cos(\omega t)$ (1: use RefState Nbr 1, see *RefState boundaries*)

RefState boundaries

For each boundary of type *5* (reference state boundary *RefState*), e.g., by setting the boundary in the *parameter.ini* file

```
BoundaryName = BC_WALL ! BC name in the mesh.h5 file
BoundaryType = (/5,1/) ! (/ Type, State/)
```

the corresponding *RefState* number must also be supplied in the parameter.ini file (here 1) and is selected from its position in the parameter file. Each *RefState* is defined in the *parameter.ini* file by supplying a value for the voltage an alternating frequency for the cosine function (a frequency of 0 results in a fixed potential over time) and phase shift

```
RefState = (/-0.18011, 1.0, 0.0/) ! RefState Nbr 1: Voltage, Frequency and Phase shift
```

This yields the three parameters used in the cosine function

$$\Phi(t) = A\cos(2\pi f t + \psi)$$

where *A=-0.18011* is the amplitude, *t* is the time, *f=1* is the frequency and *psi=0* is the phase shift.

Similar to boundary type *5* is type *6*, which simply uses a cosine function that always has the same sign, depending on the amplitude *A*

$$\Phi(t) = \frac{A}{2}(\cos(2\pi f t + \psi) + 1)$$

Linear potential function

A linear function that ramps the electric potential from 0 V to a user-defined value can be applied to a boundary via

```
BoundaryName = BC_WALL ! BC name in the mesh.h5 file
BoundaryType = (/7,1/) ! 1: 1st LinState
```

Additionally, this specific boundary condition requires a starting position `LinPhiBasePoint` and a direction along which the potential varies `LinPhiNormal`. The distance along which the potential varies as well as the final value are defined by `LinPhiHeight` and `LinPhi`, respectively. Coordinates below and above this distance are simply set to 0 V and the defined value, respectively. The example below creates a linear ramp from 0 V to 1000 V starting at 1 mm in z-direction and ramps the value over 10 mm in the same direction.

```
LinPhiBasePoint = (/0. , 0. , 1e-3/)
LinPhiNormal    = (/0. , 0. , 1.0/)
LinPhiHeight    = 10e-3
LinPhi          = 1000.
```

The linear potential uses the same functionality as RefState, hence, when two different functions are to be defined use the following example

```
BoundaryName    = BC_right
BoundaryType    = (/7,1/)          ! 7: Dirichlet with linear ramp 1st LinState
LinPhiBasePoint = (/0. , 0. , 0./) ! 1st LinState
LinPhiNormal    = (/1. , 0. , 0./) ! 1st LinState
LinPhiHeight    = 1.0              ! 1st LinState
LinPhi          = 1e3              ! 1st LinState

BoundaryName    = BC_left
BoundaryType    = (/7,2/)          ! 7: Dirichlet with linear ramp 2nd LinState
LinPhiBasePoint = (/0. , 0. , 0./) ! 2nd LinState
LinPhiNormal    = (/1. , 0. , 0./) ! 2nd LinState
LinPhiHeight    = 1.0              ! 2nd LinState
LinPhi          = 0.0              ! 2nd LinState
```

Floating boundary condition (FPC)

A floating boundary condition (FPC) can be used to model a perfect electric conducting surface. The surface can carry a charge Q, which might change over time. however, the requirement is that the surface yields a closed surface integral in 3D (or 2D with periodic/symmetric boundaries in the 3rd dimension). One or more FPCs can be set via

```
BoundaryName = BC_FPC_1 ! BC name in the mesh.h5 file
BoundaryType = (/20,1/) ! 20: activate FPC, 1: Index of the FPC group to which this BC␣
↪belongs (1st group)

BoundaryName = BC_FPC_2 ! BC name in the mesh.h5 file
BoundaryType = (/20,2/) ! 20: activate FPC, 2: Index of the FPC group to which this BC␣
↪belongs (2nd group)
```

For this boundary condition, the charge assigned to each FPC and the resulting potential are written to *FieldAnalyze.csv* automatically, e.g., "007-FPC-Charge-BCState-001" and "008-FPC-Voltage-BCState-001", where *BCState* corresponds to the ID of the FPC. If the particle boundary condition is set to *open* (or *species-swap*), then each impacting charged particle that is removed there, will be added to the accumulated charge on that FPC.

Electric potential condition (EPC)

Grounded surfaces with a specific resistance between the ground and the surface can be modelled as equipotential surfaces, where charged particles are removed and their charge is accumulated over each time step and a voltage is calculated from from the resistance of the surface and the resulting electric current via

$$U = RI = -R\frac{dQ}{dt}$$

where U is the voltage different to ground (0V), R is the resistance assigned to the surface, I is the electric current and dQ the amount of charge that is removed in each time step dt. The boundary is activated by setting one or more EPC

```
BoundaryName = BC_EPC_1 ! BC name in the mesh.h5 file
BoundaryType = (/8,1/)  ! 8: activate EPC, 1: Index of the EPC group to which this BC␣
↪belongs (1st group)

BoundaryName = BC_EPC_2 ! BC name in the mesh.h5 file
BoundaryType = (/8,2/)  ! 8: activate EPC, 2: Index of the EPC group to which this BC␣
↪belongs (2nd group)
```

The resulting current I and voltage U are automatically written to *FieldAnalyze.csv*, e.g., "007-EPC-Current-BCState-001" and "008-EPC-Voltage-BCState-001".

Power control

An automatic adjustment of a *DC* electric potential to ensure that a fixed power absorbed by the charged particles of the system is achieved, requires the following parameters

```
BoundaryName = BC_WALL ! BC name in the mesh.h5 file
BoundaryType = (/2,2/) ! all BCs with this type will be adjusted to the same electric␣
↪potential that is adjusted over time
```

For an *AC* boundary the parameters are the following

```
BoundaryName = BC_WALL  ! BC name in the mesh.h5 file
BoundaryType = (/60,1/) ! applicable to one BC with this BCType. The BCState=1␣
↪corresponds to the RefState number
```

Additionally, a starting value for the potential Φ (or amplitude A in the case of *AC*), lower and upper boundaries and a relaxation factor are required as well as the target input power, which is set via

```
CoupledPowerPotential = (/10. , 1000. , 2000./) ! lower, starting and maximum values for␣
↪the electric potential
CoupledPowerRelaxFac  = 0.05  ! the new potential is updated by 5% in each time step
CoupledPowerTarget    = 1e-10 ! target power of 1e-10 Watt
```

The values in `CoupledPowerPotential` correspond to the lower boundary, the starting value and the upper boundary, respectively. When a simulation is restarted from a state file, the last known value of the BC will be used instead of the starting value, which is only applied when starting a fresh simulation from scratch. Note that in the case of an AC boundary, the amplitude defined in the `RefState` will be overwritten with the initial value given in `CoupledPowerPotential`.

There are three models implemented by which the power control adjusts the electric potential (or peak potential in case of AC)

```
CoupledPowerMode = 3 ! 1: instantaneous power (default value), 2: moving average, 3:␣
↪integrated and averaged over 1 cycle
CoupledPowerFrequency = 13.56e6 ! Frequency with which the coupled power voltage is␣
↪adapte
```

where `CoupledPowerMode` selects the model and `CoupledPowerFrequency` defines the adaption frequency and corresponding period over which the power is integrated but only when model 3 is used. Model 1 uses the instantaneously determined absorbed power that inherently oscillates, model 2 uses an averaged value (that is reset during every load balance or normal restart event) and model 3 integrated the power linearly between particle analysis steps and adjusts the power after each period of the user-defined frequency `CoupledPowerFrequency`.

Zero potential enforcement

It is important to note that when no Dirichlet boundary conditions are selected by the user, the code automatically enforces mixed boundaries on either Neumann or periodic boundaries. Depending on the simulation domain, the direction with the largest extent is selected and on those boundaries an additional Dirichlet boundary condition with $\phi = 0$ is enforced to ensure convergence of the HDG solver. The boundary conditions selected by the user are only altered at these locations and not removed. The information regarding the direction that is selected for this purpose is printed to std.out with the following line

```
 |      Zero potential side activated in direction (1: x, 2: y, 3: z) |        1 | ␣
↪OUTPUT |
```

To selected the direction by hand, simply supply the desired direction via

```
HDGZeroPotentialDir = 1
```

with 1: x-, 2: y-, 3: z-direction.

Bias Voltage DC

A bias voltage develops if charged particles impact on an electrode that is connected via capacitor to a matching box. Hence, if there is an overhead of electrons impacting on the electrode, a negative bias voltage arises, which counters the flow of electrons to that boundary. This feature only works when `PARTICLES=ON` is enabled and Poisson's equation is solved `PICLAS_EQNSYSNAME=poisson`. The following parameters are required

```
UseBiasVoltage              = T       ! Activate bias voltage model
BiasVoltage-NPartBoundaries = 2       ! Nbr of particle boundaries where the ion excess␣
↪is measured
Biasvoltage-PartBoundaries  = (/1,2/) ! Particle boundary index of where the ion excess␣
↪is measured
BiasVoltage-Frequency       = 3e9     ! Frequency with which the bias voltage is adapted
BiasVoltage-Delta           = 1.0     ! Voltage difference used for adapting the bias␣
↪voltage
```

where `UseBiasVoltage` switches the modell on/off, `BiasVoltage-NPartBoundaries` defines the number of boundaries over which the total electric charge is summarized, `Biasvoltage-PartBoundaries` specifies the indices of the particle boundaries where the total electric charge is summarized, e.g., the powered and the grounded electrode., `BiasVoltage-Frequency` is the frequency with which the bias voltage is updated by the voltage difference `BiasVoltage-Delta`.

Additionally, the field boundary must be defined

```
BoundaryName = BC_left  ! BC name in the mesh.h5 file
BoundaryType = (/50,0/) ! Dirichlet with 0V initial BC
```

where the value *50* is used for for pure DC boundaries. Furthermore, the bias voltage model relies on the functionality of the integral particle flux measurement on the corresponding boundaries, see *BoundaryParticleOutput (BPO)* in Section *Surface Variables*. The definition of BPO variables must include at least the ones above, e.g.

```
CalcBoundaryParticleOutput  = T
BPO-NPartBoundaries         = 2
BPO-PartBoundaries          = (/1,2/)
BPO-NSpecies                = 3
BPO-Species                 = (/2,3,4/)
```

where the defined species information must consider all relevant charged particle species that affect the bias voltage.

Bias Voltage AC

If an AC potential boundary is coupled with a bias potential, the `BoundaryType` has to be changed as compared with the previous section

```
BoundaryName = BC_left  ! BC name in the mesh.h5 file
BoundaryType = (/51,1/) ! Dirichlet with 0V initial BC
```

Furthermore, a RefState must be defined, which specifies the parameters for the $cos(\omega t)$ function, for details, see *RefState boundaries*. Note that this BC is only implemented with zero crossing.

Bias Voltage AC and Fixed coupled power

If an **AC potential boundary** is coupled with a **bias potential** and an **automatic adjustment of the AC amplitude** to ensure that a fixed power to the system is achieved, the `BoundaryType` has to be changed as compared with the previous section

```
BoundaryName = BC_left  ! BC name in the mesh.h5 file
BoundaryType = (/52,1/) ! Dirichlet with 0V initial BC
```

where a RefState must be defined, which specifies the parameters for the $cos(\omega t)$ function, for details, see *RefState boundaries*. Note that this BC is only implemented with zero crossing. For details on the power coupling, see *Power control*.

Dielectric Materials

Dielectric material properties can be considered by defining regions (or specific elements) in the computational domain, where permittivity and permeability constants for linear isotropic non-lossy dielectrics are used. The interfaces between dielectrics and vacuum regions must be separated by element-element interfaces due to the DGSEM (Maxwell) and HDG (Poisson) solver requirements, but can vary spatially within these elements.

The dielectric module is activated by setting

```
DoDielectric = T
```

and specifying values for the permittivity ε_R and permeability μ_R constants

```
DielectricEpsR = X
DielectricMuR = X
```

Furthermore, the corresponding regions in which the dielectric materials are found must be defined, e.g., simple boxes via

```
xyzDielectricMinMax  = (/0.0 , 1.0 , 0.0 , 1.0 , 0.0 , 1.0/)
```

for the actual dielectric region (vector with 6 entries yielding x-min/max, y-min/max and z-min/max) or the inverse (vacuum, define all elements which are NOT dielectric) by

```
xyzPhysicalMinMaxDielectric = (/0.0 , 1.0 , 0.0 , 1.0 , 0.0 , 1.0/)
```

Spherical regions can be defined by setting a radius value

```
DielectricRadiusValue = X
```

and special pre-defined regions (which also consider spatially varying material properties) may also be used, e.g.,

```
DielectricTestCase = FishEyeLens
```

where the following pre-defined cases are available as given in table Table 1.5

Table 1.5: Dielectric Test Cases

Option	Additional Parameters	Notes
`FishEyeLens`	none	function with radial dependence: $\varepsilon_r = n_0^2/(1+(r/r_{max})^2)^2$
`Circle`	`DielectricRadiusVal`	Circular dielectric in x-y-direction (constant in z-direction)
	`DielectricRadiusVal`	with optional cut-out radius DielectricRadiusValueB along the
	`DielectricCircleAxi`	axis given by DielectricCircleAxis
`DielectricResonatorAnt`	`DielectricRadiusVal`	Circular dielectric in x-y-direction (only elements with $z > 0$)
`FH_lens`	none	specific geometry (`SUBROUTINE SetGeometry` yields more information)

For the Maxwell solver (DGSEM), the interface fluxes between vacuum and dielectric regions can either be conserving or non-conserving, which is selected by

```
DielectricFluxNonConserving = T
```

which uses non-conserving fluxes. This is recommended for improved simulation results, as described in [10]. When particles are to be considered in a simulation, these are generally removed from dielectric materials during the emission (inserting) stage, but may be allowed to exist within dielectrics by setting

```
DielectricNoParticles = F
```

which is set true by default, hence, removing the particles.

Dielectric Zones

Regions or zones (corresponding to zones as defined by hopr) can also be used to define dielectrics. In this case, the number of zones must be supplied

```
DielectricNbrOfZones = 8
```

and a vector of the same size for the corresponding zone IDs, ε_R and μ_R for each zone, respectively

```
DielectricZoneID     = (/9   , 10  , 11  , 12  , 13  , 14  , 15  , 16/)
DielectricZoneEpsR   = (/2.0 , 2.0 , 2.0 , 2.0 , 2.0 , 2.0 , 2.0 , 2.0/)
DielectricZoneMuR    = (/1.0 , 1.0 , 1.0 , 1.0 , 1.0 , 1.0 , 1.0 , 1.0/)
```

Note that this method cannot be used in combination with any other way of defining dielectrics.

1.4.4 Boundary Conditions - Particle Solver

Within the parameter file it is possible to define different particle boundary conditions. The number of boundaries is defined by

```
Part-nBounds = 2
Part-Boundary1-SourceName   = BC_OPEN
Part-Boundary1-Condition    = open
Part-Boundary2-SourceName   = BC_WALL
Part-Boundary2-Condition    = reflective
```

The `Part-Boundary1-SourceName=` corresponds to the name given during the preprocessing step with HOPR. The available conditions (`Part-Boundary1-Condition=`) are described in the table below.

Condition	Description
`open`	Every particle crossing the boundary will be deleted
`symmetric`	A perfect specular reflection, without sampling of particle impacts
`reflective`	Definition of different surface models: Section *Reflective Wall*, Section *Surface Chemistry*, Section *Catalytic Surfaces*
`rot_periodic`	Definition of rotational periodicity: Section *Rotational Periodicity*
`rot_periodic_inter_`	Extension of rotational periodicity, allowing non-conformal interfaces and varying periodicity

Reflective Wall

A reflective boundary can be defined with

```
Part-Boundary2-SourceName   = BC_WALL
Part-Boundary2-Condition    = reflective
```

A perfect specular reflection is performed, if no other parameters are given. Gas-surface interactions can be modelled with the extended Maxwellian model [11], using accommodation coefficients of the form

$$\alpha = \frac{E_i - E_r}{E_i - E_w}$$

where i, r and w denote the incident, reflected and wall energy, respectively. The coefficient `MomentumACC` is utilized to decide whether a diffuse (`MomentumACC` $> R$) or specular reflection (`MomentumACC` $< R$) occurs upon particle

impact, where $R = [0, 1)$ is a random number. Separate accommodation coefficients can be defined for the translation (`TransACC`), rotational (`RotACC`), vibrational (`VibACC`) and electronic energy (`ElecACC`) accommodation at a constant wall temperature [K].

```
Part-Boundary2-MomentumACC  = 1.
Part-Boundary2-WallTemp     = 300.
Part-Boundary2-TransACC     = 1.
Part-Boundary2-VibACC       = 1.
Part-Boundary2-RotACC       = 1.
Part-Boundary2-ElecACC      = 1.
```

An additional option `Part-Boundary2-SurfaceModel` is available, that is used for heterogeneous reactions (reactions that have reactants in two or more phases) or secondary electron emission models. These models are described in detail in Section *Surface Chemistry*.

Wall movement (Linear & rotational)

Additionally, a linear wall velocity [m/s] can be given

```
Part-Boundary2-WallVelo = (/0,0,100/)
```

In the case of rotating walls the `-RotVelo` flag, a rotation frequency [Hz], and the rotation axis (x=1, y=2, z=3) must be set. Note that the definition of the rotational direction is defined by the sign of the frequency using the right-hand rule.

```
Part-Boundary2-RotVelo = T
Part-Boundary2-RotFreq = 100
Part-Boundary2-RotAxis = 3
```

The wall velocity will then be superimposed onto the particle velocity.

Linear temperature gradient

A linear temperature gradient across a boundary can be defined by supplying a second wall temperature and the start and end vector as well as an optional direction to which the gradient shall be limited (default: 0, x = 1, y = 2, z = 3)

```
Part-Boundary2-WallTemp2      = 500.
Part-Boundary2-TempGradStart  = (/0.,0.,0./)
Part-Boundary2-TempGradEnd    = (/1.,0.,1./)
Part-Boundary2-TempGradDir    = 0
```

In the default case of the `TempGradDir = 0`, the temperature will be interpolated between the start and end vector, where the start vector corresponds to the first wall temperature `WallTemp`, and the end vector to the second wall temperature `WallTemp2`. Position values (which are projected onto the temperature gradient vector) beyond the gradient vector utilize the first (Start) and second temperature (End) as the constant wall temperature, respectively. In the special case of `TempGradDir = 1/2/3`, the temperature gradient will only be applied along the chosen the direction. As oppposed to the default case, the positions of the surfaces are not projected onto the gradient vector before checking wether they are inside the box spanned by `TempGradStart` and `TempGradEnd`. The applied surface temperature is output in the `DSMCSurfState` as `Wall_Temperature` for verification.

Radiative equilibrium

Another option is to adapt the wall temperature based on the heat flux assuming that the wall is in radiative equilibrium. The temperature is then calculated from

$$q_w = \varepsilon \sigma T_w^4,$$

where ε is the radiative emissivity of the wall (default = 1) and $\sigma = 5.67E-8 W m^{-2} K^{-4}$ is the Stefan-Boltzmann constant. The adaptive boundary is enabled by

```
Part-AdaptWallTemp = T
Part-Boundary1-UseAdaptedWallTemp = T
Part-Boundary1-RadiativeEmissivity = 0.8
```

If provided, the wall temperature will be adapted during the next output of macroscopic variables, where the heat flux calculated during the preceding sampling period is utilized to determine the side-local temperature. The temperature is included in the `State` file and thus available during a restart of the simulation. The surface output (in `DSMCSurfState`) will additionally include the temperature distribution in the `Wall_Temperature` variable (see Section *Particle Flow and Surface Sampling*). To continue the simulation without further adapting the temperature, the first flag has to be disabled (`Part-AdaptWallTemp = F`). It should be noted that the the adaptation should be performed multiple times to achieve a converged temperature distribution.

Rotational Periodicity

The rotational periodic boundary condition can be used in order to reduce the computational effort in case of an existing rotational periodicity. In contrast to symmetric boundary conditions, a macroscopic flow velocity in azimuthal direction can be simulated (e.g. circular flow around a rotating cylinder). Exactly two corresponding boundaries must be defined by setting `rot_periodic` as the BC condition and the rotating angle for each BC. Multiple pairs of boundary conditions with different angles can be defined.

```
Part-Boundary1-SourceName       = BC_Rot_Peri_plus
Part-Boundary1-Condition        = rot_periodic
Part-Boundary1-RotPeriodicAngle = 90.

Part-Boundary2-SourceName       = BC_Rot_Peri_minus
Part-Boundary2-Condition        = rot_periodic
Part-Boundary2-RotPeriodicAngle = -90.
```

CAUTION! The correct sign for the rotating angle must be determined. The position of particles that cross one rotational periodic boundary is tranformed according to this angle, which is defined by the right-hand rule and the rotation axis:

```
Part-RotPeriodicAxi = 1    ! (x = 1, y = 2, z = 3)
```

The usage of rotational periodic boundary conditions is limited to cases, where the rotational periodic axis is one of the three Cartesian coordinate axis (x, y, z) with its origin at (0, 0, 0).

Intermediate Plane Definition

If several segments with different rotation angles are defined, exactly two corresponding BCs must be defined for each segment. Since the plane between these segments with different rotational symmetry angles represents a non-conforming connection, additional two BCs must be defined as `rot_periodic_inter_plane` at this intermediate plane. Both BCs must refer to each other in the definition in order to ensure the connection.

```
Part-Boundary40-SourceName       = BC_INT_R1_BOT
Part-Boundary40-Condition        = rot_periodic_inter_plane
Part-Boundary40-AssociatedPlane  = 41

Part-Boundary41-SourceName       = BC_INT_S1_TOP
Part-Boundary41-Condition        = rot_periodic_inter_plane
Part-Boundary41-AssociatedPlane  = 40
```

Note that using the intermediate plane definition with two corresponding BCs allows the user to mesh the segments independently, creating a non-conforming interface at the intermediate plane. However, use of these non-conformal grids is so far only possible in standalone DSMC simulations.

Porous Wall / Pump

The porous boundary condition uses a removal probability to determine whether a particle is deleted or reflected at the boundary. The main application of the implemented condition is to model a pump, according to [12]. It is defined by giving the number of porous boundaries and the respective boundary number (`BC=2` corresponds to the `BC_WALL` boundary defined in the previous section) on which the porous condition is.

```
Surf-nPorousBC=1
Surf-PorousBC1-BC=2
Surf-PorousBC1-Type=pump
Surf-PorousBC1-Pressure=5.
Surf-PorousBC1-PumpingSpeed=2e-9
Surf-PorousBC1-DeltaPumpingSpeed-Kp=0.1
Surf-PorousBC1-DeltaPumpingSpeed-Ki=0.0
```

Currently, two porous BC types are available, `pump` and `sensor`. For the former, the removal probability is determined through the given pressure [Pa] at the boundary. A pumping speed can be given as a first guess, however, the pumping speed S $[m^3/s]$ will be adapted if the proportional factor (K_p, `DeltaPumpingSpeed-Kp`) is greater than zero

$$S^{n+1}(t) = S^n(t) + K_\mathrm{p}\Delta p(t) + K_\mathrm{i}\int_0^t \Delta p(t')dt',$$

where Δp is the pressure difference between the given pressure and the actual pressure at the pump. An integral factor (K_i, `DeltaPumpingSpeed-Ki`) can be utilized to mimic a PI controller. The proportional and integral factors are relative to the given pressure. However, the integral factor has not yet been thoroughly tested. The removal probability α is then calculated by

$$\alpha = \frac{Sn\Delta t}{N_\mathrm{pump}w}$$

where n is the sampled, cell-local number density and N_pump is the total number of impinged particle at the pump during the previous time step. Δt is the time step and w the weighting factor. The pumping speed S is only adapted if the resulting removal probability α is between zero and unity. The removal probability is not species-specific.

To reduce the influence of statistical fluctuations, the relevant macroscopic values (pressure difference Δp and number density n) can be sampled for N iterations by defining (for all porous boundaries)

```
AdaptiveBC-SamplingIteration=10
```

A porous region on the specified boundary can be defined. At the moment, only the `circular` option is implemented. The origin of the circle/ring on the surface and the radius have to be given. In the case of a ring, a maximal and minimal radius is required (`-rmax` and `-rmin`, respectively), whereas for a circle only the input of maximal radius is sufficient.

```
Surf-PorousBC1-Region=circular
Surf-PorousBC1-normalDir=1
Surf-PorousBC1-origin=(/5e-6,5e-6/)
Surf-PorousBC1-rmax=2.5e-6
```

The absolute coordinates are defined as follows for the respective normal direction.

Normal Direction	Coordinates
x (=1)	(y,z)
y (=2)	(z,x)
z (=3)	(x,y)

Using the regions, multiple pumps can be defined on a single boundary. Additionally, the BC can be used as a sensor by defining the respective type:

```
Surf-PorousBC1-BC=3
Surf-PorousBC1-Type=sensor
```

Together with a region definition, a pump as well as a sensor can be defined on a single and/or multiple boundaries, allowing e.g. to determine the pressure difference between the pump and a remote area of interest.

Surface Chemistry

Modelling of reactive surfaces is enabled by setting `Part-BoundaryX-Condition=reflective` and an appropriate particle boundary surface model `Part-BoundaryX-SurfaceModel`:

```
Part-Boundary1-SurfaceModel = 0
```

The available conditions (`Part-BoundaryX-SurfaceModel=`) are described in the table below, ranging from simple empirical models and secondary electron/ion emission to finite-rate catalysis modelling including a surface treatment.

Model	Description
0 (default)	Standard extended Maxwellian scattering
1	Empirical modelling of sticking coefficient/probability
2	Fixed probability surface chemistry
5	Secondary electron emission as given by Ref. [13].
7	Secondary electron emission due to ion impact (SEE-I with Ar^+ on different metals) as used in Ref. [14] and given by Ref. [15] with a default yield of 13 %.
8	Secondary electron emission due to ion impact (SEE-E with e^- on dielectric surfaces) as used in Ref. [16] and given by Ref. [17].
9	Secondary electron emission due to ion impact (SEE-I with Ar^+) with a constant yield of 1 %. Emitted electrons have an energy of 6.8 eV upon emission.
10	Secondary electron emission due to ion impact (SEE-I with Ar^+ on copper) as used in Ref. [18] originating from [19]
11	Secondary electron emission due to electron impact (SEE-E with e^- on quartz (SiO_2)) as described in Ref. [20] originating from [21]
20	Finite-rate catalysis model, Section *Catalytic Surfaces*

For surface sampling output, where the surface is split into, e.g., 3×3 sub-surfaces, the following parameters mus be set

```
BezierSampleN                  = 3
DSMC-nSurfSample               = 3
Part-WriteMacroSurfaceValues   = T
Particles-DSMC-CalcSurfaceVal = T
Part-IterationForMacroVal      = 200
```

where `BezierSampleN=DSMC-nSurfSample`. In this example, sampling is performed over and every 200 iterations.

Empirical model for a sticking coefficient

To model the sticking of gas particles on cold surfaces, an empirical model is available, which is based on experimental measurements. The sticking coefficient is modelled through the product of a non-bounce probability $B(\alpha)$ and a condensation probability $C(\alpha, T)$

$$p_s(\alpha, T) = B(\alpha)C(\alpha, T)$$

The non-bounce probability introduces a linear dependency on the impact angle α

$$B(\alpha) = \begin{cases} 1, & |\alpha| < \alpha_\mathrm{B} \\ \dfrac{90° - |\alpha|}{90° - |\alpha_\mathrm{B}|}, & \alpha_\mathrm{B} \leq |\alpha| \leq 90° \end{cases}$$

α_B is a model-dependent cut-off angle. The condensation probability introduces a linear dependency on the surface temperature T

$$C(\alpha, T) = \begin{cases} 1, & T < T_1 \\ \dfrac{T_2(\alpha) - T}{T_2(\alpha) - T_1(\alpha)}, & T_1 \leq T \leq T_2 \\ 0, & T > T_2 \end{cases}$$

The temperature limits T_1 and T_2 are model parameters and can be given for different impact angle ranges defined by the maximum impact angle α_max. These model parameters are read-in through the species database and have to be provided in the `/Surface-Chemistry/StickingCoefficient` dataset in the following format (example values):

$\alpha_{\max}$ [deg]	α_{B} [deg]	T_1 [K]	T_2 [K]
45	80	50	100
90	70	20	50

In this example, within impact angles of $0 \leq \alpha \leq 45$, the model parameters of the first row will be used and for $45 < \alpha \leq 90$ the second row. The number of rows is not limited. The species database is read-in by

```
Particles-Species-Database = SpeciesDatabase.h5
```

As additional output, the cell-local sticking coefficient will be added to the sampled surface output. A particle sticking to the surface will be deleted and its energy added to the heat flux sampling. This model can be combined with the linear temperature gradient and radiative equilibrium modelling as described in Section *Reflective Wall*.

Fixed probability surface chemistry

This simple fixed-probability surface chemistry model allows the user to define arbitrary surface reactions, by defining the impacting species, the products and a fixed event probability. The reaction is then assigned to the boundaries by specifying their number and index as defined previously.

```
Surface-Reaction1-Type               = P
Surface-Reaction1-Reactants          = (/1,0/)
Surface-Reaction1-Products           = (/2,1,0/)
Surface-Reaction1-EventProbability   = 0.25
Surface-Reaction1-NumOfBoundaries    = 2
Surface-Reaction1-Boundaries         = (/1,3/)
```

Optionally, a reaction-specific accommodation coefficient for the products can be defined, otherwise the surface-specific accommodation will be utilized for the product species:

```
Surface-Reaction1-ProductAccommodation = 0.
```

In the case that the defined event does not occur, a regular interaction using the surface-specific accommodation coefficients is performed. Examples are provided as part of the regression tests: `regressioncheck/NIG_DSMC/SURF_PROB_DifferentProbs` and `regressioncheck/NIG_DSMC/SURF_PROB_MultiReac`.

Secondary Electron Emission (SEE)

Different models are implemented for secondary electron emission that are based on either electron or ion bombardment, depending on the surface material. All models require the specification of the electron species that is emitted from the surface via

```
Part-SpeciesA-PartBoundB-ResultSpec = C
```

where electrons of species `C` are emitted from boundary `B` on the impact of species `A`.

Model 5

The model by Levko [13] can be applied for copper electrodes for electron and ion bombardment and is activated via `Part-BoundaryX-SurfaceModel=5`. For ions, a fixed emission yield of 0.02 is used and for electrons an energy-dependent function is employed.

Model 7

The model by Depla [15] can be used for various metal surfaces and features a default emission yield of 13 % and is activated via `Part-BoundaryX-SurfaceModel=7` and is intended for the impact of Ar^+ ions. For more details, see the original publication.

The emission yield and energy can be varied for this model by setting

```
SurfModEmissionYield  = 1.45 ! ratio of emitted electron flux vs. impacting ion flux [-]
SurfModEmissionEnergy = 6.8  ! [eV]
```

respectively. The emission yield represents the ratio of emitted electrons vs. impacting ions and the emission energy is given in electronvolts. If the energy is not set, the emitted electron will have the same velocity as the impacting ion.

Additionally, a uniform energy distribution function for the emitted electrons can be set via

```
SurfModEnergyDistribution = uniform-energy
```

which will scale the energy of the emitted electron to fit a uniform distribution function.

Model 8

The model by Morozov [17] can be applied for dielectric surfaces and is activated via `Part-BoundaryX-SurfaceModel=8` and has an additional parameter for setting the reference electron temperature (see model for details) via `Part-SurfaceModel-SEE-Te`, which takes the electron temperature in Kelvin as input (default is 50 eV, which corresponds to 11604 K). The emission yield is determined from an energy-dependent function. The model can be switched to an automatic determination of the bulk electron temperature via

```
Part-SurfaceModel-SEE-Te-automatic = T ! Activate automatic bulk temperature calculation
Part-SurfaceModel-SEE-Te-Spec      = 2 ! Species ID used for automatic temperature␣
↪calculation (must correspond to electrons)
```

where the species ID must be supplied, which corresponds to the electron species for which, during `Part-AnalyzeStep`, the global translational temperature is determined and subsequently used to adjust the energy dependence of the SEE model. The global (bulk) electron temperature is written to *PartAnalyze.csv* as *XXX-BulkElectronTemp-[K]*.

Model 10

An energy-dependent model of secondary electron emission due to Ar^+ ion impact on a copper cathode as used in Ref. [18] originating from [19] is activated via `Part-BoundaryX-SurfaceModel=10`. For more details, see the original publications.

Model 11

An energy-dependent model (linear and power fit of measured SEE yields) of secondary electron emission due to e^- impact on a quartz (SiO_2) surface as described in Ref. [20] originating from [21] is activated via `Part-BoundaryX-SurfaceModel=11`. For more details, see the original publications.

Catalytic Surfaces

Catalytic reactions can be modeled in PICLas using a finite-rate reaction model with an implicit treatment of the reactive surface. For a better resolution of the parameters, the catalytic boundaries are discretized into a certain number of subsides. A definition of the boundary temperature in the parameter input file is required in all cases. Different types of surfaces can be defined by the lattice constant of the unit cell `Part-BoundaryX-LatticeVec` and the number of particles in the unit cell `Part-BoundaryX-NbrOfMol-UnitCell`. These parameters are used in the calculation of the number of active sites.

By default, the simulation is started with a clean surface, but an initial species-specific coverage can be specified by `Part-BoundaryX-SpeciesX-Coverage`, which represents the relative number of active sites that are occupied by adsorbate particles. Maximum values for the coverage values can be specified by:

```
Part-Boundary1-Species1-MaxCoverage
Part-Boundary1-MaxTotalCoverage
```

Multi-layer adsorption is enabled by a maximal total coverage greater than 1.

The reaction paths are defined in the input parameter file. First, the number of gas-surface reactions to be read in must be defined:

```
Surface-NumOfReactions = 2
```

A catalytic reaction and the boundary on which it takes place is then defined by

```
Surface-Reaction1-SurfName          = Adsorption
Surface-Reaction1-Type              = A
Surface-Reaction1-Reactants         = (/1,0/)
Surface-Reaction1-Products          = (/2,1,0/)
Surface-Reaction1-NumOfBoundaries   = 2
Surface-Reaction1-Boundaries        = (/1,3/)
```

All reactants and products are defined by their respective species index. In the case of multiple reacting, the order does not influence the input. The following options are available for the catalytic reaction type:

Model	Description
A	Adsorption: Kisliuk or Langmuir model
D	Desorption: Polanyi-Wigner model
ER	Eley-Rideal reaction: Arrhenius based chemistry
LH	Langmuir-Hinshelwood reaction: Arrhenius based chemistry
LHD	Langmuir-Hisnhelwood reaction with instantaneous desorption

For the treatment of multiple reaction paths of the same species, a possible bias in the reaction rate is avoided by a randomized treatment. Bulk species can participate in the reaction. In this case, the bulk species is defined by `Surface-Species` and the corresponding species index. All reaction types allow for the definition of a reaction enthalpy. In addition, this value can be linearly increased (negative factor) or decreased (positive factor) by a scaling factor for the heat of reaction. Both values are given in [K].

```
Surface-Reaction1-ReactHeat      = 17101.4
Surface-Reaction1-HeatScaling    = 1202.9
```

Depending on the reaction type, different additional parameters have to be defined. More details on the specific cases are given in the following subsections. An example input file for CO and O2 on a palladium surface can be found in the regression tests `regressioncheck/WEK_DSMC/ChannelFlow_SurfChem_AdsorpDesorp_CO_O2`.

Adsorption

For the modelling of the adsorption of a gas particle on the surface, two models are available: the simple Langmuir model, with a linear dependence of the adsorption probability on the surface coverage, and the precursor-based Kisliuk model:

$$S = S_0(1 + K(1/\theta^{\alpha} - 1))^{-1}$$

Here, S_0 is the binding coefficient for a clean surface, α is the dissociation constant (2 for dissociative adsorption) and K is the equilibrium constant between adsorption and desorption from the precursor state. For $K = 1$, the model simplifies to the Langmuir case. The parameters can be defined in PICLas as follows:

```
Surface-Reaction1-StickingCoefficient  = 0.2
Surface-Reaction1-DissOrder            = 1
Surface-Reaction1-EqConstant           = 0.6
```

A special case of adsorption is the dissociative adsorption (`Surface-ReactionX-DissociativeAdsorption = true`), where only half of the molecule binds to the surface, while the other half remains in the gas phase. The adsorbate half `Surface-ReactionX-AdsorptionProduct` and the gas phase product `Surface-ReactionX-GasPhaseProduct` are specified by their respective species indices. The adsorption probability is calculated analogously to the general case.

Lateral interactions between multiple adsorbate species, which can disfavor further adsorption can be taken into account by the command `Surface-ReactionX-Inhibition` and the species index of the inhibiting species.

Desorption

The desorption of an adsorbate particle into the gas phase is modelled by the Polanyi-Wigner equation.

$$k(T) = AT^{b}\theta_A^{\alpha}e^{-E_a/T}$$

where A is the prefactor ([1/s, m^2/s] depending on the dissociation constant), α the dissociation constant and E_a the activation energy [K]. These parameters can be defined in PICLas as follows:

```
Surface-ReactionX-Prefactor
Surface-ReactionX-Energy
```

Catalytic Reaction

The Eley-Rideal and the Langmuir-Hinshelwood reaction use Arrhenius-type reaction rates along with the coverage of all surface-bound reactants θ_{AB}, to reproduce of the catalytic reaction.

$$k(T) = AT^b\theta_{AB}e^{-E_a/T}$$

The Arrhenius prefactor ([m^3/s] for the Eley-Rideal reaction and [m^2/s] for the Langmuir-Hinshelwood case) and the activation energy are read in analogously to the desorption case. For the reactions, an energy accommodation coefficient `Surface-ReactionX-EnergyAccommodation` with values between 0 and 1 can be specified, which defines the amount of the reaction energy that is transferred to the surface.

In the general Langmuir-Hinshelwood case with the reaction type `LH`, the product species stays adsorbed on the surface, until a desorption takes place in a later step. For reactions in combination with very high desorption rates, the reaction type `LHD` is more fitting. The product species are inserted directly into the gas phase without an intermediate desorption step.

Example inputs for both catalytic reactions can be found in the regression tests: `regressioncheck/NIG_Reservoir/CAT_RATES_ER` and `regressioncheck/NIG_Reservoir/CAT_RATES_LH`.

Diffusion

With `Surface-Diffusion = true` an instantaneous diffusion over all catalytic boundaries is enabled. This is equivalent to an averaging of the coverage values for all surface subsides.

Parameter Read-In from the Species Database

All information about a catalytic reaction can be retrieved from the species database. Here the catalytic reaction parameters are stored in containers and accessed via the reaction name, e.g. `Adsorption_CO_Pt`.

Deposition of Charges on Dielectric Surfaces

Charged particles can be absorbed (or reflected and leave their charge behind) at dielectric surfaces when using the deposition method `cell_volweight_mean`. The boundary can be used by specifying

```
```
Part-Boundary1-Condition   = reflective
Part-Boundary1-Dielectric   = T
Part-Boundary1-NbrOfSpeciesSwaps = 3
Part-Boundary1-SpeciesSwaps1   = (/1,0/) ! e-
Part-Boundary1-SpeciesSwaps2   = (/2,2/) ! Ar
Part-Boundary1-SpeciesSwaps3   = (/3,2/) ! Ar+
```
```

which sets the boundary dielectric and the given species swap parameters effectively remove electrons (e^-) on impact, reflect Ar atoms and neutralize Ar^+ ions by swapping these to Ar atoms. Note that currently only singly charged particles can be handled this way. When multiple charged particles would be swapped, their complete charge mus be deposited at the moment.

The boundary must also be specified as an *inner* boundary via

```
BoundaryName                     = BC_INNER
BoundaryType                     = (/100,0/)
```

or directly in the *hopr.ini* file that is used for creating the mesh.

1.4.5 Particle Initialization & Emission

The RAM to store the particles is dynamically allocated. However, it is possible to restrict the number of particles per MPI process by setting

```
Part-MaxParticleNumber=1000000
```

New memory is allocated in separate chunks because allocating memory for the particle data and copying it to the new memory area is expensive. The chunksize is relative to the particles used and can be set with

```
Part-MaxPartNumIncrease=0.1
```

A higher value increases the amount of unnecessary RAM allocated to particles, while a lower value increases the number of memory adjustment operations. The optimal trade-off depends on the simulation and the machine, but it only affects the performance of the simulations, not the quality of the results.

The following section gives an overview of the available options regarding the definition of species and particle initialization and emission. Simulation particles can be inserted initially within the computational domain and/or emitted at every time step. First of all, the number of species is defined by

```
Part-nSpecies=1
```

Regardless whether a standalone PIC, DSMC, or a coupled simulation is performed, the atomic mass [kg], the charge [C] and the weighting factor w [-], sometimes referred to as macro-particle factor (MPF), are required for each species.

```
Part-Species1-MassIC=5.31352E-26
Part-Species1-ChargeIC=0.0
Part-Species1-MacroParticleFactor=5E2
```

Species that are not part of the initialization or emission but might occur as a result of e.g. chemical reactions should also be defined with these parameters.

Due to the often repetitive definitions, the default value for a given parameter can be set using the wildcard `$`. Different values for individual parameters can be specified by explicitly specifying the numbered parameter, irrespective of the ordering in the parameter file.

```
Part-Species1-Init1-VeloIC = 1.
Part-Species$-Init$-VeloIC = 2.
```

Due to runtime considerations, the evaluation of the wildcard character is performed from left to right. Thus, a parameter like `Part-Species1-Init$-VeloIC` will not work.

Different velocity distributions are available for the initialization/emission of particles.

Distribution	Description
maxwell	Maxwell-Boltzmann distribution
maxwell_lpn	Maxwell-Boltzmann distribution for low particle numbers
WIP	**WORK IN PROGRESS**

Some emission types allow the usage of an emission-specific particle weighting factor. The default weighting factor given by `Part-Species1-MacroParticleFactor` can be overwritten by supplying a different one for each initialization, for which the variable weighting factor (or variable macro-particle factor `vMPF`) model must be activated

```
Part-vMPF = T
Part-Species1-Init1-MacroParticleFactor = 1e4
```

Initialization

At the beginning of a simulation, particles can be inserted using different initialization routines. Initialization regions are defined per species and can overlap. First, the number of initialization conditions/regions has to be defined

```
Part-Species1-nInits = 1
```

The type of the region is defined by the following parameter

```
Part-Species1-Init1-SpaceIC = cell_local
```

Different `SpaceIC` are available and an overview is given in the table below.

Distribution	Description	Reference
cell_local	Particles are inserted in every cell at a constant number density	Section *Cell local*
disc	Particles are inserted on a circular disc	Section *Circular Disc*
cuboid	Particles are inserted in the given cuboid volume at a constant number density	Section *Cuboid*
cylinder	Particles are inserted in the given cylinder volume at a constant number density	Section *Cylinder*
sphere	Particles are inserted in the given sphere volume at a constant number density	Section *Sphere*
photon_cylinder	Ionization of a background gas through photon impact (cylinder distribution)	Section *Photo-ionization*
photon_SEE_disc	Secondary electron emission through photon impact (disk distribution)	Section *Photo-ionization*
photon_honeycomb	Ionization of a background gas through photon impact (honeycomb distribution)	Section *Photo-ionization*
photon_SEE_honeycomb	Secondary electron emission through photon impact (honeycomb distribution)	Section *Photo-ionization*
photon_rectangle	Ionization of a background gas through photon impact (rectangular distribution)	Section *Photo-ionization*
photon_SEE_rectangle	Secondary electron emission through photon impact (rectangular distribution)	Section *Photo-ionization*
EmissionDistribution	Initial only ($t = 0$) field-based (n, T, v) particle distribution from .h5	Section *Emission Distribution*

Common parameters required for most of the insertion routines are given below. The drift velocity is defined by the direction vector `VeloVecIC`, which is a unit vector, and a velocity magnitude [m/s]. The thermal velocity of particle is determined based on the defined velocity distribution and the given translation temperature `MWTemperatureIC` [K]. Finally, the 'real' number density is defined by `PartDensity` [1/m^3], from which the actual number of simulation particles will be determined (depending on the chosen weighting factor).

```
Part-Species1-Init1-VeloIC=1500
Part-Species1-Init1-VeloVecIC=(/-1.0,0.0,0.0/)
Part-Species1-Init1-velocityDistribution=maxwell_lpn
Part-Species1-Init1-MWTemperatureIC=300.
Part-Species1-Init1-PartDensity=1E20
```

In the case of molecules, the rotational and vibrational temperature [K] have to be defined. If electronic excitation is considered, the electronic temperature [K] has to be defined

```
Part-Species1-Init1-TempRot=300.
Part-Species1-Init1-TempVib=300.
Part-Species1-Init1-TempElec=300.
```

The parameters given so far are sufficient to define an initialization region for a molecular species using the `cell_local` option. Additional options required for other insertion regions are described in the following.

Cell local

Additional options are available to limit the local emission to certain limits in each dimension (x,y,z) as defined by:

```
Part-Species1-Init1-MinimalLocation       = (/ -1.0, -1.0, -999. /)
Part-Species1-Init1-MaximalLocation       = (/  1.0,  1.0,  999. /)
```

To limit the insertion only to a specific dimension, simply provide a sufficiently large number for the other dimensions. This approach can also be utilized for 2D and axisymmetric simulations.

When using a variable particle weighting as described in Section *Variable Particle Weighting*, the variable `Part-Species1-vMPFSplitThreshold` will be utilized as the target number of particles per cell during the insertion and the weighting factor will be determined from the given number density and cell volume.

Circular Disc

To define the circular disc the following parameters are required:

```
Part-Species1-Init1-SpaceIC               = disc
Part-Species1-Init1-RadiusIC              = 1
Part-Species1-Init1-BasePointIC           = (/ 0.0, 0.0, 0.0 /)
Part-Species1-Init1-BaseVector1IC         = (/ 1.0, 0.0, 0.0 /)
Part-Species1-Init1-BaseVector2IC         = (/ 0.0, 1.0, 0.0 /)
Part-Species1-Init1-NormalIC              = (/ 0.0, 0.0, 1.0 /)
```

The first and second base vector span a plane, where a circle with the given radius will be defined at the base point.

Cuboid

To define the cuboid the following parameters are required:

```
Part-Species1-Init1-SpaceIC               = cuboid
Part-Species1-Init1-RadiusIC              = 1
Part-Species1-Init1-CuboidHeightIC        = 1
Part-Species1-Init1-BasePointIC           = (/ 0.0, 0.0, 0.0 /)
Part-Species1-Init1-BaseVector1IC         = (/ 1.0, 0.0, 0.0 /)
Part-Species1-Init1-BaseVector2IC         = (/ 0.0, 1.0, 0.0 /)
Part-Species1-Init1-NormalIC              = (/ 0.0, 0.0, 1.0 /)
```

The first and second base vector span a side of the cuboid, and then its extruded in the normal direction up to the cuboid height.

For symmetric simulations `Part-Species1-Init1-BaseVector2IC` is set in direction of the symmetry `(/ 0.0, 0.0, 1.0 /)`

Cylinder

To define the cylinder the following parameters are required:

```
Part-Species1-Init1-SpaceIC               = cylinder
Part-Species1-Init1-RadiusIC              = 1
Part-Species1-Init1-CylinderHeightIC      = 1
Part-Species1-Init1-BasePointIC           = (/ 0.0, 0.0, 0.0 /)
Part-Species1-Init1-BaseVector1IC         = (/ 1.0, 0.0, 0.0 /)
Part-Species1-Init1-BaseVector2IC         = (/ 0.0, 1.0, 0.0 /)
Part-Species1-Init1-NormalIC              = (/ 0.0, 0.0, 1.0 /)
```

The first and second base vector span a plane, where a circle with the given radius will be defined at the base point and then extruded in the normal direction up to the cylinder height.

For symmetric simulations `Part-Species1-Init1-BaseVector2IC` is set in direction of the symmetry `(/ 0.0, 1.0, 0.0 /)` for 2D planar simulations, and `(/ 0.0, 0.0, 1.0 /)` for axisymmetric ones.

Sphere

To define the cuboid the following parameters are required:

```
Part-Species1-Init1-SpaceIC               = sphere
Part-Species1-Init1-RadiusIC              = 1
Part-Species1-Init1-BasePointIC           = (/ 0.0, 0.0, 0.0 /)
Part-Species1-Init1-NormalIC              = (/ 0.0, 0.0, 1.0 /)
```

Photo-ionization

A special case is the ionization of a background gas through photon impact, modelling a light pulse. The volume affected by the light pulse is approximated by a cylinder (or honeycomb/rectangle), which is defined as described in Section *Cylinder*. Additionally, the SpaceIC has to be adapted and additional parameters are required:

```
Part-Species1-Init1-SpaceIC       = photon_cylinder ! or photon_honeycomb, or photon_
↪rectangle
Part-Species1-Init1-PulseDuration = 1               ! [s]
Part-Species1-Init1-WaistRadius   = 1E-6            ! [m]
Part-Species1-Init1-WaveLength    = 1E-9            ! [m]
Part-Species1-Init1-NbrOfPulses   = 1               ! [-], default = 1
```

The pulse duration and waist radius are utilized to define the spatial and temporal Gaussian profile of the intensity. The number of pulses allows to consider multiple light pulses within a single simulation. To define the intensity of the light pulse, either the average pulse power (energy of a single pulse times repetition rate), the pulse energy or the intensity amplitude have to be provided.

```
Part-Species1-Init1-Power                 = 1         ! [W]
Part-Species1-Init1-RepetitionRate        = 1         ! [Hz]
! or
```

(continues on next page)

(continued from previous page)

```
Part-Species1-Init1-Energy                  = 1         ! [J]
! or
Part-Species1-Init1-IntensityAmplitude      = 1         ! [W/m^2]
```

The intensity can be scaled with an additional factor to account for example for reflection or other effects:

```
Part-Species1-Init1-EffectiveIntensityFactor    = 1         ! [-]
```

It should be noted that this initialization should be done with a particle species (i.e. not the background gas species) that is also a product of the ionization reaction. The ionization reactions are defined as described in Section *Chemistry & Ionization* by

```
DSMC-NumOfReactions = 1
DSMC-Reaction1-ReactionType = phIon
DSMC-Reaction1-Reactants    = (/3,0,0/)
DSMC-Reaction1-Products     = (/1,2,0/)
DSMC-Reaction1-CrossSection = 4.84E-24      ! [m^2]
```

The probability that an ionization event occurs is determined based on the given cross-section, which is usually given for a certain wave length/photon energy. It should be noted that the background gas species should be given as the sole reactant and electrons should be defined as the first and/or second product. Electrons will be emitted perpendicular to the light path defined by the cylinder axis according to a cosine squared distribution.

Finally, the secondary electron emission through the impinging light pulse on a surface can also be modelled by an additional insertion region (e.g. as an extra initialization for the same species). Additionally to the definition of the light pulse as described above (pulse duration, waist radius, wave length, number of pulses, and power/energy/intensity), the following parameters have to be set

```
Part-Species1-Init2-SpaceIC               = photon_SEE_disc     ! or photon_SEE_
↪honeycomb, or photon_SEE_rectangle
Part-Species1-Init2-velocityDistribution  = photon_SEE_energy
Part-Species1-Init2-YieldSEE              = 0.1                 ! [-]
Part-Species1-Init2-WorkFunctionSEE       = 2                   ! [eV]
```

The emission area is defined as a disc by the parameters introduced in Section *Circular Disc*. The yield controls how many electrons are emitted per photon impact and their velocity distribution is defined by the work function. The scaling factor defined by `EffectiveIntensityFactor` is not applied to this surface emission. Both emission regions can be sped-up if the actual computational domain corresponds only to a quarter of the cylinder:

```
Part-Species1-Init1-FirstQuadrantOnly       = T
Part-Species1-Init2-FirstQuadrantOnly       = T
```

Emission Distribution

To initialize a pre-defined distribution, e.g., from the output of a field-based or continuum-based solver, a particle distribution can be created from a .h5 file that contains n, T, v_r and v_z (particle number density, temperature and velocity in 2D cylindrical coordinates). This data needs to be supplied in a specific format and a different array for each species that is to be initialized in such a way is required. This emission option is selected via

```
! Define the name of the data file
Part-EmissionDistributionFileName = reggie-linear-rot-symmetry-species-init.h5

```

(continues on next page)

(continued from previous page)

```
! OPTIONAL: Polynomial degree for particle emission in each element
Part-EmissionDistributionN = 1

! For each species, the following information is required
Part-Species1-nInits   = 1
Part-Species1-Init1-SpaceIC                  = EmissionDistribution
Part-Species1-Init1-EmissionDistributionName = HeIon
```

where `Part-EmissionDistributionFileName` defines the .h5 data file that contains the data, `Part-EmissionDistributionN` is an optional parameter for tuning the quality of the distribution within each element. It defines the polynomial degree for the particle emission in each element. The default value is 2(N+1) with N being the polynomial degree of the solution. The parameter `Part-Species1-Init1-SpaceIC` activates this specific emission type and `Part-Species1-Init1-EmissionDistributionName` is the name of the container in the .h5 file that yields the data for each species.

An example setup is given in the regression check directory under ./regressioncheck/CHE_PIC_maxwell_RK4/2D_variable_particle_init_n_T_v. The example uses 2D data defined in cylindrical coordinates for the velocity vector $v = v(r, z)$, which will be transformed to Cartesian coordinates in piclas.

The .h5 file `reggie-linear-rot-symmetry-species-init.h5` contains the following information: **Attributes** that define the index of the coordinates and the properties

```
T   4
n   3
r   1
vr  5
vz  6
z   2
```

and two **array** container, one for each species labelled `HeIon` and `electron` for singly charged Helium ions and electrons. These names must be used in the parameter input file. Each of these containers must provide the data in a $m \times n$ array and must be equidistant in each coordinate direction. The electron data begins with the following data

```
1.0   5.0     NaN      NaN        NaN        NaN
1.0   7.0  2.4E17  10000.0  1000000.0  1000000.0
1.0   9.0  2.4E17  10000.0  1000000.0  1000000.0
1.0  11.0  2.4E17  10000.0  1000000.0  1000000.0
1.0  13.0  2.4E17  10000.0  1000000.0  1000000.0
...
...
```

and is allowed to contain NaN values, because the original data might be projected onto a equidistant Cartesian grid from a unstructured and non-rectangular mesh. These NaN values will automatically be replaced with zeros during read-in of the data. The original 2D data (equidistant mesh) must be unrolled into a 1D structure with one data point per row. As can be seen from the dataset above, the first column containing the r-coordinates is the outer loop and the z-coordinates in the second column represent the inner loop in this logic. Note that the temperature (translational, vibrational and electronic) of ions and atoms/molecules will be initialized with 300 K. This will be changed in a future release. At the moment, only the electron temperature will be considered.

Neutralization Boundaries (neutral outflow condition)

There are different methods implemented to neutralize a charged particle flow, e.g., as encountered when simulation electric propulsion systems. Currently all methods require a specific geometry to function properly. For more details, see the regression tests under *regressioncheck/NIG_PIC_poisson_Boris-Leapfrog*. The following table lists the *SpaceIC* emission types

Distribution	Description
2D_landmark_neutraliza	Charoy 2019 2D PIC benchmark, electrons are injected with 10 eV at the cathode if the anode current is negative
2D_Liu2010_neutralizati	Liu 2010 2D PIC benchmark, electrons are injected at the cathode if the cathode current is negative
2D_Liu2010_neutralizati	Liu 2010 2D PIC benchmark, electrons are injected in the first cell layer at the cathode if the net charge in these elements is positive
3D_Liu2010_neutralizati	Liu 2010 3D PIC benchmark, electrons are injected at the cathode if the cathode current is negative
3D_Liu2010_neutralizati	Liu 2010 3D PIC benchmark, electrons are injected in the first cell layer at the cathode if the net charge in these elements is positive

For the *XD_Liu2010_neutralization* emission, a constant emitted electron temperature is defined via

```
Part-SpeciesX-InitX-MWTemperatureIC = 5.80E+04 ! 5.0 eV
```

whereas it is also possible to use a variable temperature, in which case the global (bulk) electron temperature is used, by setting

```
Part-SpeciesX-InitX-velocityDistribution = 2D_Liu2010_neutralization
```

for the 2D setup and

```
Part-SpeciesX-InitX-velocityDistribution = 3D_Liu2010_neutralization
```

for the 3D setup. The bulk electron temperature is determined automatically and output to *PartAnalyze.csv* as *XXX-BulkElectronTemp-[K]* to track this value over time.

Polychromatic Photo-ionization

The volumetric photo-ionization can consider multiple wavelengths (polychromatic spectrum) and/or energy-dependent cross-section data. The corresponding ionization reactions are defined described in Section *Chemistry & Ionization* by

```
DSMC-NumOfReactions = 1
DSMC-Reaction1-ReactionType = phIonXsec
DSMC-Reaction1-Reactants    = (/3,0,0/)
DSMC-Reaction1-Products     = (/1,2,0/)
```

where the reaction type `phIonXsec` refers to energy-dependent cross-section data for photoionization reactions. In this example, species 3 refers to H2 molecules, species 1 and 2 to electrons and H2+ ions respectively. The cross sections and photon energy spectrum must be supplied via

```
Particles-CollXSec-Database = XSec_Database_H2_Photoionization.h5
```

that must contain the data in the following form

```
XSec_Database_H2_Photoionization.h5
  - H2-photon (Group)
    - REACTION (Group)
      - H2Ion1-electron (Dataset)
    - SPECTRUM (Group)
      - H2-photon (Dataset)
```

where `HIon1-electron (Dataset)` contains the tabulated cross-sections and `H2-photon (Dataset)` contains the tabulated photon energies and energy fractions (the fractions must add up to unity). In principle, the spectrum can contain only 1 single photon energy (corresponding to a single wavelength) that contains all the energy, hence, the table contains the energy in eV and the number 1. (100% of the energy).

Initial Ionization

A neutral DSMC simulation can be converted into a PIC simulation (actually any simulation result state file `*_State_*.h5`) by specifying the number of species, their species ID and the desired ionization degree.

```
! Initial Ionization
Part-DoInitialIonization       = T       ! ON/OFF Switch
InitialIonizationSpecies       = 2       ! Total number of ions/neutrals that are to be␣
↪created
InitialIonizationSpeciesID     = (/1,3/) ! Species 1 und 3 will be created (all read-in␣
↪particles will be converted).
                                         ! Electrons do not have to be defined here as␣
↪they are found automatically.
InitialIonizationChargeAverage = 0.1     ! 10% ionization degree
```

The switch `Part-DoInitialIonization=T` must be deactivated after the ionization was successful in order to prevent multiple ionization events of subsequent state files when further restarts are performed. Note that all particle species that are found in the state file are considered for ionization, i.e., that the number of species (`InitialIonizationSpecies`) is only required for all non-electron species that need to be present after the initial ionization has been performed. In this example, the state file contains solely particles of species 1 (neutral atoms or molecules). These particles are converted to 10 % species 3 (ions) and 90 percent species 1 (the original neutral particles), in addition to electrons which are found automatically by comparing the charge of the species (in this example species index 2). For each ion an electron is created to maintain charge neutrality. The ionization degree (`InitialIonizationChargeAverage`) should be a number between 0 and 1.

Surface Flux

A surface flux enables the emission of particles at a boundary in order to simulate, e.g. a free-stream. They are defined species-specifically and can overlap. First, the number of surface fluxes has to be given

```
Part-Species1-nSurfaceFluxBCs=1
```

The surface flux is mapped to a certain boundary by giving its boundary number (e.g. `BC=1` corresponds to the previously defined boundary `BC_OPEN`)

```
Part-Species1-Surfaceflux1-BC=1
```

The remaining parameters such as flow velocity, temperature and number density are given analogously to the initial particle insertion presented in Section *Initialization*. An example to define the surface flux for a diatomic species is given below

```
Part-Species1-Surfaceflux1-VeloIC=1500
Part-Species1-Surfaceflux1-VeloVecIC=(/-1.0,0.0,0.0/)
Part-Species1-Surfaceflux1-velocityDistribution=maxwell_lpn
Part-Species1-Surfaceflux1-MWTemperatureIC=300.
Part-Species1-Surfaceflux1-PartDensity=1E20
Part-Species1-Surfaceflux1-TempRot=300.
Part-Species1-Surfaceflux1-TempVib=300.
Part-Species1-Surfaceflux1-TempElec=300.
```

Emission Current & Mass Flow

Instead of the particle number density `PartDensity`, an emission current I [A] (e.g. to model a thermionic emission) or a mass flow $\dot{m}$ [kg/s] (e.g. to model outgassing) can be given:

```
Part-Species1-Surfaceflux1-EmissionCurrent=2
! or
Part-Species1-Surfaceflux1-Massflow=1e-11
```

In this case, the number of simulation particles to be inserted each time step Δt is determined directly from the rate. The emission current only allows charged species and determines the number of particles according to the charge. The velocity magnitude can be zero (per default) or a defined value (through `VeloIC` and `VeloVecIC`). The respective boundary can be `open` or `reflective`. An example can be found in the regression test `regressioncheck/CHE_DSMC/SurfFlux_Tria_CurrentMassflow` For subsonic boundary conditions, where the velocity at the boundary is unknown, refer to Section *Adaptive/Subsonic Boundaries*.

Thermionic Emission (including Schottky effect)

The Richardson-Dushman equation including the Schottky effect is implemented and can be enabled to model thermionic emission

$$j = A^* T_\mathrm{w}^2 \exp\left(-\frac{W^*}{k_\mathrm{B} T_\mathrm{w}}\right),$$

where the work function W^* is defined by

$$W^* = W - \Delta W \qquad \Delta W = \sqrt{\frac{q_\mathrm{e}^3 |\mathbf{E}|}{4\pi\epsilon_0}}.$$

The magnitude of the electric field strength $|\mathbf{E}|$ is calculated with the average value of the interpolation points at the boundary. The material-specific properties such as the work function W [eV] and the (modified) Richardson constant A^* [A/cm^2/K^2] have to be provided as input. In addition to the surface flux parameters, a wall temperature T_w for the respective boundary has to be defined (as shown in Section *Boundary Conditions - Particle Solver*)

```
Part-Boundary1-WallTemp = 2000.
Part-Species1-Surfaceflux1-ThermionicEmission                    = TRUE
Part-Species1-Surfaceflux1-ThermionicEmission-SchottkyEffect     = TRUE
Part-Species1-Surfaceflux1-ThermionicEmission-WorkFunction       = 3
Part-Species1-Surfaceflux1-ThermionicEmission-RichardsonConstant = 120
```

The provided temperature for the surface flux of the species determines the energy of emitted particles. While the thermionic emission can be enabled for PIC as well as DSMC simulations, the addition of the Schottky effect requires a field solver. An overview of the limitations of this modelling regarding the applied field strength, wall temperature and/or material is given by Ref. [22] and Ref. [23]. An example can be found in the regression test `regressioncheck/CHE_poisson/SurfFlux_ThermionicEmission_Schottky`.

Circular Inflow

The emission of particles from a surface flux can be limited to the area within a circle, ring or circle cut-out. The respective boundary has to coincide or be parallel to the xy-, xz, or yz-planes. This allows to define inflow boundaries without specifically meshing the geometrical feature, e.g. small orifices. The feature can be enabled per species and surface flux

```
Part-Species1-Surfaceflux1-CircularInflow = TRUE
```

The normal direction of the respective boundary has to be defined by

```
Part-Species1-Surfaceflux1-axialDir = 1
```

Finally, the origin of the circle/ring on the surface and the radius have to be given. In the case of a ring, a maximal and minimal radius is required (`-rmax` and `-rmin`, respectively), whereas for a circle only the input of maximal radius and for the circle cut-out only the minimal radius is sufficient.

```
Part-Species1-Surfaceflux1-origin   = (/5e-6,5e-6/)
Part-Species1-Surfaceflux1-rmax     = 2.5e-6
Part-Species1-Surfaceflux1-rmin     = 1e-6
```

The absolute coordinates are defined as follows for the respective normal direction.

Normal Direction	Coordinates
x (=1)	(y,z)
y (=2)	(z,x)
z (=3)	(x,y)

Multiple circular inflows can be defined on a single boundary through multiple surface fluxes, e.g. to enable the simulation of multiple inlets on a chamber wall. Circular inflows are also supported with axisymmetric simulations, under the assumptions that the chosen surface is in the yz-plane (and thus has a normal direction in x) and the minimal and maximum radii are in the positive y-direction. Examples are given as part of the regression tests in `regressioncheck/CHE_DSMC/SurfFlux_Tria_CircularInflow_Circle`, `SurfFlux_Tria_CircularInflow_CircleCutout` and `SurfFlux_Tria_CircularInflow_Ring`.

Adaptive/Subsonic Boundaries

Different adaptive boundaries can be defined as a part of a surface flux to model subsonic in- and outflows, where the emission is adapted based on the prevalent conditions at the boundary. The modelling is based on the publications by Ref. [24] and Ref. [12].

```
Part-Species1-Surfaceflux1-Adaptive = TRUE
Part-Species1-Surfaceflux1-Adaptive-Type = 1
```

An overview over the available types is given below.

- `Type=1`: Constant static pressure and temperature inlet, defined as Type 1 in Ref. [24]
- `Type=2`: Constant static pressure outlet, defined as Type 1 in Ref. [24]
- `Type=3`: Constant mass flow and temperature inlet, where the given mass flow and sampled velocity are used to determine the number of particles for the surface flux. It requires the BC to be defined as `open`. Defined as Type 2 in Ref. [24]

- `Type=4`: Constant mass flow inlet and temperature inlet, where number of particles to be inserted is determined directly from the mass flow and the number of particles leaving the domain, $N_{\rm in} = N_{\dot{m}} + N_{\rm out}$. Defined as cf_3 in Ref. [12]

Depending of the type of the chosen boundary type either the mass flow [kg/s] or the static pressure [Pa] have to be given

```
Part-Species1-Surfaceflux1-Adaptive-Massflow = 1.00E-14
Part-Species1-Surfaceflux1-Adaptive-Pressure = 10
```

The adaptive boundaries require the sampling of macroscopic properties such as flow velocity at the boundary. To compensate for the statistical fluctuations, three possible sampling approaches are available. The first approach uses a relaxation factor $f_{\rm relax}$, where the current value of the sampled variable v^n is updated according to

$$v^n = (1 - f_{\rm relax})\, v^{n-1} + f_{\rm relax} v^{\rm samp}$$

The relaxation factor $f_{\rm relax}$ is defined by

```
AdaptiveBC-RelaxationFactor = 0.001
```

The second and third approach allows to sample over a certain number of iterations. If the truncated running average option is enabled, the macroscopic properties will be continuously updated while the oldest sample will be replaced with the most recent. If the truncated running average option is disabled, the macroscopic properties will be only updated every given number of iterations, and the complete sample will be resetted afterwads. If a number of iterations is given, it will be used instead of the first approach with the relaxation factor.

```
AdaptiveBC-SamplingIteration      = 100
AdaptiveBC-TruncateRunningAverage = T       ! DEFAULT: F
```

The adaptive particle emission can be combined with the circular inflow feature. In this context when the area of the actual emission circle/ring is very small, it is preferable to utilize the `Type=4` constant mass flow condition. `Type=3` assumes an open boundary and accounts for particles leaving the domain through that boundary already when determining the number of particles to be inserted. As a result, this method tends to over predict the given mass flow, when the emission area is very small and large sample size would be required to have enough particles that leave the domain through the emission area. For the `Type=4` method, the actual number of particles leaving the domain through the circular inflow is counted and the mass flow adapted accordingly, thus the correct mass flow can be reproduced.

Additionally, the `Type=4` method can be utilized in combination with a reflective boundary condition to model diffusion and leakage (e.g. in vacuum tanks) based on a diffusion rate Q [Pa m^3 s^{-1}]. The input mass flow [kg s^{-1}] for the simulation is then determined by

$$\dot{m} = \frac{QM}{1000RT},$$

where $R = 8.314$ J mol^{-1}K^{-1} is the gas constant, M the molar mass in [g mol^{-1}] and T is the gas temperature [K]. It should be noted that while multiple adaptive boundaries are possible, adjacent boundaries that share a mesh element should be avoided or treated carefully. Examples are given as part of the regression tests in `regressioncheck/CHE_DSMC/SurfFlux_Tria_Adaptive_ConstMassflow` and `SurfFlux_Tria_Adaptive_ConstPressure`.

Verification

To verify the resulting current [A], mass flow rate [kg s^{-1}] or the pressure at an adaptive surface flux [Pa], the following option can be enabled

```
CalcSurfFluxInfo = T
```

This will output a species-specific rate and/or the average pressure in the adjacent cells (in case of an adaptive/subsonic BC) for each surface flux condition in the `PartAnalyze.csv`. It gives the current values for the time step. For the former, positive values correspond to a net flux into the domain and negative values vice versa.

Missing descriptions

ReduceNoise, DoForceFreeSurfaceFlux

DoPoissonRounding: [25]

AcceptReject, ARM_DmaxSampleN: [26]

1.4.6 Particle-In-Cell

Charge and Current Deposition

Charge and current deposition can be performed using different methods, among others, shape functions, B-splines or locally volume-weighted approaches.

PIC-Deposition-Type	Description
cell_volweight_mean	Linear distribution in each element (continuous on element boundaries)
shape_function	standard shape function with fixed radius
shape_function_cc	charge corrected shape function with fixed radius
shape_function_adaptive	charge corrected shape function with element-dependent radius

Linear Distribution Over Cell Interfaces

A linear deposition method that also considers neighbouring elements can be selected by

```
PIC-Deposition-Type = cell_volweight_mean
```

and is referred to as the CVWM method. This method also considers the corner nodes of each element to which all neighbouring elements contribute, hence, resulting in a non-local deposition scheme. Note that the CVWM method allows switching of charge deposition on Dirichlet boundaries via

```
PIC-DoDirichletDeposition = F
```

which simply nullifies the deposited charge on wall boundary nodes for Dirichlet sides to account for mirror charges. The default value for this parameter is true and it is currently only available for the CVWM method in combination with the HDG method.

Shape Function

High-order field solvers require deposition methods that reduce the noise, e.g., shape functions [27]. The standard 3D shape function is selected by

```
PIC-Deposition-Type = shape_function
```

or

```
PIC-Deposition-Type = shape_function_cc
```

where `shape_function_cc` is a numerically charge-conserving method that adjusts the deposited charge by comparing its integral value to the total charge given by the particles.

The shape function sphere might be truncated at walls or open boundaries, which is compensated when using `shape_function_cc` by increasing the deposited charge of truncated particles.

Additionally, an element-local shape function radius can be used, which is determined for each element separately depending on the size of the element and its direct neighbours by setting

```
PIC-Deposition-Type = shape_function_adaptive
```

The shape function radius in this case is limited by the size of the surrounding elements and may not reach past its direct neighbours.

The direct influence of only the neibouring elements can be extended further by activating

```
PIC-shapefunction-adaptive-smoothing = T
```

which increases the radius of influence and therefore takes more elements into account for the calculation of the shape function radius in each element, hence, leading to a smoother transition in regions, where the element sizes rapidly change.

This shape function method also is numerically charge conserving by integrating each particle's deposited charge and adjusting to this value. Depending on the polynomial degree N, the number of DOF that are within the shape function radius can be changed via

```
PIC-shapefunction-adaptive-DOF = 33
```

The default values (maximum allowed for each polynomial degree N) depend on the dimensionality of the deposition kernel, 1D: $2(N+1)$, 2D: $\pi(N+1)^2$, 3D: $(4/3)\pi(N+1)^3$.

The following polynomial isotropic shape functions are all designed to be used in three dimensions, where reductions to 2D and 1D are possible.

Shape Function 1D

A one-dimensional shape function in x-direction is given by

$$S_{1D}(r,R,\alpha) = \frac{\Gamma(\alpha+3/2)}{\sqrt{\pi}R\Gamma(\alpha+1)\Delta y\Delta z}\left(1-\left(\frac{r}{R}\right)^2\right)^{\alpha},$$

which is normalized to give $\int_{z_1}^{z_2}\int_{y_1}^{y_2}\int_{-R}^{R} S_{1D}(r,R,\alpha)dxdydz = 1$, where the radius $r = |\boldsymbol{x}-\boldsymbol{x}_n| = |x-x_n|$ is the distance between the position of the grid point at position $\boldsymbol{x}$ and the n-th particle at position $\boldsymbol{x}_n$, R is the cut-off radius,

$\Delta y = y_2 - y_1$ and $\Delta z = z_2 - z_1$ are the domain lengths in y- and z-direction, respectively, and $\Gamma(z)$ is the gamma function given by

$$\Gamma(z) = \int_0^\infty x^{z-1} \exp(-x) dx \ .$$

The direction in which deposition is performed is chosen via

```
PIC-shapefunction-direction = 1 ! for x-direction
                              2 ! for y-direction
                              3 ! for z-direction
```

and the dimensionality of the shape function is controlled by

```
PIC-shapefunction-dimension = 1 ! for 1D
                              2 ! for 2D
                              3 ! for 3D
```

which has to be set to 1 in the case of 1D deposition.

Shape Function 2D

A two-dimensional shape function in x-y-direction is given by

$$S_{2D}(r, R, \alpha) = \frac{\alpha + 1}{\pi R^2 \Delta z} \left(1 - \left(\frac{r}{R}\right)^2\right)^\alpha \ ,$$

which is normalized to give $\int_{z_1}^{z_2} \int_0^{2\pi} \int_0^R S_{2D}(r, R, \alpha) r dr d\phi d\theta = 1$, where the radius $r = |\boldsymbol{x} - \boldsymbol{x}_n|$ is the distance between the position of the grid point at position $\boldsymbol{x}$ and the n-th particle at position $\boldsymbol{x}_n$, R is the cut-off radius and $\Delta z = z_2 - z_1$ is the domain length in z-direction. The perpendicular direction to the two axes, in which deposition is performed is chosen via

```
PIC-shapefunction-direction = 1 ! for const. depo in x-direction
                              2 ! for const. depo in y-direction
                              3 ! for const. depo in z-direction
```

when the charge is to be deposited const. along the x- or y- or z-direction. If the charge is to be deposited over the area instead of the volume, the flag

```
PIC-shapefunction-3D-deposition=F
```

must be set, which simply sets $\Delta z = 1$ for the example described above. Again, the dimensionality of the shape function is controlled by

```
PIC-shapefunction-dimension = 1 ! for 1D
                              2 ! for 2D
                              3 ! for 3D
```

which has to be set to 2 in the case of 2D deposition.

Shape Function 3D

A three-dimensional shape function in x-y-direction is given by [28]

$$S_{3D}(r,R,\alpha) = \frac{\Gamma(\alpha+5/2)}{\pi^{3/2}R^3\Gamma(\alpha+1)}\left(1-\left(\frac{r}{R}\right)^2\right)^{\alpha},$$

which is normalized to give $\int_0^{\pi}\int_0^{2\pi}\int_0^{R} S_{2D}(r,R,\alpha)r^2\sin(\phi)drd\phi d\theta = 1$, where the radius $r = |\boldsymbol{x}-\boldsymbol{x}_n|$ is the distance between the position of the grid point at position $\boldsymbol{x}$ and the n-th particle at position $\boldsymbol{x}_n$ and R is the cut-off radius.

1.4.7 Magnetic Background Field

Certain application cases allow the utilization of a constant magnetic background field. This background field can either be supplied via .csv (1D) or .h5 (2D) file, however, in this case the field must be based on an equidistant Cartesian mesh. Another method is to use the built-in tool **superB**, which is also available as stand-alone executable to generate magnetic fields based on magnets or coils, which results in the creation of a .h5 file containing the field data based on a PICLas (HOPR) mesh file. The following two sections give an overview of using the different methods.

Variable External Field

One-, two- and three-dimensional magnetic fields can be used as fixed background fields for certain time discretization methods (full Maxwell time discs and the Poisson Boris-Leapfrog scheme) The read-in variable for either .csv (only for 1D along z-direction) or .h5 (only for 2D axis symmetric z-direction or fully 3D) files is set via

```
PIC-variableExternalField = X.csv, X.h5
```

Three examples are located within the regression test directory

```
regressioncheck/CHE_PIC_maxwell_RK4/gyrotron_variable_Bz
regressioncheck/CHE_PIC_maxwell_RK4/2D_variable_B
regressioncheck/CHE_PIC_maxwell_RK4/3D_variable_B
```

for 1D, 2D and 3D fields, respectively. Note that 1D currently only allows magnetic fields of type $B_z(z)$ and 2D only allows the components $B_r(r,z)$ and $B_z(r,z)$ that comprise a rotationally symmetric vector field $\mathbf{B}$.

The first example (1D and .csv file) uses data via

```
PIC-variableExternalField = variable_Bz.csv
```

which is csv-based data in the form (the delimiter is actually not a comma)

```
-0.00132         2.7246060625
-0.000217551020408        2.700481016
0.0008848979591837        2.6762685135
0.0019873469387755        2.6519260266
0.0030897959183674        2.6274128336
....
```

and the second (2D and .h5 file)

```
PIC-variableExternalField = reggie-linear-rot-symmetry.h5
```

that is read from a .h5 file. The data structure in the .h5 file must be of the form (dataset is labelled "data") and contains

```
r1 z1 Br1 Bz1
r2 z2 Br2 Bz2
r3 z3 Br3 Bz3
r4 z4 Br4 Bz4
r5 z5 Br5 Bz5
....
```

where for each data point one row is required. The ordering of the data is also important. It is only allowed that the first N rows have the same r value and varying z-components (r is the outer loop variable and z is the inner loop variable when unrolling the data into an array). This is automatically checked by comparing the distances in r and z direction, which must be equidistant. In addition, the attributes r, z, Br and Bz, which contain the indices of the corresponding column number in "data".

Three-dimensional fields must be supplied in the following format

```
x1 y1 z1 Bx1 By1 Bz1
x2 y2 z2 Bx2 By2 Bz2
x3 y3 z3 Bx3 By3 Bz3
x4 y4 z4 Bx4 By4 Bz4
x5 y5 z5 Bx5 By5 Bz5
....
```

where the data (dataset is labelled "data" in the .h5 file) is sorted in lines in ascending coordinates. For everything to work, the order must be like this

```
x1 y1 z1 Bx1 By1 Bz1
x2 y1 z1 Bx2 By2 Bz2
x1 y2 z1 Bx3 By3 Bz3
x2 y2 z1 Bx4 By4 Bz4
x1 y1 z2 Bx5 By5 Bz5
x2 y1 z2 Bx6 By6 Bz6
....
```

where first the x-coordinate changes, then y and finally the z.

superB

The magnetic field resulting from certain types of coils and permanent magnets can be calculated during the initialization within PICLas or with the standalone tool **superB** (see Section *Compiler options* for compilation), which can be used to solely create a .h5 file that contains the B-field data via

```
superB parameter_superB.ini
```

For usage in PICLas, the background field can be enabled by

```
PIC-BG-Field = T
```

The first option is to use a previously calculated background field. It can be read-in with

```
PIC-BGFileName     = BField.h5 ! Path to a .h5 file that contains the B-field data
PIC-NBG            = 1         ! Polynomial degree of the B-field
PIC-BGFieldScaling = 1.        ! Scaling factor for the B-field
```

Additionally, the polynomial degree for the background field can be set by `PIC-NBG` and might differ from the actually read-in polynomial degree that is used to represent the numerical solution for the field solver. Optionally, the read-in field can be scaled by the last of the three parameters above.

The second option is to calculate the magnetic field during the initialization, which will produce an output .h5 file of the field. The field will automatically be calculated from the supplied parameters, if the corresponding .h5 file does not exist. For this purpose, different coil and permanent magnet geometries can be defined. For visualization purposes, the geometry of the respective coils and permanent magnets can be directly written out as a VTK with

```
PIC-CalcBField-OutputVTK = T
```

In the following the parameters for different coils and permanent magnets based on the implementation by Hinsberger [29] are presented.

Magnetic Field by Permanent Magnets

First, the total number of permanent magnets has to be defined and the type selected. Options are `cuboid`, `sphere`, `cylinder` and `conic`.

```
NumOfPermanentMagnets = 1
PermanentMagnet1-Type = cuboid
                        sphere
                        cylinder ! also used for hollow cylinders
                        conic
```

All options require the input of a base/origin vector, a number of discretization nodes (results in a different number of total points depending on the chosen geometry) and a magnetisation in [A/m]

```
PermanentMagnet1-BasePoint = (/0.,0.,0./)
PermanentMagnet1-NumNodes = 10
PermanentMagnet1-Magnetisation = (/0.,0.,1./)
```

The geometries require different input parameters given below

```
! Three vectors spanning the cuboid
PermanentMagnet1-BaseVector1 = (/1.,0.,0./)
PermanentMagnet1-BaseVector2 = (/0.,1.,0./)
PermanentMagnet1-BaseVector3 = (/0.,0.,1./)
! Radius required for a spherical, cylindrical and conical magnet
PermanentMagnet1-Radius = 1.
! Height vector required for a cylindrical and conical magnet
PermanentMagnet1-HeightVector = (/0.,0.,1./)
! Second radius only required for a conical magnet or a hollow cylinder with inner radius
! 'Radius2' and outer radius "Radius1'
PermanentMagnet1-Radius2 = 1.
```

Magnetic Field by Coils

The total number of coils and the respective type of the cross-section (`linear`,`circle`,`rectangle`,`custom`) is defined by

```
NumOfCoils = 1
Coil1-Type = linear
             circle
             rectangle
             custom
```

All options require the input of a base/origin vector, a length vector (vector normal to the cross-section of the coil) and the current in [A]

```
Coil1-BasePoint = (/0.0,0.0,0.0/)
Coil1-LengthVector = (/0.0,1.0,0.0/)
Coil1-Current = 1.
```

The first option `linear` represents a simple linear conductor (e.g. a straight wire) and requires only the input of a number of discretization points

```
Coil1-NumNodes = 5
```

The other three types, which are actually coils, are described by the number of loops and the number of discretization points per loop

```
Coil1-LoopNum = 10
Coil1-PointsPerLoop = 10
```

The cross-section of the coil is defined in a plane normal to the `-LengthVector`. A circular coil cross-section requires simply the input of a radius whereas a rectangular coil cross-section is spanned by two vectors (`-RectVec1` and `-RectVec2`) and an additional vector, which must be orthogonal to the `-LengthVector` to define the orientation of the cross-section (`-AxisVec1`). In these two cases, the base/origin vector defines the middle point of the cross-section.

```
! Circular coil cross-section
Coil1-Radius = 1.
! Rectangular coil cross-section
Coil1-RectVec1 = (/1.0,0.0/)
Coil1-RectVec2 = (/0.0,1.0/)
Coil1-AxisVec1 = (/0.0,0.0,1.0/)
```

The last cross-section type `custom` allows the definition of a cross-section as a combination of multiple linear (`line`) and circular (`circle`) segments and also requires an additional vector to define the orientation of the cross-section (`-AxisVec1`)

```
Coil1-AxisVec1 = (/0.0,0.0,1.0/)
Coil1-NumOfSegments = 3
! Linear segment defined by
Coil1-Segment1-SegmentType = line
Coil1-Segment1-NumOfPoints = 5
Coil1-Segment1-LineVector = (/1.0,1.0/)
! Circular segment connected to the previous segment
Coil1-Segment2-SegmentType = circle
Coil1-Segment2-NumOfPoints = 5
```

(continues on next page)

(continued from previous page)

```
Coil1-Segment2-Radius = 1.
Coil1-Segment2-Phi1 = 90.
Coil1-Segment2-Phi2 = 0.
! Linear segment connected to the previous segment, closing the cross-section
Coil1-Segment3-SegmentType = line
Coil1-Segment3-NumOfPoints = 5
Coil1-Segment3-LineVector = (/-2.0,0.0/)
```

The `-NumOfPoints` controls the number of discretization points per segment. A linear segment is simply described by a vector in the cross-section plane. The circular segment is defined by a radius and the initial as well as final angle of the segment. It should be noted that the base point defines the start of the first segment as opposed to the circular and rectangular cross-sections, where it is the middle point of the cross-section.

Time-dependent Magnetic Coils

A time-dependent magnetic field can be created by a time-varying electric current running through a coil. Time-dependent coils can be combined with an arbitrary number of permanent magnets and coils (with a constant current). Currently, all time-dependent coils must use the same frequency but can have different phases. The time-dependent settings are required in addition to the ones used for a standard coil

```
Coil1-TimeDepCoil      = T
Coil1-CurrentFrequency = 1e6
Coil1-CurrentPhase     = 0.0
nTimePoints            = 11
```

where the frequency and phase of the sin function that is used for the electric current as well as the number of points in time for the interpolation of the current is required. In piclas, times between two interpolation points are determined by linear interpolation from the stored solution.

1.4.8 Direct Simulation Monte Carlo

To enable the simulation with DSMC, an appropriate time discretization method including the DSMC module should be chosen before the code compilation. A stand-alone DSMC simulation can be enabled by compiling PICLas with the following parameter

```
PICLAS_TIMEDISCMETHOD = DSMC
```

The DSMC method can then be enabled in the parameter file by

```
UseDSMC = T
```

Additionally, the number of simulated physical models depending on the application can be controlled through

```
Particles-DSMC-CollisMode = 1   ! Elastic collisions only
                            2   ! Internal energy exchange
                            3   ! Chemical reactions
```

`CollisMode = 1` can be utilized for the simulation of a non-reactive, cold atomic gas, where no chemical reactions or electronic excitation is expected. `CollisMode = 2` should be chosen for non-reactive diatomic gas flows to include the internal energy exchange (by default including the rotational and vibrational energy treatment). Finally, reactive gas flows can be simulated with `CollisMode = 3`. The following sections describe the required definition of species

parameter (Section *Species Definition*), the parameters for the internal energy exchange (Section *Inelastic Collisions & Relaxation*) and chemical reactions (Section *Chemistry & Ionization*).

A fixed ("manual") simulation time step Δt is defined by

```
ManualTimeStep = 1.00E-7
```

Species Definition

For the DSMC simulation, additional species-specific parameters (collision model parameters, characteristic vibrational temperature, etc.) are required. This file is also utilized for the definition of chemical reactions paths. To avoid the manual input, species parameter can be read from a database instead. The procedure is described in Section *Unified Species Database*.

To define a species, its name as well as an `InteractionID` have to be defined

```
Part-Species1-SpeciesName = CH4
Part-Species1-InteractionID = 2
```

During the file-based parameter read-in, name is only utilized to retrieve the electronic energy levels from an additional database. The interaction ID determines the type of a species as follows

ID	Type
1	Atom
2	Molecule (diatomic and polyatomic)
4	Electron
10	Atomic Ion
20	Molecular Ion

Depending on the utilized collision model, different parameters have to be defined. As an example, the parameters for the Variable Hard Sphere (VHS) collision cross-section model are be defined by the temperature exponent $\omega = [0, 0.5]$, reference temperature T_{ref} [K] and diameter d_{ref} [m]

```
Part-Species1-omega = 0.24
Part-Species1-Tref = 273
Part-Species1-dref = 4.63E-10
```

More detail on the utilization of species-specific, collision-specific parameters and the utilization of the Variable Soft Sphere (VSS) model are given in Section *Pairing & Collision Modelling*.

In case of chemical reactions, the input of a species-specific heat/enthalpy of formation [K] is required. A reliable source for these reference values are the Active Thermochemical Tables (ATcT) by the Argonne National Laboratory [30, 31].

```
Part-Species1-HeatOfFormation_K = 0.0
```

In the case of ionization reactions the heat of formation is not required, as the ionization energy is read-in as the last electronic level of the neutral (or previous) state. Therefore, an additional entry for each ionic species about its previous state is required. This is done by providing the species index, an example is given below assuming that the first species is C, whereas the second and thirds species are the first and second ionization levels, respectively.

```
Part-Species2-PreviousState = 1
Part-Species3-PreviousState = 2
```

If the previous state of the ionized species is not part of the simulation (e.g. a reservoir with N, N_2^+, and e), the previous state of N_2^+ can be set to zero, however, in that case a heat/enthalpy of formation has to be provided, which includes the ionization energy (as given in ATcT).

Diatomic molecular species require the definition of the characteristic temperature [K] and their dissociation energy [eV] (which is at the moment utilized as a first guess for the upper bound of the temperature calculation as well as the threshold energy for the dissociation by the Quantum-Kinetic chemistry model)

```
Part-Species1-CharaTempVib = 4194.9
Part-Species1-Ediss_eV = 4.53
```

Polyatomic molecular species require an additional flag, the input of the number of atoms and whether the molecule is linear (e.g. CO_2, $\xi_{\mathrm{rot}} = 2$) or non-linear (e.g. H_2O, CH_4, $\xi_{\mathrm{rot}} = 3$). The number of the vibrational degrees of freedom is then given by

$$\alpha = 3N_{\mathrm{atom}} - 3 - \xi_{\mathrm{rot}}$$

As an example the parameters of CH_3 are given below. The molecule has four vibrational modes, with two of them having a degeneracy of two. These values are simply given the according amount of times

```
Part-Species1-NumOfAtoms = 4
Part-Species1-LinearMolec = FALSE
Part-Species1-CharaTempVib1 = 4320.6
Part-Species1-CharaTempVib2 = 872.1
Part-Species1-CharaTempVib3 = 4545.5
Part-Species1-CharaTempVib4 = 4545.5
Part-Species1-CharaTempVib5 = 2016.2
Part-Species1-CharaTempVib6 = 2016.2
```

These parameters allow the simulation of non-reactive gases. Additional parameters required for the consideration of chemical reaction are given in Section *Chemistry & Ionization*.

Pairing & Collision Modelling

By default, a conventional statistical pairing algorithm randomly pairs particles within a cell. Here, the mesh should resolve the mean free path to avoid numerical diffusion. To circumvent this requirement, an octree-based sorting and cell refinement [32] can be enabled by

```
Particles-DSMC-UseOctree        = T
Particles-OctreePartNumNode     = 80        ! (3D default, 2D default: 40)
Particles-OctreePartNumNodeMin  = 50        ! (3D default, 2D default: 28)
```

The algorithm refines a cell recursively as long as the mean free path is smaller than a characteristic length (approximated by the cubic root of the cell volume) and the number of particles is greater than `Particles-OctreePartNumNode`. The latter condition serves the purpose to accelerate the simulation by avoiding looking for the nearest neighbour in a cell with a large number of particles. To avoid cells with a low particle number, the cell refinement is stopped when the particle number is below `Particles-OctreePartNumNodeMin`. These two parameters have different default values for 2D/axisymmetric and 3D simulations.

To further reduce numerical diffusion, the nearest neighbour search for the particle pairing can be enabled

```
Particles-DSMC-UseNearestNeighbour = T
```

An additional attempt to increase the quality of simulations results is to prohibit repeated collisions between particles [33, 34]. This options is enabled by default in 2D/axisymmetric simulations, but disabled by default in 3D simulations.

```
Particles-DSMC-ProhibitDoubleCollision = T
```

The Variable Hard Sphere (VHS) is utilized by default with collision-averaged parameters, which are given per species

```
Part-Species1-omega = 0.24
Part-Species1-Tref = 273
Part-Species1-dref = 4.63E-10
```

To enable the Variable Soft Sphere (VSS) model, the additional α parameter is required

```
Part-Species1-alphaVSS = 1.2
```

In order to enable the collision-specific definition of the VHS/VSS parameters, a different input is required

```
! Input in parameter.ini
Particles-DSMC-averagedCollisionParameters = F
! Input in species.ini
Part-Collision1 - partnerSpecies = (/1,1/)              ! Collision1: Parameters for the␣
↪collision between equal species
Part-Collision1 - Tref           = 273
Part-Collision1 - dref           = 4.037e-10
Part-Collision1 - omega          = .216
Part-Collision1 - alphaVSS       = 1.448
Part-Collision2 - partnerSpecies = (/2,1/)              ! Collision2: Parameters for the␣
↪collision between species 2 and 1
```

The numbers in the `partnerSpecies` definition correspond to the species numbers and their order is irrelevant. Collision-specific parameters can be obtained from e.g. [35].

Cross-section based collision probabilities

Cross-section data to model collisional and relaxation probabilities (e.g. in case of electron-neutral collisions), analogous to Monte Carlo Collisions, can be utilized and is described in Section *Collision cross-sections* and *Cross-section Chemistry (XSec)*.

Inelastic Collisions & Relaxation

To consider inelastic collisions and relaxation processes within PICLas, the chosen `CollisMode` has to be at least 2

```
Particles-DSMC-CollisMode = 2
```

Two selection procedures are implemented, which differ whether only a single or as many as possible relaxation processes can occur for a collision pair. The default model (`SelectionProcedure = 1`) allows the latter, so-called multi-relaxation method, whereas `SelectionProcedure = 2` enables the prohibiting double-relaxation method [36]

```
Particles-DSMC-SelectionProcedure = 1    ! Multi-relaxation
                                    2    ! Prohibiting double-relaxation
```

Rotational, vibrational and electronic relaxation (not included by default, see Section *Electronic Relaxation* for details) processes are implemented in PICLas and their specific options to use either constant relaxation probabilities (default) or variable, mostly temperature dependent, relaxation probabilities are discussed in the following sections. To achieve consistency between continuum and particle-based relaxation modelling, the correction factor of Lumpkin [37] can be enabled (default = F):

```
Particles-DSMC-useRelaxProbCorrFactor = T
```

Rotational Relaxation

To adjust the rotational relaxation this variable has to be changed:

```
Particles-DSMC-RotRelaxProb = 0.2   ! Value between 0 and 1 as a constant probability
                              2   ! Model by Boyd
                              3   ! Model by Zhang
```

If `RotRelaxProb` is between 0 and 1, it is set as a constant rotational relaxation probability (default = 0.2). `RotRelaxProb = 2` activates the variable rotational relaxation model according to Boyd [38]. Consequently, for each molecular species two additional parameters have to be defined, the rotational collision number and the rotational reference temperature. As an example, nitrogen is used [39].

```
Part-Species1-CollNumRotInf = 23.3
Part-Species1-TempRefRot = 91.5
```

It is not recommended to use this model with the prohibiting double-relaxation selection procedure (`Particles-DSMC-SelectionProcedure = 2`). Low collision energies result in high relaxation probabilities, which can lead to cumulative collision probabilities greater than 1.

If the relaxation probability is equal to 3, the relaxation model of Zhang et al. [40] is used. However, it is only implemented for nitrogen and not tested. It is not recommended for use.

Vibrational Relaxation

Analogous to the rotational relaxation probability, the vibrational relaxation probability is implemented. This variable has to be changed, if the vibrational relaxation probability should be adjusted:

```
Particles-DSMC-VibRelaxProb = 0.004 ! Value between 0 and 1 as a constant probability
                                2 ! Model by Boyd
```

If `VibRelaxProb` is between 0 and 1, it is used as a constant vibrational relaxation probability (default = 0.004). The variable vibrational relaxation model of Boyd [39] can be activated with `VibRelaxProb = 2`. For each molecular species pair, the constants A and B according to Millikan and White [41] (which will be used for the calculation of the characteristic velocity and vibrational collision number according to Abe [42]) and the vibrational cross section have to be defined. The given example below is a 2 species mixture of nitrogen and oxygen, using the values for A and B given by Farbar [43] and the vibrational cross section given by Boyd [39]:

```
Part-Species1-MWConstA-1-1 = 220.00
Part-Species1-MWConstA-1-2 = 115.10
Part-Species1-MWConstB-1-1 = -12.27
Part-Species1-MWConstB-1-2 = -6.92
Part-Species1-VibCrossSection = 1e-19

Part-Species2-MWConstA-2-2 = 129.00
Part-Species2-MWConstA-2-1 = 115.10
Part-Species2-MWConstB-2-2 = -9.76
Part-Species2-MWConstB-2-1 = -6.92
Part-Species2-VibCrossSection = 1e-19
```

It is not possible to calculate an instantaneous vibrational relaxation probability with this model [44]. Thus, the probability is calculated for every collision and is averaged. To avoid large errors in cells containing only a few particles, a relaxation of this average probability is implemented. The relaxation factor α can be changed with the following parameter in the ini file:

```
Particles-DSMC-alpha = 0.99
```

The new probability is calculated with the vibrational relaxation probability of the n^{th} iteration P_{v}^{n}, the number of collision pairs n_{pair} and the average vibrational relaxation probability of the actual iteration $P_{\mathrm{v}}^{\mathrm{iter}}$.

$$P_{\mathrm{v}}^{n+1} = P_{\mathrm{v}}^{n} \cdot \alpha^{2 \cdot n_{\mathrm{pair}}} + (1 - \alpha^{2 \cdot n_{\mathrm{pair}}}) \cdot P_{\mathrm{v}}^{\mathrm{iter}}$$

This model is extended to more species by calculating a separate probability for each species. An initial vibrational relaxation probability is set by calculating $\mathrm{INT}(1/(1-\alpha))$ vibrational relaxation probabilities for each species and cell by using an instantaneous translational cell temperature.

Electronic Relaxation

For the modelling of electronic relaxation, three models are available: the model by Liechty et al. [45] and a BGK Landau-Teller like model [46], where each particle has a specific electronic state and the model by Burt and Eswar [47], where each particle has an electronic distribution function attached. The three models utilize tabulated energy levels, which can be found in literature for a wide range of species (e.g. for monatomic [48], diatomic [49], polyatomic [50] molecules). PICLas utilizes a species database, which contains the electronic energy levels of the species and is located in the top folder `SpeciesDatabase.h5`. Details regarding the database and the addition of new species can be found in Section *Unified Species Database*. To include electronic excitation in the simulation, the following parameters are required

```
Particles-DSMC-ElectronicModel  = 0     ! No electronic energy is considered (default)
                                = 1     ! Model by Liechty
                                = 2     ! Model by Burt
                                = 4     ! BGK Landau-Teller like model
Particles-DSMCElectronicDatabase = DSMCSpecies_electronic_state_full_Data.h5
```

In case of a large number of electronic levels, their number can be reduced by providing a relative merge tolerance. Levels those relative differences are below this parameter will be merged:

```
EpsMergeElectronicState = 1E-3
```

However, this option should be evaluated carefully based on the specific simulation case and tested against a zero/very low merge tolerance. Finally, the default relaxation probability of 0.01 can be adjusted by

```
Part-Species$-ElecRelaxProb = 0.3
```

Additionally, variable relaxation probabilities can be used. For each species where its value differs from the default relaxation probability, the following parameter needs to be defined

```
Part-Species3-ElecRelaxProb = 1.0
Part-Species4-ElecRelaxProb = 0.5
Part-Species5-ElecRelaxProb = 0.1
```

Chemistry & Ionization

Three chemistry models are currently available in PICLas

- Quantum Kinetic (QK)
- Total Collision Energy (TCE)
- Cross-section (XSec)

The model of each reaction can be chosen separately. If a collision pair has multiple reaction paths (e.g. CH3 + H, two possible dissociation reactions and a recombination), the reaction paths of QK and TCE are treated together, meaning that it is decided between those reaction paths. If a reaction path is selected, the reaction is performed and the following routines of the chemistry module are not performed. It is recommended not to combine cross-section based reaction paths with other reaction paths using QK/TCE for the same collision pair.

Chemical reactions and ionization processes require

```
Particles-DSMC-CollisMode = 3
```

The reactions paths can then be defined in the species parameter file. First, the number of reactions to read-in has to be defined

```
DSMC-NumOfReactions = 2
```

A reaction is then defined by

```
DSMC-Reaction1-ReactionModel = TCE
DSMC-Reaction1-Reactants     = (/1,1,0/)
DSMC-Reaction1-Products      = (/2,1,2,0/)
```

where the reaction model can be defined as follows

Model	Description
TCE	Total Collision Energy: Arrhenius-based chemistry
QK	Quantum Kinetic: Threshold-based chemistry
XSec	Cross-section based chemistry
phIon	Photo-ionization (e.g. N + ph -> N^+ + e)
phIonXSec	Photo-ionization (e.g. N + ph -> N^+ + e) with cross-section based chemistry

The reactants (left-hand side) and products (right-hand side) are defined by their respective species index. The photo-ionization reaction is a special case to model the ionization process within a defined volume by photon impact (see Section *Photo-ionization*). It should be noted that for the dissociation reaction, the first given species is the molecule to be dissociated. The second given species is the non-reacting partner, which can either be defined specifically or set to zero to define multiple possible collision partners. In the latter case, the number of non-reactive partners and their species have to be given by

```
DSMC-Reaction1-Reactants=(/1,0,0/)
DSMC-Reaction1-Products=(/2,0,2,0/)
DSMC-Reaction1-NumberOfNonReactives=3
DSMC-Reaction1-NonReactiveSpecies=(/1,2,3/)
```

This allows to define a single reaction for an arbitrary number of collision partners. In the following, three possibilities to model the reaction rates are presented.

Total Collision Energy (TCE)

The Total Collision Energy (TCE) model [51] utilizes Arrhenius type reaction rates to reproduce the probabilities for a chemical reaction. The extended Arrhenius equation is

$$k(T) = AT^b e^{-E_a/T}$$

where A is the prefactor ([1/s, m^3/s, m^6/s] depending on the reaction type), b the power factor and E_a the activation energy [K]. These parameters can be defined in PICLas as follows

```
DSMC-Reaction1-Arrhenius-Prefactor=6.170E-9
DSMC-Reaction1-Arrhenius-Powerfactor=-1.60
DSMC-Reaction1-Activation-Energy_K=113200.0
```

An example initialization file for a TCE-based chemistry model can be found in the regression tests (e.g. `regressioncheck/NIG_Reservoir/CHEM_EQUI_TCE_Air_5Spec`).

Quantum Kinetic Chemistry (QK)

The Quantum Kinetic (QK) model [52] chooses a different approach and models chemical reactions on the microscopic level. Currently, the QK model is only available for ionization and dissociation reactions. It is possible to utilize TCE- and QK-based reactions in the same simulation for different reactions paths for the same collision pair, such as the ionization and dissociation reactions paths (e.g. N_2 + e can lead to a dissociation with the TCE model and to an ionization with the QK model). An example setup can be found in the regression tests (e.g. `regressioncheck/NIG_Reservoir/CHEM_QK_multi-ionization_C_to_C6+`).

Besides the reaction model, reactants and products definition no further parameter are required for the reaction. However, the dissociation energy [eV] has to be defined on a species basis

```
Part-Species1-Ediss_eV = 4.53
```

The ionization threshold is determined from the last level of the previous state of the ionized product and thus requires the read-in of the electronic state database.

Cross-section Chemistry (XSec)

The cross-section based chemistry model utilizes experimentally measured or ab-initio calculated cross-sections (analogous to the collision probability procedure described in Section *Collision cross-sections*). It requires the same database, where the reaction paths are stored per particle pair, e.g. the `N2-electron` container contains the `REACTION` folder, which includes the reactions named by their products, e.g. `N2Ion1-electron-electron`.

If the defined reaction cannot be found in the database, the code will abort. It should be noted that this model is not limited to the utilization with MCC or a background gas and can be used with conventional DSMC as an alternative chemistry model. Here, the probability will be added to the collision probability to reproduce the reaction rate. Examples of the utilization of this model can be found in the regression tests (e.g. `regressioncheck/NIG_Reservoir/CHEM_RATES_XSec_Chem_H2-e`).

Backward Reaction Rates

Backward reaction rates can be calculated for any given forward reaction rate by using the equilibrium constant

$$K_{\mathrm{equi}} = \frac{k_{\mathrm{f}}}{k_{\mathrm{b}}}$$

where K_{equi} is calculated through partition functions. This option can be enabled for all given reaction paths in the first parameter file by

```
Particles-DSMC-BackwardReacRate = T
```

or it can be enabled for each reaction path separately in the species parameter file, e.g. to disable certain reaction paths or to treat the backward reaction directly with a given Arrhenius rate

```
DSMC-Reaction1-BackwardReac = T
```

It should be noted that if the backward reactions are enabled globally, a single backward reaction can be disabled by setting the reaction-specific flag to false and vice versa, if the global backward rates are disabled then a single backward reaction can be enabled. Since the partition functions are tabulated, a maximum temperature and the interval are required to define the temperature range which is expected during the simulation.

```
Particles-DSMC-PartitionMaxTemp = 120000.
Particles-DSMC-PartitionInterval = 20.
```

Should a collision temperature be outside of that range, the partition function will be calculated on the fly. Additional species-specific parameters are required in the species initialization file to calculate the rotational partition functions

```
Part-Species1-SymmetryFactor=2
! Linear poly- and diatomic molecules
Part-Species1-CharaTempRot=2.1
! Non-linear polyatomic molecules
Part-Species1-CharaTempRot1=2.1
Part-Species1-CharaTempRot2=2.1
Part-Species1-CharaTempRot3=2.1
```

The rotational symmetry factor depends on the symmetry point group of the molecule and can be found in e.g. Table 2 in [53]. While linear polyatomic and diatomic molecules require a single characteristic rotational temperature, three values have to be supplied for non-linear polyatomic molecules. Finally, electronic energy levels have to be supplied to consider the electronic partition function. For this purpose, the user should provide an electronic state database as presented in Section *Electronic Relaxation*.

Additional Features

Deletion of Chemistry Products

Specified product species can be deleted immediately after the reaction occurs, e.g. if an ionization process with a background gas is simulated and the neutral species as a result from dissociation are not of interest. To do so, the number of species to be deleted and there indices have to be defined

```
Particles-Chemistry-NumDeleteProducts = 2
Particles-Chemistry-DeleteProductsList = (/2,3/)
```

Ambipolar Diffusion

A simple ambipolar diffusion model in order to be able to ignore the self-induced electric fields, e.g. for the application in hypersonic re-entry flows, where ionization reactions cannot be neglected, can be enabled by

```
Particles-DSMC-AmbipolarDiffusion = T
```

Electrons are now attached to and move with the ions, although, they still have their own velocity vector and are part of the pairing and collisional process (including chemical reactions). The velocity vector of the ion species is not additionally corrected to account for the acceleration due to the self-induced electric fields. The restart from a state file without previously enabled ambipolar diffusion is currently not supported. However, the simulation can be continued if a macroscopic output is available with the macroscopic restart. In that case, the electrons are not inserted but paired with an ion and given the sampled velocity from the macroscopic restart file.

Ensuring Physical Simulation Results

To determine whether the DSMC related parameters are chosen correctly, so-called quality factors can be written out as part of the regular DSMC state file output by

```
Particles-DSMC-CalcQualityFactors = T
```

This flag writes out the spatial distribution of the mean and maximal collision probability (`DSMC_MeanCollProb` and `DSMC_MaxCollProb`). On the one hand, maximal collision probabilities above unity indicate that the time step should be reduced. On the other hand, very small collision probabilities mean that the time step can be further increased. Additionally, the ratio of the mean collision separation distance to the mean free path is written out (`DSMC_MCSoverMFP`)

$$\frac{l_{\mathrm{mcs}}}{\lambda} < 1$$

The mean collision separation distance is determined during every collision and compared to the mean free path, where its ratio should be less than unity. Values above unity indicate an insufficient particle discretization.

Additionaly, the above flag writes out the percentage of cells with a resolved timestep (`ResolvedTimestep`), the maximum collision probability of the entire computational domain (`Pmax`), the maximum of the `MCSoverMFP` of the entire domain (`MaxMCSoverMFP`), and the percentage of cells with a resolved time step and resolved weighting factor w (`ResolvedCellPercentage`) to the file `PartAnalyze.csv`. In case of a reservoir simulation, the mean collision probability (`Pmean`) is the output instead of the `ResolvedTimestep`.

In order to estimate the required weighting factor w, the following equation can be utilized for a 3D simulation

$$w < \frac{1}{\left(\sqrt{2}\pi d_{\mathrm{ref}}^2 n^{2/3}\right)^3},$$

where d_{ref} is the reference diameter and n the number density. Here, the largest number density within the simulation domain should be used as the worst-case. For supersonic/hypersonic flows, the conditions behind a normal shock can be utilized as a first guess. For a thruster/nozzle expansion simulation, the chamber or throat conditions are the limiting factor.

1.4.9 Background Gas

A constant or spatially varying background gas (single species or mixture) can be utilized to enable efficient particle collisions between the background gas and other particle species (represented by actual simulation particles). The assumption is that the density of the background gas n_{gas} is much greater than the density of the particle species, e.g. the charged species in a plasma, n_{charged}

$$n_{\mathrm{gas}} >> n_{\mathrm{charged}}.$$

Under this assumption, collisions within the particle species can be neglected and collisions between the background gas and particle species do not alter the background gas conditions. It can be activated by using the regular particle insertion parameters (as defined in Section *Initialization*, the `-Init1` construct can be neglected since no other initialization regions are allowed if the species is already defined a background species) and by defining the `SpaceIC` as `background` as well as the number density [1/m^3] as shown below

```
Part-Species1-SpaceIC     = background
Part-Species1-PartDensity = 1E+22
```

Other species parameters such as mass, charge, temperature and velocity distribution for the background are also defined by the regular read-in parameters. A mixture as a background gas can be simulated by simply defining multiple background species. Every time step particles are generated from the background gas (for a mixture, the species of the generated particle is chosen based on the species composition) and paired with the particle species. Subsequently, the collision probabilities are calculated using the conventional DSMC routines and the VHS cross-section model. Afterwards, the collision process is performed (if the probability is greater than a random number) and it is tested whether additional energy exchange and chemical reactions occur. While the VHS model is sufficient to model collisions between neutral species, it cannot reproduce the phenomena of a neutral-electron interaction. For this purpose, the cross-section based collision probabilities should be utilized, which are discussed in the following section.

Distribution from DSMC result

A spatially varying background gas distribution may be used by running a stand-alone DSMC simulation beforehand and using a time-averaged DSMC state file (*PROJECT_DSMCState_*.h5*) as input for the actual simulation by setting

```
Particles-BGGas-UseDistribution              = T
Particles-MacroscopicRestart-Filename        = DSMCResult.h5
Part-SpeciesX-InitX-BGG-Distribution-SpeciesIndex = 1
```

where the first parameter activates the background gas distribution and the second parameter supplies the relative path to the file from which the background gas density, velocity and temperature field is read (cell-constant values). The third parameter defines which species index within the DSMC file is to be used as it may contain multiple species.

Regions

Another possibility to define a non-constant background gas is available through the definition of regions. Multiple regions defined by simple geometrical volumes (e.g. cylinder) can be mapped to different species. First, one or more regions are defined:

```
Particles-BGGas-nRegions                  = 1
Particles-BGGas-Region1-Type              = cylinder
Particles-BGGas-Region1-RadiusIC          = 5E-6
Particles-BGGas-Region1-CylinderHeightIC  = 5E-6
Particles-BGGas-Region1-BasePointIC       = (/0.,0.,0./)
Particles-BGGas-Region1-BaseVector1IC     = (/1.,0.,0./)
Particles-BGGas-Region1-BaseVector2IC     = (/0.,1.,0./)
```

Here, a cylinder is defined by two base vectors (from which a normal is determined for the direction of the cylinder height), basepoint, radius and cylinder height. The definition of the species is the same as described above, with the addition of an additional parameter, defining in which region, these properties should be applied to:

```
Part-Species1-Init1-BGG-Region            = 1
```

While a species can be part of different regions through multiple inits and multiple species can be part of a single region, overlapping regions are not allowed. Whether an element is within a region is determined through the midpoint of the element and thus it does not have to be fully enveloped.

Trace species

If the number densities of the background gas species differ greatly and a specific background species is of interest (or the interaction with it) that has a lower number density compared to the other background species, it can be defined as a so-called trace species as shown below.

```
Part-vMPF                         = T
Part-Species1-Init1-TraceSpecies = T
```

The first flag enables the variables weighting factor feature in general (details about this feature can be found in Section *Variable Particle Weighting*). An additional flag defines the background gas species as a trace species, where multiple trace species can be defined. Finally, the weighting factors of the background species can be adopted to define the difference in the weighting factors.

```
Part-Species1-MacroParticleFactor = 1E2
Part-Species2-MacroParticleFactor = 1E4
```

Using the values above, each collision with the first background species will result in 100 collision tests using the simulation particle (ie. not the background species) and randomly generated background particles. Consequently, the number of samples for the trace species will be increased and simulation particles with the weighting factor of the trace background species will be introduced into the simulation.

Cross-section based collision probability

For modelling of particle collisions with the Particle-in-Cell method, often the Monte Carlo Collision (MCC) algorithm is utilized. Here, experimentally measured or ab-initio calculated cross-sections are typically utilized to determine the collision probability, based on the cross-section [m^2], the timestep [s], the particle velocity [m/s] and the target gas number density [m^{-3}]

$$P = 1 - e^{-\sigma v \Delta t n_{\mathrm{gas}}}.$$

In PICLas, the null collision method after [54],[55] is available, where the number of collision pairs is determined based a maximum collision frequency. Thus, the computational effort is reduced as not every particle has to be checked for a collision, such as in the previously described DSMC-based background gas. To activate the MCC procedure, the collision cross-sections have to be supplied via read-in from a database

```
Particles-CollXSec-Database = MCC_Database.h5
Particles-CollXSec-NullCollision = TRUE
```

Cross-section data can be retrieved from the LXCat database [56] and converted with a Python script provided in the tools folder: `piclas/tools/crosssection_database`. Details on how to create an own database with custom cross-section data is given in Section *Collision cross-sections*. Finally, the input which species should be treated with the MCC model is required

```
Part-Species2-SpeciesName = electron
Part-Species2-UseCollXSec = T
```

The read-in of the cross-section data is based on the provided species name and the species name of the background gas (e.g. if the background species name is Ar, the code will look for a container named `Ar-electron` in the MCC database). Finally, the cross-section based collision modelling (e.g. for neutral-charged collisions) and the VHS model (e.g. for neutral-neutral collisions) can be utilized within a simulation for different species.

Cross-section based vibrational relaxation probability

In the following, the utilization of cross-section data is extended to the determination of the vibrational relaxation probability. When data is available, it will be read-in by the Python script described above. If different vibrational levels are available, they will be summarized to a single relaxation probability. Afterwards the regular DSMC-based relaxation procedure will be performed. To enable the utilization of these levels, the following flag shall be supplied

```
Part-Species2-UseVibXSec = T
```

It should be noted that even if Species2 corresponds to an electron, the vibrational cross-section data will be read-in for any molecule-electron pair. If both species are molecular, priority will be given to the species utilizing this flag.

Cross-section based electronic relaxation probability

In the following, the utilization of cross-section data is extended to the electronic excitation for neutral-electron collisions. When data is available, it will be read-in by the Python script described above. Each level will be handled separately, allowing the atom/molecule to be excited in each level. The cross-section data will be used to determine whether and which excitation will occur. During the excitation procedure the energy of the atom/molecule will be set to respective level. To enable this model, the following flags are required

```
Particles-DSMC-ElectronicModel    = 3
Part-Species1-UseElecXSec         = T
```

The species-specific flag `UseElecXSec` should be set to `TRUE` for the heavy-species and not the electron species. Currently, this model is only implemented for the background gas, however, an extension to regular DSMC simulations is envisioned. It can be used with cross-section based as well as Variable Hard Sphere (VHS) collision modelling. The output of the relaxation rates is provided through

```
CalcRelaxProb = T
```

However, the electronic temperature is currently not added to the total temperature for the PartAnalyze.csv output. Additionally, the cell-local excitation rate [1/s] per electronic level of the respective species can be sampled and output as described in Section *Electronic excitation*.

1.4.10 Unified Species Database

A unified database of species data, electronic states, cross-sections, and chemistry models can be used as a more convenient alternative input for the simulations. The use of the database allows to reduce the length of the input files and ensures a consistent storage of the parameters, together with the respective reference. The database (`SpeciesDatabase.h5`) is provided at the top level of the PICLas directory. It contains over 40 of the most common species in PICLas simulations and the most common complex chemical reaction models with 300 single reactions. In addition, several cross-section models are included. This feature is still under development and the provided species & reaction data should be treated carefully as we are in process of verifying the data.

The data in the Unified Species Database is grouped, as shown in the following example:

```
Cross-Sections (group)
    H2-H2Ion1 (dataset)
Reaction (group)
    CH3_CH2+H (dataset)
        Chemistry model (attribute)
        Reaction model (attribute)
        Arrhenius parameters (attribute)
        Products    (attribute)
        Reactants   (attribute)
    O2+M_O+O+M (dataset)
        Chemistry model (attribute)
        Reaction model (attribute)
        Arrhenius parameters (attribute)
        Products    (attribute)
        Reactants   (attribute)
    Fe_FeIon1+electron (dataset)
        Chemistry model (attribute)
        Reaction model (attribute)
        Products    (attribute)
        Reactants   (attribute)
Species (group)
    H2 (group)
        Electronic levels (dataset)
        Species parameters (attribute)
    H2Ion1 (group)
        Electronic levels (dataset)
        Species parameters (attribute)
    electron (group)
        Electronic levels (dataset)
        Species parameters (attribute)
```

To read in data from a given species database, the database name must be specified in the `parameter.ini` file of the simulation

```
Particles-Species-Database = SpeciesDatabase.h5
```

When reading the database, all available parameters are taken directly from the database by default. Missing parameters can be added as usual in the `parameter.ini` or the `DSMC.ini` files. If no database is specified or the specified database is not available, the parameters are read from the regular parameter files as described in the sections above. All data available in the database and how to choose between different input forms is described in the following sections for the different data types available. An example where the given database is used can be found in `regressioncheck/NIG_Reservoir/CHEM_EQUI_TCE_Air_5Spec_Database`.

For instructions on extending the existing database or creating a new one, please refer to Chapter *Unified Species Database (USD)* for instructions.

Species data

To include a species in a simulation, it has to be selected by its name with the following command:

```
Part-Species1-SpeciesName = CH4
```

The database follows the PICLas nomenclature for atoms and molecules. Cations are given by

```
Part-Species4-SpeciesName = CH3Ion1
```

where the number after 'Ion' refers to the degree of ionization. The database contains general species data such as the mass, charge and number of atoms, as well as the VHS and VSS parameters. For molecules, rotational and vibrational frequencies can be given as well. Additionally, each species contains a dataset with the electronic energy levels. A complete list of the parameter and models is available in Section *Species Definition*. Per default, all available parameters are read from the database. While we are in the process of verifying the database, the table gives an overview over the already verified parameters and the respective data sources. In general, the source/reference will also be provided with the parameter in the database itself.

Parameter	Unit	Source (unless otherwise noted)
Heat of formation (at 298.15K)	Kelvin	Active Thermochemical Tables (ATcT)

It possible to revert to the regular parameter read-in from parameter files per species

```
Part-Species1-DoOverwriteParameters = true
```

If this flag is set, all parameters for this species need to be set manually and a separate electronic state database has to be provided:

```
Particles-DSMCElectronicDatabase = DSMCSpecies_electronic_state_full_Data.h5
```

Reaction data

The database contains different chemistry models including various reactions, utilizing the *Total Collision Energy (TCE)* model or *Quantum Kinetic Chemistry (QK)* model. Each reaction is assigned to a single or more chemistry models (N). This information is stored in the chemistry model attribute along with the non reactive species of this reaction. This attribute contains an array which either has the dimension (N,1) or (N,2), where the second column contains the non reactive species, if existent. For the TCE model, the Arrhenius prefactor, powerfactor and the activation energy are stored. For the QK model only dissociation energy of the molecule is required, which has been provided as a species parameter.

To utilize the read-in of reaction from the database, a chemistry model has to be defined. All reactions with this model are then read-in from the database and no additional parameter input is required:

```
DSMC-ChemistryModel = Titan_14Spec_24Reac_Gokcen2007
```

Name	Description	Spec	React.	Reference
Air_5Spec_8Reac	Air without ions	5	8	M. Laux. *Direkte Simulation verdünnter, reagierender Strömungen.* PhD Thesis, University of Stuttgart, 1995.
Air_5Spec_8Reac	Air without ions	8	8	C. Park. *Review of chemical-kinetic problems of future NASA missions. I - Earth entries.* Journal of Thermophysics and Heat Transfer, 7(3):385–398, 1993.
Air_11Spec_27R	Air with ions	11	27	C. Park. *Review of chemical-kinetic problems of future NASA missions. I - Earth entries.* Journal of Thermophysics and Heat Transfer, 7(3):385–398, 1993.
Air_11Spec_43R	Air with ions and backward reaction rates	11	43	E. Farbar and I. Boyd. *Simulation of Fire II Reentry Flow Using the Direct Simulation Monte Carlo Method.* 40th Thermophysics Conference, 2008.
Air_11Spec_51R	Air with ions	11	51	C. Park. *Review of chemical-kinetic problems of future NASA missions. I - Earth entries.* Journal of Thermophysics and Heat Transfer, 7(3):385–398, 1993.
Fe-in-Air_3Spec_7Reac	Outgassing of iron into air	3	7	Voronov. G. *A practical fit formula for ionization rate coefficients of atoms and ions by electron impact: Z= 1–28.* Atomic Data and Nuclear Data Tables, 65(1):1–35, 1997.
				J. M. Plane, W. Feng, and E. C. Dawkins. *The Mesosphere and Metals: Chemistry and Changes.* Chemical Reviews, 115(10):4497–4541, 2015.
CH4_7Spec_7Re		7	7	-
CH4-Ar_8Spec_7Reac		8	7	-
CO2_6Spec_10R		6	10	C. Johnston and A. Brandis. *Modeling of nonequilibrium CO Fourth-Positive and CN Violet emission in CO2–N2 gases.* Journal of Quantitative Spectroscopy and Radiative Transfer, 149:303–317, 2014.
Mars_11Spec_27	Mars without ions	11	27	C. Johnston and A. Brandis. *Modeling of nonequilibrium CO Fourth-Positive and CN Violet emission in CO2–N2 gases.* Journal of Quantitative Spectroscopy and Radiative Transfer, 149:303–317, 2014.
Mars_16Spec_31	Mars with ions	16	31	C. Park, J. T. Howe, R. L. Jaffe, and G. V. Candler. *Review of chemical-kinetic problems of future NASA missions. II - Mars entries.* Journal of Thermophysics and Heat Transfer, 8(1):9–23, 1994.
Mars_17Spec_42	Mars with ions and O2+	17	42	C. Johnston and A. Brandis. *Modeling of nonequilibrium CO Fourth-Positive and CN Violet emission in CO2–N2 gases.* Journal of Quantitative Spectroscopy and Radiative Transfer, 149:303–317, 2014.
Titan_14Spec_24Re	Titan without ions but with Argon	14	24	R. Savajano, R. Sobbia, M. Gaffuri, and P. Leyland. *Reduced Chemical Kinetic Model for Titan Entries.* International Journal of Chemical Engineering, vol. 2011, Article ID 970247, 2011.
				T. Gokcen. *N2-CH4-Ar Chemical Kinetic Model for Simulations of Atmospheric Entry to Titan.* Journal of Thermophysics and Heat Transfer, 21(1):9–18, 2007.
Titan_18Spec_30Re	Titan with ions but without Argon	18	30	T. Gokcen. *N2-CH4-Ar Chemical Kinetic Model for Simulations of Atmospheric Entry to Titan.* Journal of Thermophysics and Heat Transfer, 21(1):9–18, 2007.

Currently, these reactions cannot be utilized separately, but only as part of a chemistry model.

Cross-section data

The use of the unified species database for the cross-section data follows the description given in Section *Cross-section based collision probability* for collision and *Cross-section Chemistry (XSec)* for chemistry modelling, respectively. An example for is located in `regressioncheck/NIG_Reservoir/CHEM_RATES_XSec_Chem_H2_Plasma_Database`.

1.4.11 Fokker-Planck Collision Operator

The implementation of the FP-based collision operator is based on the publications by [57] and [58]. It is a method, which allows the simulation of gas flows in the continuum and transitional regime, where the DSMC method is computationally too expensive. The collision integral is hereby approximated by a drift and diffusion process

$$\left.\frac{\partial f}{\partial t}\right|_{\text{coll}} \approx -\sum_{i=1}^{3}\frac{\partial}{\partial v_i}(A_i f) + \frac{1}{2}\sum_{i=1}^{3}\sum_{j=1}^{3}\frac{\partial^2}{\partial v_i \partial v_j}(D_{ij} f),$$

where $\mathbf{A}$ is the drift vector and $\mathcal{D}$ the diffusion matrix.

The current implementation supports:

- 2 different methods: Cubic and Ellipsoidal Statistical (ES)
- Single species, monatomic and polyatomic gases
- Thermal non-equilibrium with rotational and vibrational excitation (continuous or quantized treatment)
- 2D/Axisymmetric simulations
- Variable time step (adaption of the distribution according to the maximal relaxation factor and linear scaling)

Relevant publications of the developers:

- Implementation of the cubic Fokker-Planck in PICLas [58]
- Comparison of the cubic and ellipsoidal statistical Fokker-Planck [59]
- Simulation of a nozzle expansion (including the pressure chamber) with ESBGK, ESFP and coupled ESBGK-DSMC, comparison to experimental measurements [60]

To enable the simulation with the FP module, the respective compiler setting has to be activated:

```
PICLAS_TIMEDISCMETHOD = FP-Flow
```

A parameter file and species initialization file is required, analogous to the DSMC setup. It is recommended to utilize a previous DSMC parameter file to ensure a complete simulation setup. To enable the simulation with the FP methods, select the Fokker-Planck method, cubic (=1) and ES (=2):

```
Particles-FP-CollModel = 2
```

The vibrational excitation can be controlled with the following flags, including the choice between continuous and quantized vibrational energy:

```
Particles-FP-DoVibRelaxation = T
Particles-FP-UseQuantVibEn   = T
```

An octree cell refinement until the given number of particles is reached can be utilized, which corresponds to an equal refinement in all three directions (x,y,z):

```
Particles-FP-DoCellAdaptation = T
Particles-FP-MinPartsPerCell  = 10
```

A coupled FP-DSMC simulation can be enabled, where the FP method will be utilized if the number density $[\mathrm{m}^{-3}]$ is above a certain value:

```
Particles-CoupledFPDSMC       = T
Particles-FP-DSMC-SwitchDens  = 1E22
```

The flag `Particles-DSMC-CalcQualityFactors` controls the output of quality factors such as mean/maximal relaxation factor (mean: average over a cell, max: maximal value within the octree), max rotational relaxation factor, which are defined as

$$\frac{\Delta t}{\tau} < 1,$$

where Δt is the chosen time step and $1/\tau$ the relaxation frequency. The time step should be chosen as such that the relaxation factors are below unity. The `FP_DSMC_Ratio` gives the percentage of the sampled time during which the FP model was utilized. In a couple FP-DSMC simulation this variable indicates the boundary between FP and DSMC. However, a value below 1 can occur for pure FP simulations due to low particle numbers, when an element is skipped. Additionally, the Prandtl number utilized by the ESFP model is given.

1.4.12 Bhatnagar-Gross-Krook Collision Operator

The implementation of the BGK-based collision operator is based on the publications by [61] and [46]. It allows the simulation of gas flows in the continuum and transitional regimes, where the DSMC method is computationally too expensive. The collision integral is hereby approximated by a relaxation process:

$$\left.\frac{\partial f}{\partial t}\right|_{\mathrm{coll}} \approx \nu(f^t - f),$$

where f^t is the target distribution function and ν the relaxation frequency.

The current implementation supports:

- Three different BGK methods (i.e. different target distribution functions): Ellipsoidal Statistical, Shakov, and standard BGK
- Single species: monatomic, diatomic, and polyatomic gases
- Gas mixtures with an arbitrary number of monatomic, diatomic, and polyatomic species
- Thermal non-equilibrium with rotational and vibrational excitation (continuous or quantized treatment)
- 2D axisymmetric simulations
- Variable time step (adaption of the distribution according to the maximal relaxation factor and linear scaling)

Relevant publications of the developers:

- Implementation and comparison of the ESBGK, SBGK, and Unified models in PICLas for atomic species [61]
- Extension of the modeling to diatomic species including quantized vibrational energy treatment, validation of ESBGK with the Mach 20 hypersonic flow measurements of the heat flux on a 70° cone [46]
- Simulation of a nozzle expansion (including the pressure chamber) with ESBGK, SBGK and coupled ESBGK-DSMC, comparison to experimental measurements [60],[62]
- Extension to polyatomic molecules, simulation of the carbon dioxide hypersonic flow around a flat-faced cylinder, comparison of ESBGK, SBGK and DSMC regarding the shock structure and heat flux [63]

- Implemention of Brull's multi-species modeling for monatomic gas mixtures using Wilke's mixture rules and collision integrals for the calculation of transport coefficients [64]
- Extension of the implementation of Brull's ESBGK multi-species model to diatomic gas mixtures using Wilke's mixture rules (under review)
- Extension of the ESBGK method to multi-species modeling of polyatomic molecules, based on the ESBGK models of Mathiaud, Mieussens, Pfeiffer, and Brull, and including internal energies with multiple vibrational degrees of freedom, using Wilke's mixture rules and collision integrals in comparison (under review)

To enable the simulation with the BGK module, the respective compiler setting has to be activated:

```
PICLAS_TIMEDISCMETHOD = BGK-Flow
```

A parameter file and species initialization file is required, analogous to the DSMC setup. It is recommended to utilize a previous DSMC parameter file to ensure a complete simulation setup. To enable the simulation with the BGK methods, select the BGK method, ES (= 1), Shakov (= 2), and standard BGK (= 3):

```
Particles-BGK-CollModel = 1
```

The **recommended method is ESBGK**. If the simulation contains a gas mixture, a choice for the determination of the transport coefficients is available. The first model uses Wilke's mixture rules (= 1) to calculate the gas mixture viscosity and thermal conductivity. The **recommended second model utilizes collision integrals** (derived for the VHS model, = 2) to calculate these mixture properties.

```
Particles-BGK-MixtureModel    = 2
```

The vibrational excitation can be controlled with the following flags, including the choice between continuous and quantized vibrational energy.

```
Particles-BGK-DoVibRelaxation = T
Particles-BGK-UseQuantVibEn   = T
```

An octree cell refinement can be utilized until the given number of particles is reached, which corresponds to an equal refinement in all three directions (x,y,z):

```
Particles-BGK-DoCellAdaptation = T
Particles-BGK-MinPartsPerCell  = 10
```

It is recommended to utilize at least between 7 and 10 particles per (sub)cell. To enable the cell refinement above a certain number density, the following option can be utilized:

```
Particles-BGK-SplittingDens = 1E23
```

A coupled BGK-DSMC simulation can be enabled, where the BGK method will be utilized if the number density $[\mathrm{m}^{-3}]$ is above a certain value:

```
Particles-CoupledBGKDSMC       = T
Particles-BGK-DSMC-SwitchDens  = 1E22
```

The flag `Particles-DSMC-CalcQualityFactors` controls the output of quality factors such as mean/maximum relaxation factor (mean: average over a cell, max: maximal value within the octree), max. rotational relaxation factor, which are defined as

$$\frac{\Delta t}{\tau} < 1,$$

where Δt is the chosen time step and $1/\tau$ the relaxation frequency. The time step should be chosen as such that the relaxation factors are below unity. The `BGK_DSMC_Ratio` gives the percentage of the sampled time during which the

BGK model was utilized. In a coupled BGK-DSMC simulation this variable indicates the boundary between BGK and DSMC. However, a value below 1 can occur for pure BGK simulations due to low particle numbers, when an element is skipped.

An option is available to utilize a moving average for the variables used in the calculation of the relaxation frequency:

```
Particles-BGK-MovingAverage = T
```

The purpose is to increase the sample size and reduce the noise for steady gas flows. For this, the factor f

```
Particles-BGK-MovingAverageFac = 0.01
```

between zero and one must be defined with which the old M^n and newly sampled moments M are weighted to define the moments for the next time step M^{n+1}:

$$M^{n+1} = fM + (1-f)M^n.$$

1.4.13 Features of the Particle Solver

This section describes common features, which are available to the particle-based methods such as PIC, DSMC, MCC, BGK and FP.

Macroscopic Restart

The so-called macroscopic restart, allows to restart the simulation by using an output file of a previous simulation run (the regular state file has still to be supplied). This enables to change the weighting factor, without beginning a new simulation.

```
Particles-MacroscopicRestart = T
Particles-MacroscopicRestart-Filename = Test_DSMCState.h5
```

The particle velocity distribution within the domain is then generated assuming a Maxwell-Boltzmann distribution, using the translational temperature per direction of each species per cell. The rotational and vibrational energy per species is initialized assuming an equilibrium distribution.

Variable Time Step

A spatially variable or species-specific time step (VTS) can be activated for steady-state simulations, where three options are currently available and described in the following:

- Distribution: use a simulation result to adapt the time step in order to resolve physical parameters (e.g. collision frequency)
- Linear scaling: use a linearly increasing/decreasing time step along a given direction
- Species-specific: every species can have its own time step

Distribution

The first option is to adapt the time step during a simulation restart based on certain parameters of the simulation such as maximal collision probability (DSMC), mean collision separation distance over mean free path (DSMC), maximal relaxation factor (BGK/FP) and particle number. This requires the read-in of a DSMC state file that includes DSMC quality factors (see Section *Ensuring Physical Simulation Results*).

```
Part-VariableTimeStep-Distribution = T
Part-VariableTimeStep-Distribution-Adapt = T
Part-VariableTimeStep-Distribution-MaxFactor = 1.0
Part-VariableTimeStep-Distribution-MinFactor = 0.1
Part-VariableTimeStep-Distribution-MinPartNum = 10          ! Optional
! DSMC only
Part-VariableTimeStep-Distribution-TargetMCSoverMFP = 0.3   ! Default = 0.25
Part-VariableTimeStep-Distribution-TargetMaxCollProb = 0.8  ! Default = 0.8
! BGK/FP only
Part-VariableTimeStep-Distribution-TargetMaxRelaxFactor = 0.8
! Restart from a given DSMC state file (Disable if no adaptation is performed!)
Particles-MacroscopicRestart = T
Particles-MacroscopicRestart-Filename = Test_DSMCState.h5
```

The second flag allows to enable/disable the adaptation of the time step distribution. Typically, a simulation would be performed until a steady-state (or close to it, e.g. the particle number is not increasing significantly anymore) is reached with a uniform time step. Then a restart with the above options would be performed, where the time step distribution is adapted using the DSMC output of the last simulation. Now, the user can decide to continue adapting the time step with the subsequent DSMC outputs (Note: Do not forget to update the DSMCState file name!) or to disable the adaptation and to continue the simulation with the distribution from the last simulation (the adapted particle time step is saved within the regular state file). The DSMC state file after a successful simulation run will contain the adapted time step (as a factor of the `ManualTimeStep`) as an additional output. The time step factor within the DSMC state is always given priority over the time step stored in the state file during the adaptation step.

The `MaxFactor` and `MinFactor` allow to limit the adapted time step within a range of $f_{\min}\Delta t$ and $f_{\max}\Delta t$. The time step adaptation can be used to increase the number of particles by defining a minimum particle number (e.g `MinPartNum` = 10, optional). For DSMC, the parameters `TargetMCSoverMFP` (ratio of the mean collision separation distance over mean free path) and `TargetMaxCollProb` (maximum collision probability) allow to modify the target values for the adaptation. For the BGK and FP methods, the time step can be adapted according to a target maximal relaxation frequency.

The last two flags enable to initialize the particles distribution from the given DSMC state file, using the macroscopic properties such as flow velocity, number density and temperature (see Section *Macroscopic Restart*). Strictly speaking, the VTS procedure only requires the `Filename` for the read-in of the aforementioned parameters, however, it is recommended to perform a macroscopic restart to initialize the correct particle number per cells. Otherwise, cells with a decreased/increased time step will require some time until the additional particles have reached/left the cell.

The time step adaptation can also be utilized in coupled BGK-DSMC simulations, where the time step will be adapted in both regions according to the respective criteria as the BGK factors are zero in the DSMC region and vice versa. Attention should be payed in the transitional region between BGK and DSMC, where the factors are potentially calculated for both methods. Here, the time step required to fulfil the maximal collision probability criteria will be utilized as it is the more stringent one.

Linear scaling

The second option is to use a linearly increasing time step along a given direction. This option does not require a restart or a previous simulation result. Currently, only the increase of the time step along the **x-direction** is implemented. With the start point and end point, the region in which the linear increase should be performed can be defined. To define the domain border as the end point in maximal x-direction, the vector `(/-99999.,0.0,0.0/)` should be supplied. Finally, the `ScaleFactor` defines the maximum time step increase towards the end point $\Delta t(x_{\mathrm{end}}) = f\Delta t$.

```
Part-VariableTimeStep-LinearScaling = T
Part-VariableTimeStep-ScaleFactor   = 2
Part-VariableTimeStep-Direction     =      (/1.0,0.0,0.0/)
Part-VariableTimeStep-StartPoint    =     (/-0.4,0.0,0.0/)
Part-VariableTimeStep-EndPoint      =  (/-99999.,0.0,0.0/)
```

Besides DSMC, the linear scaling is available for the BGK and FP method. Finally, specific options for 2D/axisymmetric simulations are discussed in Section *2D/Axisymmetric Simulations*

Species-specific time step

This option is decoupled from the other two time step options as the time step is not applied on a per-particle basis but for each species. Currently, its main application is for PIC-MCC simulations (only Poisson field solver with Euler, Leapfrog and Boris-Leapfrog time discretization methods), where there are large differences in the time scales (e.g. electron movement requires a time step of several orders of magnitude smaller than for the ions). The species-specific time step is actvitated per species by setting a factor

```
Part-Species1-TimeStepFactor = 0.01
```

that is multiplied with the provided time step. If no time step factor is provided, the default time step will be utilized. In this example, the species will be effectively simulated with a time step 100 smaller than the given time step.

To accelerate the convergence to steady-state, the following flag can be used to perform collisions and reactions at the regular time step.

```
Part-VariableTimeStep-DisableForMCC = T
```

For species with a time step factor lower than 1, it is compared with a random number to decide whether the collision/reaction is performed for that species.

Symmetric Simulations

For one-dimensional (e.g. shock-tubes), two-dimensional (e.g. cylinder) and axisymmetric (e.g. re-entry capsules) cases, the computational effort can be greatly reduced.

1D Simulations

To enable one-dimensional simulations, the symmetry order has to be set

```
Particles-Symmetry-Order=1
```

The calculation is performed along the x-axis. The y and z dimension should be centered to the xz-plane (i.e. $|y_{\min}| = |y_{\max}|$). All sides of the hexahedrons must be parallel to the xy-, xz-, and yz-plane. Boundaries in y and z direction shall be defined as 'symmetric'.

```
Part-Boundary5-SourceName=SYM
Part-Boundary5-Condition=symmetric
```

2D/Axisymmetric Simulations

To enable two-dimensional simulations, the symmetry order has to be set

```
Particles-Symmetry-Order=2
```

Two-dimensional and axisymmetric simulations require a mesh in the xy-plane, where the x-axis is the rotational axis and y ranges from zero to a positive value. Additionally, the mesh shall be centered around zero in the z-direction with a single cell row, such as that $|z_{\min}| = |z_{\max}|$. The rotational symmetry axis shall be defined as a separate boundary with the `symmetric_axis` boundary condition

```
Part-Boundary4-SourceName=SYMAXIS
Part-Boundary4-Condition=symmetric_axis
```

The boundaries (or a single boundary definition for both boundary sides) in the z-direction should be defined as symmetry sides with the `symmetric` condition

```
Part-Boundary5-SourceName=SYM
Part-Boundary5-Condition=symmetric
```

It should be noted that the two-dimensional mesh assumes a length of $\Delta z = 1$, regardless of the actual dimension in z. Therefore, the weighting factor should be adapted accordingly.

To enable axisymmetric simulations, the following flag is required

```
Particles-SymmetryAxisymmetric=T
```

To fully exploit rotational symmetry, a radial weighting can be enabled, which will linearly increase the weighting factor w towards $y_{\max}$ (i.e. the domain border in y-direction), depending on the current y-position of the particle.

```
Particles-RadialWeighting=T
Particles-RadialWeighting-PartScaleFactor=100
```

A radial weighting factor of 100 means that the weighting factor at $y_{\max}$ will be $100w$. Although greatly reducing the number of particles, this introduces the need to delete and create (in the following "clone") particles, which travel

upwards and downwards in the y-direction, respectively. If the new weighting factor is smaller than the previous one, a cloning probability is calculated by

$$P_{\mathrm{clone}} = \frac{w_{\mathrm{old}}}{w_{\mathrm{new}}} - \mathrm{INT}\left(\frac{w_{\mathrm{old}}}{w_{\mathrm{new}}}\right) \qquad \text{for} \quad w_{\mathrm{new}} < w_{\mathrm{old}}.$$

For the deletion process, a deletion probability is calculated, if the new weighting factor is greater than the previous

$$P_{\mathrm{delete}} = 1 - P_{\mathrm{clone}} \qquad \text{for} \quad w_{\mathrm{old}} < w_{\mathrm{new}}.$$

If the ratio between the old and the new weighting factor is $w_{\mathrm{old}}/w_{\mathrm{new}} > 2$, the time step or the radial weighting factor should be reduced as the creation of more than one clone per particle per time step is not allowed. The same applies if the deletion probability is above 0.5.

For the cloning procedure, two methods are implemented, where the information of the particles to be cloned are stored for a given number of iterations (`CloneDelay=10`) and inserted at the old position. The difference is whether the list is inserted chronologically (`CloneMode=1`) or randomly (`CloneMode=2`) after the first number of delay iterations.

```
Particles-RadialWeighting-CloneMode=2
Particles-RadialWeighting-CloneDelay=10
```

This serves the purpose to avoid the so-called particle avalanche phenomenon [65], where clones travel on the exactly same path as the original in the direction of a decreasing weight. They have a zero relative velocity (due to the same velocity vector) and thus a collision probability of zero. Combined with the nearest neighbor pairing, this would lead to an ever-increasing number of identical particles travelling on the same path. An indicator how often identical particle pairs are encountered per time step during collisions is given as an output (`2D_IdenticalParticles`, to enable the output see Section *Ensuring Physical Simulation Results*). Additionally, it should be noted that a large delay of the clone insertion might be problematic for time-accurate simulations. However, for the most cases, values for the clone delay between 2 and 10 should be sufficient to avoid the avalance phenomenon.

Another issue is the particle emission on large sides in y-dimension close to the rotational axis. As particles are inserted linearly along the y-direction of the side, a higher number density is inserted closer to the axis. This effect is directly visible in the free-stream in the cells downstream, when using mortar elements, or in the heat flux (unrealistic peak) close to the rotational axis. It can be avoided by splitting the surface flux emission side into multiple subsides with the following flag (default value is 20)

```
Particles-RadialWeighting-SurfFluxSubSides = 20
```

An alternative to the particle position-based weighting is the cell-local radial weighting, which can be enabled by

```
Particles-RadialWeighting-CellLocalWeighting = T
```

However, this method is not preferable if the cell dimensions in y-direction are large, resulting in numerical artifacts due to the clustered cloning processes at cell boundaries.

Variable Time Step: Linear scaling

The linear scaling of the variable time step is implemented slightly different to the 3D case. Here, a particle-based time step is used, where the time step of the particle is determined on its current position. The first scaling is applied in the radial direction, where the time step is increased towards the radial domain border. Thus, $\Delta t(y_{\mathrm{max}}) = f\Delta t$ and $\Delta t(y_{\mathrm{min}} = 0) = \Delta t$.

```
Part-VariableTimeStep-LinearScaling = T
Part-VariableTimeStep-ScaleFactor = 2
```

Additionally, the time step can be varied along the x-direction by defining a "stagnation" point, towards which the time step is decreased from the minimum x-coordinate ($\Delta t(x_{\min}) = f_{\text{front}}\Delta t$) and away from which the time step is increased again towards the maximum x-coordinate ($\Delta t(x_{\max}) = f_{\text{back}}\Delta t$). Therefore, only at the stagnation point, the time step defined during the initialization is used.

```
Part-VariableTimeStep-Use2DFunction = T
Part-VariableTimeStep-StagnationPoint = 0.0
Part-VariableTimeStep-ScaleFactor2DFront = 2.0
Part-VariableTimeStep-ScaleFactor2DBack = 2.0
```

Variable Particle Weighting

Variable particle weighting is currently supported for PIC (with and without background gas) or a background gas (an additional trace species feature is described in Section *Background Gas*). The general functionality can be enabled with the following flag:

```
Part-vMPF                          = T
```

The split and merge algorithm is called at the end of every time step. In order to manipulate the number of particles per species per cell, merge and split thresholds can be defined as is shown in the following.

```
Part-Species2-vMPFMergeThreshold    = 100
```

The merge routine randomly deletes particles until the desired number of particles is reached and the weighting factor is adopted accordingly. Afterwards, the particle velocities v_i are scaled in order to ensure momentum and energy conservation with

$$\alpha = \frac{E_{\text{trans}}^{\text{old}}}{E_{\text{trans}}^{\text{new}}},$$

$$v_i^{\text{new}} = v_{\text{bulk}} + \alpha(v_i - v_{\text{bulk}}^{\text{new}}).$$

Internal degrees of freedom are conserved analogously. In the case of quantized energy treatment (for vibrational and electronic excitation), energy is only conserved over time, where the energy difference (per species and energy type) in each time step due to quantized energy levels is stored and accounted for in the next merge process.

```
Part-Species2-vMPFSplitThreshold    = 10
```

The split routine clones particles until the desired number of particles is reached. The algorithm randomly chooses a particle, clones it and assigns the half weighting factor to both particles. If the resulting weighting factor would drop below a specific limit that is defined by

```
Part-vMPFSplitLimit = 1.0 ! default value is 1
```

the split is stopped and the desired number of particles will not be reached. The basic functionality of both routines is verified during the nightly regression testing in `piclas/regressioncheck/NIG_code_analyze/vMPF_SplitAndMerge_Reservoir`.

Virtual Cell Merge

In the case of very complex geometries, very small cells can sometimes be created to represent the geometry, in which, however, only very few particles are present. This results in a very large statistical noise in these cells and the collision process is only inadequately covered due to the poor statistics. In this case, cells can be virtually merged with neighbouring cells during the runtime using predefined parameters. The merged cells are then treated as one large cell. This means that DSMC collision pairings and the BGK relaxation process are performed with all particles within the merged cell, i.e. all particles of the individual cells. The averaging then also takes place for all particles in the merged cell and all individual cells then receive the values of the merged cell in the output. The virtual cell merge can be enabled with the following flag:

```
Part-DoVirtualCellMerge             = T
```

Currently, only merging based on the number of particles within the cell is implemented. The minimum number of particles below which the cell is merged is defined with the following parameter:

```
Part-MinPartNumCellMerge            = 3
```

Furthermore, the spread or aggressiveness of the merge algorithm can be changed, i.e. how deep the merge extends into the mesh starting from each cell. 0 is the least aggressive merge, 3 the most aggressive merge.

```
Part-CellMergeSpread                = 0
```

There is also the possibility to define a maximum number of cells that can be merged. In this way, a desired "resolution" of the virtual cells can be achieved.

```
Part-MaxNumbCellsMerge              = 5
```

1.4.14 Radiation

Radiation Coupling

To account for high temperature gas radiation, the DSMC method can be coupled with a radiation module [66, 67, 68]. For a deeper physical knowledge of the implemented options see Ref. [66]. The radiation module calculates cell-local emission and absorption coefficients on the same computational mesh as the flow field solver using a line-by-line (LBL) method, and the radiative transfer solver simulates the radiative energy transfer through the computational domain using a photon Monte Carlo (pMC) solver. The LBL solver can also be used in a stand-alone version. The enitre radiation simulation can be run highly parallel using a MPI-shared memory concept (MPI 3.0). The LBL solver uses a domain decomposition and the pMC solver distributes the photon bundles across the different cores. The radiation module can be enabled by compiling PICLas with the following parameter

```
PICLAS_TIMEDISCMETHOD = Radiation
```

In addition, the following parameter must be set for larger computational meshes

```
PICLAS_INKIND8 = ON
```

For the radiation module, the following options are available and must be set accordingly

```
Radiation-RadType = 1   ! Full radiation module with radiative energy transfer
                    2   ! Blackbody radiation in entire computational domain
                    3   ! Emission and absorption coefficients only (stand-alone␣
↪radiation solver)
                    4   ! Shock tube mode
```

To define the species which are used in the radiation simulation, the following parameters must be set in one single ini-file (as an example, atomic nitrogen is chosen)

```
Part-Species1-MassIC                = 2.32600E-26 ! N Molecular Mass
Part-Species1-MacroParticleFactor   = 2.5E10

Part-Species1-SpeciesName           = N
Part-Species1-InteractionID         = 1
Part-Species1-Tref                  = 273     ! in K
Part-Species1-dref                  = 3.0E-10 ! in m
Part-Species1-omega                 = 0.24

Part-Species1-RadiationIonizationEn = 117345  ! Ionization Energy, cm-1
Part-Species1-RadiationRadius_A     = 0.7     ! Radius, A
Part-Species1-Starkex               = 0.33    ! Stark Index
Part-Species1-NuclCharge            = 1       ! Charge (1:neutral particles, 2:ions)
Radiation-Species1-SpectraFileName  = Ni.dat
```

The `Radiation-Species[$]-SpectraFileName` contains information about level energy and radiative transition lines. An example is given in the regression-tests and can e.g. be built using information available in the NIST database. For convenience, the calculation of each species can be disabled with

```
Part-Species[$]-DoRadiation = F
```

In a first step of the tool chain, the local emission and absorption coefficients are determined on the same computational mesh used by the flow field solver. Therefore, the same mesh has to be read in. If the emission in a single cell is to be calculated (`Radiation-RadType = 1`), a mesh file containing one single cell must be provided.

Different radiation mechanisms can be considered, depending on the energy state of the involved electron

```
Radiation-bb-atoms     = T   ! atomic line radiation (bound-bound)
Radiation-bb-molecules = T   ! molecular band radiation (bound-bound)
Radiation-bf           = T   ! recombination radiation (bound-free)
Radiation-ff           = T   ! Bremsstrahlung (free-free)
```

If the complete radiation module (1) is selected, the flow field information (DSMCState) previously calculated with PICLas is used as input by setting the following parameters

```
Radiation-MacroRadInput = T
Radiation-MacroInput-Filename = PROJECTNAME_DSMCState_000.001000000.h5
```

Often, radiation solvers use the translational temperature of the electrons as T_{E} to determine upper state densities. However, PICLas also offers the possibility to use the actual electronic excitation temperature

```
Radiation-UseElectronicExcitation = T
```

For a single cell (`Radiation-RadType = 1`), temperatures and densities can be given by

```
Part-Species[$]-RadiationTtrans  = 10000.
Part-Species[$]-RadiationTelec   = 10000.
Part-Species[$]-RadiationTvib    = 10000.
Part-Species[$]-RadiationTrot    = 10000.
Part-Species[$]-RadiationNumDens = 1E20.
```

The units are Kelvin and m^-3. For electrons, they are red in by

```
Radiation-TElectrons             = 10000.
Radiation-NumDensElectrons       = 1E20.
```

The wavelength range and the discretization for the simulation can be set with the following parameters. Radiative transitions outside of this range will not be considered in the simulation!

```
Radiation-MinWaveLen   = 200.    ! minimum wavelength in nanometers
Radiation-MaxWaveLen   = 1000.   ! maximum wavelength in nanometers
Radiation-WaveLenDiscr = 500000  ! Number of spectral discretization points
```

To save computational memory, wavelengths can also be spectrally binned together.

```
Radiation-WaveLenReductionFactor       = 10
```

For `Radiation-RadType = 3`, the next step is to determine the radiative energy transfer through the computational domain (change in radiation intensity due to the emission and absorption of the surrounding gas). This is done with a photon Monte Carlo method, where photon bundles are traced through the computational domain by solving the radiative transfer equation. Due to the time scales involved, only the steady-state solution is used, furthermore, scattering effects within the gas are neglected. For shock tubes (`Radiation-RadType = 4`), a simplified version of the radiative energy transfer along a tanget slab is calculated. The required diameter of the shock tube can be set (in meters) with

```
Radiation-ShockTubeDiameter = 0.16
```

For the radiative energy transfer of `Radiation-RadType = 3`, the number of photon bundles in each computational cell is set by

```
Radiation-NumPhotonsPerCell = 200
```

Instead of placing the same number of photons in each computational cell and giving them different energies, it is also possible to give them the same energy and redistribute them across the computational domain according to the emitted energy.

```
Radiation-AdaptivePhotonNumEmission = T   ! true:photons have the same energy,␣
↪false:number of photons per cell is equal
```

The initial properties of the photon bundles are set randomly. However, if a more uniform distribution within a cell is desired, different options are available. For the position, they are placed in the cell using a 2,3-Halton sequence. Additionally, their directions can be distributed along a spiral configuration

```
Radiation-PhotonPosModel = 2 ! 1:random 2:Halton
Radiation-DirectionModel = 2 ! 1:random 2:spiral
```

The absorption along their paths can be calculated (1) analytically or (2) stochastically, however, it is strongly recommended to do it analytically

```
Radiation-AbsorptionModel = 1
```

To determine the wavelength of each photon bundle, two different methods are implemented, (1) an acceptance-rejection method and (2) a bisection method

```
Radiation-PhotonWaveLengthModel = 1   ! 1:Acceptance-Rejection 2:Bisection
```

The acceptance-rejection method is more computationally intensive than the bisection method. However, it should be more accurate, because it consideres individual wavelengths rather than integral values. Thus, the acceptance-rejection

method can never select a wavelength that has no emission, while the bisection method has a very low probability of doing so.

For surface properties for photons on a reflecting wall, different options are possible. They can either be specularly reflected

```
Part-Boundary3-Condition=reflective
Part-Boundary3-PhotonSpecularReflection = T
```

or diffuse. If they are reflected diffusely, the energy accommodation coefficient can bet set ([0,1]) depending on the surface properties.

```
Part-Boundary3-Condition=reflective
Part-Boundary3-PhotonSpecularReflection = F
Part-Boundary3-PhotonEnACC = 0.5
```

In addition, PICLas offers the possibility to sample spectra on a virtual sensor and to compare them with measured data. Two different options are available: (1) with a viewing angle and (2) along a line-of-sight (LOS)

```
Radiation-RadObservationPointMethod    = 2   !1:observation angle, 2:LOS
```

The location of the virtual sensor can be defined with

```
Radiation-ObservationMidPoint          = (/-1.0,1E-5,0.0/)
```

and the view direction with

```
Radiation-ObservationViewDirection     = (/1.0,0.0,0.0/)
```

Its diameter can be set with

```
Radiation-ObservationDiameter          = 0.1
```

and the viewing angle with

```
Radiation-ObservationAngularAperture   = 0.404533
```

Along a LOS, it is also possible to use as many photons per cell as wavelength discretizations. These photons then receive the actual energy of the corresponding wavelength in order to obtain a perfect sampling of the corresponding energies for spectral comparisons.

```
Radiation-ObservationCalcFullSpectra   = T
```

To simulate with a high resolution and to match the units of the radiance, the output can be expressed in different units, e.g. to have a spectral discretization of 1/A and to get the radiance in /nm, the following parameters must be set

```
Radiation-WaveLenReductionFactorOutput = 10
```

To account for instrumental broadening, the radiance profile can be mathematically convolved with a trapezoid

```
Radiation-ObservationDoConvolution = T
```

The trapezoid can be defined by a topwidth and a basewidth, both read-in in angstroms

```
Radiation-ObservationSlitFunction      = (/1.7,3.42/)
```

The simulations can also be run on a two-dimensional rotationally symmetric mesh. To do this, the following options must be set. Different tracking routines are used than with an axisymmetric particle solver, therefore, the RadialWeighting-PartScaleFactor can have different values

```
Particles-Symmetry2D                          = T
Particles-Symmetry2DAxisymmetric              = T
Particles-RadialWeighting                     = T
Particles-RadialWeighting-PartScaleFactor     = 10000
Particles-RadialWeighting-CloneMode           = 2
Particles-RadialWeighting-CloneDelay          = 6
Particles-RadialWeighting-CellLocalWeighting = F
```

Raytracing

In addition to the radiation coupling, a ray tracing model is implemented. A boundary must be defined from which rays or photons are emitted in a preliminary step, which are tracked throughout the domain until they are absorbed at a boundary. The volumes and surface elements are sampled by passing photons and from this information the ionization within each volume element and secondary electron emission from each surface is calculated in the actual plasma simulation. The output of the sampling procedure can be viewed as irradiated volumes and surfaces and is written to the output files EUV_RadiationVolState.h5 and EUV_RadiationSurfState.h5, which can be converted to .vtk format with piclas2vtk. Raytracing is activated with

```
UseRayTracing = T
```

It only requires only one single section in the parameter.ini file. The user must specify a single rectangular and planar particle-boundary (here with index 5)

```
RayTracing-PartBound = 5
```

from which the number of rays

```
RayTracing-NumRays = 200000000
```

shall be emitted in the direction

```
RayTracing-RayDirection = (/0.0, 0.0, -1.0/)
```

Currently, all coordinates of this boundary must have the same z-coordinate and it must extend into the complete domain into the x- and y-direction. The parameter

```
RayTracing-PulseDuration = 1e-9
```

defines the pulse duration τ that defines the temporal shape of the light intensity function $I \propto \exp(-(t/\tau)^2)$ in [s].

```
RayTracing-NbrOfPulses
```

defines the number of pulses that are performed and

```
RayTracing-WaistRadius
```

the waist radius w_b that defines the spatial intensity via $I \propto \exp(-(r/w_b)^2)$ in [m]. The wavelength in [m] is given by

```
RayTracing-WaveLength = 50e-9
```

and the repetition rate of the pulses in [Hz] by

```
RayTracing-RepetitionRate = 2e3
```

The time-averaged (over one pulse) power density of the pulse in [W/m2] is used in

```
RayTracing-PowerDensity = 1e3
```

which is converted to an average pulse energy considering the irradiated area, hence, the same parameter can be used for the quarter and the full mesh setups.

To account for the reflectivity of specific surfaces, the absorption rate $A_\nu = 1 - R$ (R is the reflectivity) for photons must be supplied for each particle boundary. This is done by setting the parameter

```
Part-Boundary1-PhotonEnACC = 1.0
```

to a value between zero and unity. Additionally, it is possibly to switch between perfect angular reflection and diffuse reflection for each boundary.

```
Part-Boundary$-PhotonSpecularReflection = T
```

The parameter

```
RayTracing-ForceAbsorption=T
```

activates sampling of photons on surfaces independent of what happens to them there. They might be reflected or absorbed. If this parameter is set to `false``, then only absorbed photons will be sampled on surfaces. By also sampling reflected photons, the statistic is improved, hence, it should always be activated.

The angle under which photons are emitted from the particle-boundary is calculated from the normal vector of the boundary and the parameter

```
RayTracing-RayDirection
```

Because the interaction of every ray with every volume and surface element in three dimensions would lead to an unfeasible amount of memory usage if stored on the hard drive, the calculated volume and surface intersections need to be agglomerated in such a way that the details of the irradiated geometry are preserved. One goal is to have a clean cut between shadowed and illuminated regions where this interface cuts through surface of volume elements and without the need for a cut-cell method that splits the actual mesh elements. This is achieved by introducing a super-sampling method in the volume as well as on the surface elements. For the volumetric sampling, the originally cell-constant value is distributed among the volume sub-cells depending on a element-specific number of sub cells $n_{cells} = (N_{cell}+1)^3$, where N_{cell} is the polynomial degree in each element used for visualization of the super-sampled ray tracing solution. The polynomial degree N_{cell} is chosen between unity and a user-defined value, which can be automatically selected depending on the different criteria

```
RayTracing-VolRefineMode = 0 ! 0: do nothing (default)
                             ! 1: refine below user-defined z-coordinate with NMax
                             ! 2: scale N according to the mesh element volume between␣
↪NMin>=1 and NMax>=2
                             ! 3: refine below user-defined z-coordinate and scale N␣
↪according to the mesh element volume between NMin>=1 and NMax>=2 (consider only␣
↪elements below the user-defined z-coordinate for the scaling)
```

The maximum polynomial degree within refined volume elements for photon tracking (p-adaption) can hereby be set using

```
RayTracing-NMax = 1
```

In contrast to the volume super-sampling, only one global parameter is used to refine the all surfaces for sampling. Each surface can be split into n^2 sub-surfaces on which the sampling is performed via the parameter

```
RayTracing-nSurfSample = 2
```

The surfaces (quadrilaterals) are therefore equidistantly divided at the midpoint of each edge to create approximately equal-sized sub-surfaces.

The goal of PICLas is to enable to approximation of the complete Boltzmann equation:

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \frac{\partial f}{\partial \mathbf{x}} + \frac{\mathbf{F}}{m} \cdot \frac{\partial f}{\partial \mathbf{v}} = \left.\frac{\partial f}{\partial t}\right|_{\text{coll}}$$

1.5 Visualization & Output

In general, simulation results are either available as particle data, spatially resolved variables based on the mesh (classic CFD results) and/or as integral values (e.g. for reservoir/heat bath simulations). The piclas2vtk tool converts the HDF5 files generated by **PICLas** to the binary VTK format, readable by many visualization tools like ParaView and VisIt. The tool is executed by

```
piclas2vtk [posti.ini] output.h5
```

Multiple HDF5 files can be passed to the piclas2vtk tool at once. The (optional) runtime parameters to be set in `posti.ini` are given below:

Option	Default	Description
`NVisu`	1	Number of points at which solution is sampled for visualization
`VisuParticle`	OFF	Converts the particle data (positions, velocity, species, internal energies)
`NodeTypeVisu`	VISU	Node type of the visualization basis: VISU,GAUSS,GAUSS-LOBATTO,CHEBYSHEV-GAUSS-LOBATTO

In the following, the parameters enabling the different output variables are described. It should be noted that these parameters are part of the `parameter.ini` required by **PICLas**.

1.5.1 Particle Data

At each `Analyze_dt` as well as at the start and end of the simulation, the state file (`*_State_*.h5`) is written out, which contains the complete particle information such as their positions, velocity vector, species and internal energy values.

To sample the particles impinging on a certain surface between `Analyze_dt` outputs, the following option can be enabled per boundary condition

```
Part-Boundary1-BoundaryParticleOutput = T
```

The particle data will then be written to `*_PartStateBoundary_*.h5` and includes besides the position, velocity vector and kinetic energy (in eV), additionally the impact obliqueness angle between particle trajectory and surface normal vector, e.g. an impact vector perpendicular to the surface corresponds to an impact angle of $0°$. This allows you to create a histogram of the particle impacts in the post-processing.

The output of lost particles in a separate `*_PartStateLost*.h5` file can be enabled by

```
CountNbrOfLostParts = T
```

It includes particles lost during the tracking (TrackingMethod = triatracking, tracing) as well as during the restart procedure. For the latter, the output includes particles that went missing but were found on other processors.

1.5.2 Field Solver and PIC

The following table summarizes the available fields in connection with Poisson's or Maxwell's equations (some require a particle solver to be active) that can be stored to the state file (`*_State_*.h5`) file at every `Analyze_dt` as well as at the start and end of the simulation

Option	Default	Description	Poisson	Maxwell
`PIC-OutputSource`	F	charge ρ and current density j	yes	yes
`CalcElectricTimeDerivative`	F	time derivative $\frac{\partial D}{\partial t} = \varepsilon_r \varepsilon_0 \frac{\partial E}{\partial t}$	yes	no
`CalcPotentialEnergy`	F	potential field energy of the EM field	yes	yes

Charge and current density: When running a PIC simulation, the particle-grid deposited properties, such as charge and current densities (in each direction `x, y,`and `z`) can be output by enabling

```
PIC-OutputSource = T
```

that stores the data in the same format as the solution polynomial of degree N, i.e., $(N+1)^3$ data points for each cell.

Displacement current: The temporal change of the electric displacement field $\frac{\partial D}{\partial t} = \varepsilon_r \varepsilon_0 \frac{\partial E}{\partial t}$ can be stored for Poisson's equation (`PICLAS_EQNSYSNAME=poisson`) by setting

```
CalcElectricTimeDerivative = T
```

Again, the data in the same format as the solution polynomial of degree N, i.e., $(N+1)^3$ data points for each cell in the container `DG_TimeDerivative` in the `*_State_*.h5` file and can be converted to `.vtk` format with `piclas2vtk`. Furthermore, the integrated displacement current that traverses each field boundary (except periodic BCs) can be analyzed over time and is written to `FieldAnalyze.csv`, for details see Section *Field Variables*.

Potential field energy: The global energy that is stored within the electric and the magnetic field can be analyzed over time by setting

```
CalcPotentialEnergy = T
```

and is written to `FieldAnalyze.csv`, for details see Section *Field Variables*.

Element-polynomial field properties

In general, the data is the same format as the solution polynomial of degree N, i.e., $(N+1)^3$ data points for each cell in the respective container in the `.h5` files and can be converted to `.vtk` format with `piclas2vtk`. The resolution of the converted data can be adjusted by setting `NVisu` to any integer value, which is the used for the interpolation of the original data.

The following table summarizes the available fields in connection with Poisson's or Maxwell's equations

Option	Default	Description	Poisson	Maxwell
`CalcEMFieldOutput`	F	external electric **E** and magnetic **B** field	yes	yes

External electromagnetic field vector When using external electromagnetic fields, either via a constant vector

```
PIC-externalField = (/ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 /)
```

or a cell-local polynomial distribution calculated via `superB` by

```
PIC-BGFileName = BGField.h5
```

the resulting electromagnetic field, that is used by the PIC solver in addition to the calculated fields (superposition), can be visualized via

```
CalcEMFieldOutput = T
```

and the resulting data (vector fields for E and B) is stored in `PROJECT_PIC-EMField.h5`.

Element-constant field/particle properties

The determined properties are given by a single value within each cell and are NOT sampled over time as opposed to the output described in Section *Particle Flow and Surface Sampling*. These parameters are only available for PIC simulations, are part of the regular state file (as a separate container within the HDF5 file) and automatically included in the conversion to the VTK format. They are written to `PROJECT_Solution_000.000*.h5` and after conversion are found in `PROJECT_ElemData_000.000*.vtu`.

The following table summarizes the available fields in connection with Poisson's or Maxwell's equations

Option	Default	Description	Poisson	Maxwell
`CalcCoupledPower`	F	absorbed power by charged particles due to electro(-magnetic) fields	yes	yes
`CalcPlasmaFreqeuncy`	F	(cold) plasma frequency depending on the electron particle number density	yes	yes
`CalcPICTimeStep`	F	PIC time step resolution depending on the (cold) plasma frequency	yes	yes
`CalcDebyeLength`	F	Debye length depending on the electron temperature and particle number density	yes	yes
`CalcPointsPerDebyeL`	F	PIC spatial resolution depending on the element size and the Debye length	yes	yes
`CalcPICCFLCondition`	F	single parameter combining plasma frequency and Debye length criterion	yes	yes
`CalcMaxPartDisplace`	F	largest displacement of a particle within one time step related to the cell size	yes	yes
`CalcPICTimeStepCycl`	F	PIC time step resolution depending on the cyclotron frequency (magnetic field)	yes	yes
`CalcCyclotronFreque`	F	cyclotron frequency of charged particles in magnetic fields	yes	yes

Power Coupled to Particles The energy transferred to particles during the push (acceleration due to electromagnetic fields) is determined by using

```
CalcCoupledPower = T
```

which calculates the time-averaged power (moving average) coupled to the particles in each cell (average power per cubic metre) and stores it in `PCouplDensityAvgElem` for each species separately. Additionally, the properties `PCoupl`

(instantaneous) and a time-averaged (moving average) value `PCoupledMoAv` are stored in the `ParticleAnalyze.csv` output file.

Plasma Frequency The (cold) plasma frequency can be calculated via

$$\omega_p = \omega_e = \sqrt{\frac{e^2 n_e}{\varepsilon_0 m_e}}$$

which is the frequency with which the charge density of the electrons oscillates, where ε_0 is the permittivity of vacuum, e is the elementary charge, n_e and m_e are the electron density and mass, respectively. The calculation is activated by

```
CalcPlasmaFreqeuncy = T
```

and the output container for the data is labelled `PlasmaFrequencyCell`. Note that this output is only performed for elements that are considered valid, which is also stored as a separate container `PICValidPlasmaCell` (1: True, 0: False). The criterion that an element is valid is that a quasi-neutral plasma should be present in the element for plasma and parameters to be meaningful. Therefore, the quasi-neutrality parameter must be above 0.5 and at least 20 particles present in the considered element, independent of their charge value.

PIC Particle Time Step The maximum allowed time step within the PIC schemes can be estimated by

$$\Delta_{t,\mathrm{PIC}} < \frac{0.2}{\omega_p}$$

where ω_p is the (cold) plasma frequency (only calculated for valid elements, see above). The calculation is activated by

```
CalcPICTimeStep = T
```

and the output container for the data is labelled `PICTimeStepCell`. An additional output to `PartAnalyze.csv` is the percentage of elements that do not meet the PIC time step criterion via the column `-PercentResolvedPICTimeStep`.

Debye length The Debye length can be calculated via

$$\lambda_D = \sqrt{\frac{\varepsilon_0 k_B T_e}{e^2 n_e}}$$

where ε_0 is the permittivity of vacuum, k_B is the Boltzmann constant, e is the elementary charge and T_e and n_e are the electron temperature and density, respectively. The Debye length measures the distance after which the magnitude of the electrostatic potential of a single charge drops by $1/\mathrm{e}$. The calculation is activated by

```
CalcDebyeLength = T
```

and the output container for the data is labelled `DebyeLengthCell`. Again, this output is only created for valid elements, see above for details.

Points per Debye Length The spatial resolution in terms of grid points per Debye length can be estimated via

$$\mathrm{PPD} = \frac{\lambda_D}{\Delta x} = \frac{(p+1)\lambda_D}{L} \sim 1$$

where Δx is the grid spacing (average spacing between grid points), p is the polynomial degree of the solution, λ_D is the Debye length and $L = V^{1/3}$ is the characteristic cell length, which is determined from the volume V of the grid cell. Furthermore, the calculation in each direction x, y and z is performed by setting $L = \{L_x, L_y, L_z\}$, which are the average distances of the bounding box of each cell. These values are especially useful when dealing with Cartesian grids. The calculation is activated by

```
CalcPointsPerDebyeLength = T
```

and the output container for the 3D data is labelled `PPDCell3D`. Furthermore, a container for each spatial coordinate is created, which are labelled `PPDCellDirX`, `PPDCellDirY` and `PPDCellDirZ`, respectively. As for the PIC time step, an additional output to `PartAnalyze.csv` is the percentage of elements that do not meet the PPD criterion in 3D and for each spatial direction. There are four columns in the `PartAnalyze.csv` file for the Debye criterion, `-PercentResolvedPPD3`, `-PercentResolvedPPDX`, `-PercentResolvedPPDY` and `-PercentResolvedPPDZ`, respectively.

PIC CFL Condition The plasma frequency time step restriction and the spatial Debye length restriction can be merged into a single parameter

$$\frac{\Delta t}{0.4\Delta x}\sqrt{\frac{k_b T_e}{m_e}} = \frac{(p+1)\Delta t}{0.4L}\sqrt{\frac{k_b T_e}{m_e}} \lesssim 1$$

where Δt is the time step, Δx is the grid spacing (average spacing between grid points), p is the polynomial degree of the solution, k_B is the Boltzmann constant, T_e and m_e are the electron temperature and mass, respectively. Furthermore, the calculation in each direction x, y and z is performed by setting $L = \{L_x, L_y, L_z\}$, which are the average distances of the bounding box of each cell. These values are especially useful when dealing with Cartesian grids. The calculation is activated by

```
CalcPICCFLCondition = T
```

and the output container for the 3D data is labelled `PICCFL3D`. Furthermore, a container for each spatial coordinate is created, which are labelled `PICCFLDirX`, `PICCFLDirY` and `PICCFLDirZ`, respectively.

Maximum Particle Displacement The largest displacement of a particle within one time step Δt is estimated for each cell via

$$\frac{\max(v_{\mathrm{iPart}})\Delta t}{\Delta x} = \frac{(p+1)\max(v_{\mathrm{iPart}})\Delta t}{L} < 1$$

which means that the fastest particle is not allowed to travel over the length of two grid points separated by Δx. Furthermore, the calculation in each direction x, y and z is performed by setting $L = \{L_x, L_y, L_z\}$, which are the average distances of the bounding box of each cell. These values are especially useful when dealing with Cartesian grids. The calculation is activated by

```
CalcMaxPartDisplacement = T
```

and the output container for the 3D data is labelled `MaxPartDisplacement3D`. Furthermore, a container for each spatial coordinate is created, which are labelled `MaxPartDisplacementDirX`, `MaxPartDisplacementDirY` and `MaxPartDisplacementDirZ`, respectively.

Electron Cyclotron Motion The gyrokinetic or cyclotron motion of electrons can be analyzed by calculating the cyclotron frequency for the non-relativistic case

$$\omega_c = \frac{eB}{m_e}$$

Symbol	Parameter
ω_c	cyclotron frequency (non-relativistic)
e	elementary charge (of an electron, absolute value)
B	magnitude of the magnetic flux density at the electron's position
m_e	electron rest mass

or if the electron velocity is considered to be relativistic (automatic switch), the following formula is used

$$\omega_c = \frac{eB}{\gamma m_e} = \frac{eB}{m_e\sqrt{1 - v_e^2/c^2}}$$

Symbol	Parameter
ω_c	cyclotron frequency (relativistic)
e	elementary charge (of an electron, absolute value)
B	magnitude of the magnetic flux density at the electron's position
γ	Lorentz factor
m_e	electron rest mass
v_e	magnitude of velocity
c	speed of light

and, hence, the required time step (Boris push)

$$\Delta t = \frac{0.02}{\omega_c}$$

to resolve the gyro motion adequately by setting the following two parameters

```
CalcPICTimeStepCyclotron = T
CalcCyclotronFrequency   = T
```

The first activates the calculation of the smallest time step in each cell (only regarding the cyclotron motion, not other restrictions) and the second activates the calculation of the min/max value of the cyclotron frequency (and also radius) for each cell. The containers that are written to `.h5` (and converted to `.vtu`) are

```
PICTimeStepCyclotronCell
```

for the time step restriction and

```
CyclotronFrequencyMaxCell
CyclotronFrequencyMinCell
GyroradiusMinCell
GyroradiusMaxCell
```

for the cyclotron frequency and radius (min/max values).

Time-averaged Fields

At each `Analyze_dt` and at the end of the simulation, additional time-averaged field properties can be written to `*_TimeAvg_*.h5` by enabling

```
CalcTimeAverage = T
```

where the averaging will take place over the time spanned between two `Analyze_dt` outputs. The time-averaged values and fluctuations are selected via `VarNameAvg` and `VarNameFluc`, depending on the equation system that is solved (Poisson or Maxwell). These properties are stored in the same format as the solution polynomial of degree N, i.e., $(N+1)^3$ data points for each cell. For Maxwell's equations, the following properties are available

```
VarName{Avg,Fluc} = ElectricFieldX
VarName{Avg,Fluc} = ElectricFieldY
VarName{Avg,Fluc} = ElectricFieldZ
VarName{Avg,Fluc} = MagneticFieldX
VarName{Avg,Fluc} = MagneticFieldY
VarName{Avg,Fluc} = MagneticFieldZ
VarName{Avg,Fluc} = Phi
VarName{Avg,Fluc} = Psi
VarName{Avg,Fluc} = ElectricFieldMagnitude
VarName{Avg,Fluc} = MagneticFieldMagnitude
VarName{Avg,Fluc} = PoyntingVectorX
VarName{Avg,Fluc} = PoyntingVectorY
VarName{Avg,Fluc} = PoyntingVectorZ
VarName{Avg,Fluc} = PoyntingVectorMagnitude
```

and for Poisson's equation

```
VarName{Avg,Fluc} = Phi
VarName{Avg,Fluc} = ElectricFieldX
VarName{Avg,Fluc} = ElectricFieldY
VarName{Avg,Fluc} = ElectricFieldZ
VarName{Avg,Fluc} = ElectricFieldMagnitude
```

In case of a PIC simulation, the particle-grid deposited properties (via the user-selected deposition method) are also available via `VarNameAvg` and `VarNameFluc`

```
VarName{Avg,Fluc} = PowerDensityX-Spec0x
VarName{Avg,Fluc} = PowerDensityY-Spec0x
VarName{Avg,Fluc} = PowerDensityZ-Spec0x
VarName{Avg,Fluc} = PowerDensity-Spec0x
VarName{Avg,Fluc} = ChargeDensity-Spec0x
VarName{Avg,Fluc} = ChargeDensityX-Spec0x
VarName{Avg,Fluc} = ChargeDensityY-Spec0x
VarName{Avg,Fluc} = ChargeDensityZ-Spec0x
VarName{Avg,Fluc} = ChargeDensity-Spec0x
```

which must be supplied for each particle species `x` separately.

1.5.3 Particle Flow and Surface Sampling

Flow field and surface outputs are available when the DSMC, BGK and FP methods are utilized (standalone or coupled with PIC) and stored in `*_DSMCState_*.h5`. A sampling over a certain number of iterations is performed to calculate the average macroscopic values such as number density, bulk velocity and temperature from the microscopic particle information. Two variants are available in PICLas, allowing to sample a certain amount of the simulation duration or to sample continuously during the simulation and output the result after the given number of iterations.

The first variant is usually utilized to sample at the end of a simulation, when the steady state condition is reached. The first parameter `Part-TimeFracForSampling` defines the percentage that shall be sampled relative to the simulation end time T_{end} (Parameter: `TEnd`)

```
Part-TimeFracForSampling = 0.1
Particles-NumberForDSMCOutputs = 2
```

`Particles-NumberForDSMCOutputs` defines the number of outputs during the sampling time. Example: The simulation end time is $T_{\mathrm{end}} = 1$, thus sampling will begin at $T = 0.9$ and the first output will be written at $T = 0.95$. At this point the sample will NOT be resetted but continued. Therefore, the second and last output at $T = T_{\mathrm{end}} = 1.0$ is not independent of the previous result but contains the sample of the complete sampling duration. It should be noted that if a simulation is continued at e.g. $T = 0.95$, sampling with the given parameters will begin immediately.

The second variant can be used to produce outputs for unsteady simulations, while still to be able to sample for a number of iterations (Parameter: `Part-IterationForMacroVal`). The first two flags allow to enable the output of flow field/volume and surface values, respectively.

```
Part-WriteMacroVolumeValues = T
Part-WriteMacroSurfaceValues = T
Part-IterationForMacroVal = 100
```

Example: The simulation end time is $T_{\mathrm{end}} = 1$ with a time step of $\Delta t = 0.001$. With the parameters given above, we would sample for 100 iterations up to $T = 0.1$ and get the first output. Afterwards, the sample is deleted and the sampling begins anew for the following output at $T = 0.2$. This procedure is repeated until the simulation end, resulting in 10 outputs with independent samples.

Parameters indicating the quality of the simulation (e.g. the maximal collision probability in case of DSMC) can be enabled by

```
Particles-DSMC-CalcQualityFactors = T
```

Output and sampling on surfaces can be enabled by

```
Particles-DSMC-CalcSurfaceVal = T
```

By default this will include the species-specific impact counter per iteration of simulation particles, the force per area in x, y, and z and the heat flux. The output of the surface-sampled data is written to `*_DSMCSurfState_*.h5`. Additional surface values can be sampled by using

```
CalcSurfaceImpact = T
```

which calculates the species-dependent averaged impact energy (trans, rot, vib, elec), impact vector, impact obliqueness angle (between particle trajectory and surface normal vector, e.g. an impact vector perpendicular to the surface corresponds to an impact angle of 0°), number of real particle impacts over the sampling duration and number of real particle impacts per area per second.

Electronic excitation

To sample the cell-local excitation of electronic energy levels, an additional flag has to be set

```
Part-SampleElectronicExcitation = T
```

This option adds an additional container to the DSMCState output, which after a successful conversion with `piclas2vtk` will produce an additional file `*_visuExcitationData_*.vtu`. Here the excitation rate is given per second [1/s] for each electronic level. This option is currently only supported with the cross-section based electronic excitation (see Section *Cross-section based electronic relaxation probability*).

1.5.4 Integral Variables

This analysis measures integral values from the field- and/or particle-solver data over time and writes different .csv files. Mainly field-related data is stored in `FieldAnalyze.csv`, whereas particle-related analysis is divided into mostly global data in `PartAnalyze.csv` and surface data in `SurfaceAnalyze.csv`. Some information might be scattered across different files if multiple things such as fields and particles/surfaces are involved.

Field Variables

The output of properties regarding Maxwell's or Poisson's field solver are written to `FieldAnalyze.csv`. If this analysis is not required in each time step, the parameter `Field-AnalyzeStep` (default is 1) can be set to an integer greater or equal 1.

Option	Default	Description	Poisson	Maxwell
`CalcElectricTimeDerivative`	F	time derivative $\frac{\partial D}{\partial t} = \varepsilon_r \varepsilon_0 \frac{\partial E}{\partial t}$	yes	no
`CalcPotentialEnergy`	F	potential field energy of the EM field	yes	yes
`CalcBoundaryFieldOutput`	F	electric potential at user-specified boundaries	yes	no

Displacement current: The temporal change of the electric displacement field $\frac{\partial D}{\partial t} = \varepsilon_r \varepsilon_0 \frac{\partial E}{\partial t}$ can be stored for Poisson's equation (`PICLAS_EQNSYSNAME=poisson`) by setting

```
CalcElectricTimeDerivative = T
```

The integrated displacement current that traverses each field boundary (except periodic BCs) can be analyzed over time and is written to `FieldAnalyze.csv` for each boundary separately, e.g. *"007-ElecDisplCurrent-001-BC_left"*. The electric displacement current is also considered when the total electric current on specific surfaces is calculated, for details see BoundaryParticleOutput (BPO) in Section *Surface Variables*.

Potential field energy: The global energy that is stored within the electric and the magnetic field can be analyzed over time by setting

```
CalcPotentialEnergy = T
```

and is written to `FieldAnalyze.csv`, e.g. *"002-E-El" and "003-E-Mag"* (magnetic energy is only calculated for Maxwell's equations'). Additionally, but only when solving Maxwell's equations, the energy stored in the divergence correction fields *"004-E-phi"* and *"005-E-psi"* as well as the total potential energy *"006-E-pot"*, respectively, can be analyzed.

Boundary Field Output The electric potential for specific boundaries can be analyzed over time by setting

```
CalcBoundaryFieldOutput = T
BFO-NFieldBoundaries    = 3         ! Nbr of boundaries
BFO-FieldBoundaries     = (/1,3,5/) ! Field-Boundary1, 3 and 5
```

where `BFO-NFieldBoundaries` defines the number of boundaries that are of interest and `BFO-FieldBoundaries` the boundary IDs of the field boundaries where the electric potential is to be recorded. Note that this output is only meaningful when a scalar potential is used on the corresponding boundary. Therefore, this output is only available for certain BCTypes, e.g., 2, 4, 5 and 6. The output is written to `FieldAnalyze.csv` for each boundary separately, e.g. *"006-BFO-boundary001"*.

Particle Variables

PIC resolution parameters: Information regrading the global percentage of elements that fulfill the PIC resolution criteria in time (max. PIC time step) and space (Debye length) can be written to `PartAnalyze.csv`. For further details and how this analysis are activated, see Section *Element-constant field/particle properties*.

Surface Variables

The output for different measurements involving particles and surfaces is written to `SurfaceAnalyze.csv`. If this analysis is not required in each time step, the parameter `Surface-AnalyzeStep` (default is 1) can be set to an integer greater or equal 1. Some values are sampled only during one time step and others are accumulated over the number of time steps defined by `Surface-AnalyzeStep`.

Parameter	Parameter
`CalcElectronSEE`	Secondary electron emission current for each `Part-Boundary` with SEE model
`CalcBoundaryParticleOutput`	Particle flux for each user-defined `Part-Boundary` and `Part-Species`

Secondary Electron Emission When `CalcElectronSEE=T` is activated, the secondary electron emission current (on all surfaces where such a model is used) is calculated and written to `SurfaceAnalyze.csv`. Note that all secondary electrons between two outputs are accumulated and divided by the time difference between these outputs. Hence, the averaging time can be adjusted using `Surface-AnalyzeStep`. The output in the .csv file will be similar to this example: `012-ElectricCurrentSEE-BC_WALL`

BoundaryParticleOutput (BPO): The flux of particles crossing boundaries where they are removed can be calculated by setting `CalcBoundaryParticleOutput = T`. Additionally, the boundaries and species IDs must be supplied for which the output is to be created. This is done by setting

```
CalcBoundaryParticleOutput = T        ! Activate the analysis
BPO-NPartBoundaries        = 2        ! Nbr of particle boundaries where the flux and␣
↪current are measured
BPO-PartBoundaries         = (/4,8/)  ! Only measure the flux and current on Part-
↪Boundary4 and Part-Boundary8
BPO-NSpecies               = 2        ! Nbr of species that are considered for Part-
↪Boundary4 and Part-Boundary8
BPO-Species                = (/2,3/)  ! Species IDs which should be included
```

where the number of boundaries and species as well as the corresponding IDs are defined. The other boundaries and species IDs will be ignored. Note that this feature is currently only implemented for boundaries of type `Part-BoundaryX-Condition = open` and `Part-BoundaryX-Condition = reflective`. The reflective BC must also use either species swap via `Part-BoundaryX-NbrOfSpeciesSwaps` or a secondary electron emission surface model via `Part-BoundaryX-SurfaceModel`. The output in the .csv file will be similar to this example: `004-Flux-Spec-002-BC_CATHODE` Additionally, the total electric current will be calculated if any species that is selected via `BPO-Species` carries a charge unequal to zero. Note that only species defined via `BPO-Species` will be considered in the calculation of the total electric current. Furthermore, the secondary electron emission current is automatically added to the total electric current if `CalcBoundaryParticleOutput = T`. If the electric displacement current is calculated, it also will be added to the total current, if `CalcElectricTimeDerivative = T`. The output of the total electric current in the .csv file will be similar to this example: `008-TotalElectricCurrent-BC_ANODE`

1.5.5 Dynamic Mode Decomposition

The dynamic mode decomposition is an algorithm that divides a temporal series into a set of modes which are associated with a frequency and grow/decay rate. The dynamic mode decomposition (DMD) is implemented according to Schmid et al. [69]. To use the DMD tool, it needs to be compiled first. In cmake, set the flag

```
POSTI_BUILD_DMD = ON (default is OFF)
```

which creates the executable `./bin/dmd`. The input for the DMD tool is provided by a series of state files with a high temporal resolution between each output file in order to capture the relevant frequencies that are to be visualized. To analyze a specific frequency, multiple state files should be created within one period of the oscillation.

Run the DMD executable with the help command to see the available options

```
./dmd --help
```

To execute the DMD after a simulation, with e.g. the Maxwell solver, run the command

```
dmd dmd.ini coaxial_State_000.00*
```

with an exemplary `dmd.ini` file

```
N            = 4
SvdThreshold = 1e-8 ! Define relative lower bound of singular values
```

where `N=4` is the polynomial degree of the solution and `SvdThreshold = 1e-8` is used to filter singular values of the DMD.

Depending on the available memory one might have to decrease the number of input state files. After the execution two additional files **coaxial_DMD.h5** and **coaxial_DMD_Spec.dat** will be created. The first file contains the field representation of the different modes and the second file contains the Ritz spectrum of the modes.

Ritz spectrum (coaxial_DMD_Spec.dat)

With the python script *plot_RitzSpectrum.py* the Ritz spectrum of the DMD can be plotted. The script is placed in the tools folder of **piclas**. To plot the spectrum execute:

```
python [PICLAS_DIRECTORY]/tools/plot_RitzSpectrum.py -d coaxial_DMD_Spec.dat
```

The result is a Ritz spectrum depicting all the calculated modes and their growth rate. On the *x-axis* the frequency of the modes and on the *y-axis* the growth/decay factor is plotted, whereat modes with $\omega_r < 0$ are damped. The modes placed directly on the x-axis are the global, the first, the second harmonic mode and so on. The color and size of the plotted modes represent the Euclidian norm of the mode which can be interpreted as an energy norm of the mode.

Mode visualization (coaxial_DMD.h5)

To visualize the field run the following command:

```
piclas2vtk [posti.ini] coaxial_DMD.h5
```

The new file **coaxial_DMD_000.00000000000000000.vtu** now contains multiple modes to visualize. Two additional input parameters are used to control the output of `piclas2vtk` by placing them in the `posti.ini` file

```
dmdSingleModeOutput  = 2  ! only extract mode 002
dmdMaximumModeOutput = 10 ! only extract modes 001-010
```

The parameter `dmdSingleModeOutput` is used to convert only a single specific mode to `.vtu` ignoring all other modes and `dmdMaximumModeOutput` can be used to dump all output modes up to this number. Note that each mode is written to a separate output file because a single might can become quite large very quickly and is then too large to visualize.

1.6 Tools

This section gives an overview over the tools and scripts contained in the **PICLas** repository. It also provides references to the tutorials where their usage is explained. An overview of the tools is given in TOOLS.md.

1.6.1 Unified Species Database (USD)

The unified species database was created for a more convenient alternative input for the simulations of species data, electronic states, cross-sections, and chemistry models. For the general structure and using the unified species database, please refer to Chapter *Unified Species Database* for instructions.

Maintain and edit database

A tool to modify or maintain the database it is recommended to use the `maintain_database.py` which can be found in the *tools* folder: `piclas/tools/species_database/`. Its basic usage is to close the database in other programs and run

```
python3 maintain_database.py
```

The script shows 5 options to choose from:

```
1 to maintain/edit species  or
2 to maintain/edit chemical reactions  or
3 to maintain/edit cross section data  or
4 to maintain/edit surface chemistry  or
5 to exit program
```

Species parameters

After selecting to edit/maintain species the script offers the following options

```
1 check existing species  or
2 add new species  or
3 to exit program
```

The first option loops over all atom species in the database and compares the electronic excitation states (datasets) to the electronic excitation states from the NIST database. Afterwards, the attribute values including the mass and heat of formation are compared with data obtained from the Active Thermochemical Tables for all species. Currently the verification of electronic excitation states for molecules is not implemented yet due to the lack of data in other databases.

There is an option to add electronic excitation states for molecules by providing the data as comma-separated values (csv). For activating this feature the electronic excitation states need to be stored in a .csv file in `piclas/tools/species_database/`. The name and format should follow the template `custom_electronic_levels_SPECIES.csv`, where SPECIES is replaced with the actual name of the species, e.g. 'H2'. The format is analogous to the output of the NIST database:

Term	J	Levelcm-1
—	XXX	XX
—	XXX	XX
Limit	–	XX

where each row represents one electronic state and the last row contains the ionization energy. The first column can remain empty. The second column contains the total angular momentum J, which is converted to the degeneracy g with $g = 2J + 1$. The third column contains the energy value in 1/cm, which will be converted to Kelvin.

For each atom species, the electronic excitation states are kept if the data is within a relative tolerance of 1e-05 (which is set in `piclas/tools/species_database/edit_species.py`). If the data is not within this range or the number of electronic levels do not match the differences are displayed and it is possible to choose which data should be kept

```
1 to keep data and attributes from unified species database  or
2 to save only electronic level data from https://physics.nist.gov/cgi-bin/ASD/energy1.
↪pl  or
3 to skip all electronic levels and continue with only attributes  or
4 to exit program
```

If the third option is selected the verification of electronic excitation states is skipped for all atom species and the verification of the attributes is started.

For each species in the database the heat of formation (unit: K, `HeatOfFormation_K`), atomic mass (unit: kg, `MassIC`), charge (unit: C, `ChargeIC`) and the internal species identifier (`InteractionID`) are compared. The heat of formation and mass are obtained from the Active Thermochemical Tables while the charge and identifier are derived from the species name. If no differences are determined, the output should look like this

```
HeatOfFormation_K for SPECIES is equal so will be kept
MassIC for SPECIES is equal so will be kept
ChargeIC for SPECIES is equal so will be kept
InteractionID for SPECIES is equal so will be kept
```

Otherwise the differences are displayed and it is possible to select which data should be saved

```
1 to keep attributes from unified species database or
2 to save attributes from https://atct.anl.gov/Thermochemical%20Data/version%201.130 or
3 to exit program here
```

The script can be executed by providing the species as arguments

```
python3 maintain_database.py --species H Ar
```

to limit the check to these species. Note that the electronic excitation states for molecules will only be set from custom csv files.

The 'add new species' option follows the same logic. If no species is given with the –species argument, the species to add should be input comma seperated string, e.g. `Fe,Ar,H,CIon1,CIon2,C`.

For each species the electronic excitation states will be obtained from the NIST database, the attributes from Active Thermochemical Tables and VHS parameter (`Tref`, `dref`, `omega`) by user input.

Chemical reactions

After selecting to edit/maintain reactions the script offers the following options

```
1 add new reactions  or
2 delete reactions  or
3 to exit program
```

The first option will expect a list of reactions as a comma separated string, e.g. `C+N_CNIon1+electron,C2+M_C+M+C` and loop over all given reactions. If the reaction already exists in the database all entries will be listed, e.g.

```
Reaction: C+N_CNIon1+electron#1
* Created :  August 02, 2023
Activation-Energy_K :  164400.0
Arrhenius-Powerfactor :  1.5
Arrhenius-Prefactor :  1.66053927673551e-15
ChemistryModel :  [[b'Titan_18Spec_30Reac_Gokcen2007']]
Products: CNIon1,electron
Reactants: N,C
ReactionModel:  TCE
```

Currently there are the following options to choose from

```
1 add a new reaction with different attributes  or
2 add a new chemistry model to existing reaction  or
3 to skip this reaction  or
4 to exit program
```

When adding a new reaction some parameters such as the reaction model and chemistry model need to be set per user input. Other parameters such as the reactants and products are constructed from the reaction name or if the non reactive species need to be set as well.

The 'delete reactions' option will expect a list of reactions as comma separated string, e.g. `C+N_CNIon1+electron, C2+M_C+M+C`. If there is only one reaction in the database the reaction will be deleted and if there is more than one reaction stored in the database all reactions will be displayed like shown above. To choose which of the listed reactions should be deleted the number(s) of the reaction(s) to delete have to be entered as comma separated string, e.g. `1,2,3`.

Collision cross-sections

The option to create new collision cross-section data is not implemented in the `maintain_database.py` script. To add new collision cross-section data it is recommended to use the old workflow and revert to the regular parameter read-in for these species as described in *Species data* or insert the cross-section data by hand to the unified species database.

A tool to create a database containing cross-section data can be found in the *tools* folder: `piclas/tools/crosssection_database/`. The Python script (python3.7) `create_xsec_db_lxcat.py` can be used to populate a PICLas-compatible cross-section database, using the `numpy`, `h5py` and `lxcat_data_parser` packages.

```
python3.7 create_xsec_db_lxcat.py
```

A database (containing multiple species and cross-section types) downloaded directly from the Plasma Data Exchange Project and the LXCat database and the name of output database can be supplied to the script with

```
database_input = "Database.txt"
database_output = "Database.h5"
```

Currently, PICLas only utilizes the elastic, effective and vibrational cross-sections, however, all excitation cross-section types are grouped and stored in the output file. An example is given below

```
CO2-electron (group)
    EFFECTIVE (dataset)
    ROTATION (group)
        0.02 (dataset)
    VIBRATION (group)
        0.29
        0.59
    REACTION (group)
        CO2Ion1-electron-electron
```

Datasets, which cannot be identified as rotational, vibrational or electronic excitation will grouped within an `UNDEFINED` group. By defining a species list, only certain species can be included in the output database

```
species_list = ["Ar","CO"]
```

Finally, the utilized cross-section data should be properly referenced by adding the information to the HDF5 database as an attribute

```
reference = 'XXX database, www.lxcat.net, retrieved on MMMM DD, YYYY.'
```

Users of cross-section data are encouraged to download the data directly from the LXCat project website and to consider the guidelines regarding referencing and publication.

Surface chemistry

The option to create new surface chemistry data is not implemented in the `maintain_database.py` script. To add surface chemistry data it is recommended to add data by hand to the unified species database.

1.6.2 Userblock

The `userblock` contains the complete information about a **PICLas** run (git branch of the repository, differences to that branch, `cmake` configuration and parameter file) and is prepended to every `.h5` state file. The parameter file is prepended in ASCII format, the rest is binary and is generated automatically during the build process with the `generateuserblock.sh` script.

`extract_userblock.py`

It can be extracted and printed using the `extract_userblock.py` script. Its basic usage is

```
python2 extract_userblock.py -XXX [statefile.h5]
```

where `-XXX` can be replaced by

- `-s` to show all available parts of the userblock (such as `CMAKE` or `GIT BRANCH`)
- `-a` to print the complete userblock
- `-p [part]` to print one of the parts listed with the `-s` command.

rebuild.py

The second python tool in this folder is `rebuild.py`. It extracts the userblock from a state file and builds a **PICLas** repository and binary identical to the one that the state file was created with. In order to do so, it clones a **PICLas** git repository, checks out the given branch, applies the stored changes to the git `HEAD` and builds **PICLas** with the stored `cmake` options. If run with the parameter file given in the `INIFILE` part of the userblock, this binary should reproduce the same results/behaviour (possible remaining sources of different output are for example differences in restart files, compilers, linked libraries or machines). The basic usage is

```
python2 rebuild.py [dir] [statefile.h5]
```

where `dir` is an empty directory that the repository is cloned into and where the `piclas` executable is built. `statefile.h5` is the state file whose userblock is used to rebuild the `piclas` executable. Help can be shown via `-h` for both userblock scripts.

1.7 Tutorials

This chapter will give a detailed overview of simulations with **PICLas**. It assumes that you are familiar with setting the compiler options and compiling the code. The paths to the executables are omitted. It is assumed that you have either added aliases for **piclas**, **hopr** and **piclas2vtk**, or that you added the binary directories to your `$PATH` variable as described in *Directory paths*.

Each tutorial is equipped with .ini files, *hopr.ini*, *parameter.ini*, *posti.ini* and for DSMC setups *DSMC.ini*, as well as the mesh file **_mesh.h5* in the HDF5 format (created with **hopr**).

```
hopr.ini
parameter.ini
DSMC.ini
posti.ini
mesh.h5
```

It is suggested to copy each folder to a new directory, where you can run and modify the parameter (*.ini) files.

1.7.1 Plasma Wave (PIC, Poisson's Equation)

The setup considers a 1D plasma oscillation, which is a common and simple electrostatic PIC benchmark [54], [70],[71]. In PICLas it can be simulated either with the full Maxwell solver (DGSEM) or with the Poisson solver (HDGSEM), where the latter is chosen for this this tutorial. In this setup, electrons oscillate around the almost immobile ions, which creates a fluctuating electric field.

Before beginning with the tutorial, copy the `pic-poisson-plasma-wave` directory from the tutorial folder in the top level directory to a separate location

```
cp -r $PICLAS_PATH/tutorials/pic-poisson-plasma-wave .
cd pic-poisson-plasma-wave
```

Mesh Generation with HOPR (pre-processing)

Before the actual simulation is conducted, a mesh file in the correct HDF5 format has to be supplied. The mesh files used by **piclas** are created by supplying an input file *hopr.ini* with the required information for a mesh that has either been created by an external mesh generator or directly from block-structured information in the hopr.ini file itself. Here, a block-structured grid is created directly from the information in the hopr.ini file. To create the .h5 mesh file, simply run

```
hopr hopr.ini
```

This creates the mesh file *plasma_wave_mesh.h5* in HDF5 format and is depicted in Fig. 1.1. Alternatively, if you do not want to run **hopr** yourself, you can also use the provided mesh.

The size of the simulation domain is set to $[2\pi \times 0.2 \times 0.2]$ m^3 and is defined by the single block information in the line, where each node of the hexahedral element is defined

```
Corner         =   (/0.,0.,0.,,6.2831,0.,0.,,6.2831, ... /)
```

The number of mesh elements for the block in each direction can be adjusted by changing the line

```
nElems         = (/60,1,1/)                ! number of elements in each direction (x,y,z)
```

Each side of the block has to be assigned a boundary index, which corresponds to the boundaries defined in the next steps

```
BCIndex        = (/5,3,2,4,1,6/)
```

The field boundaries can directly be defined in the hopr.ini file (contrary to the particle boundary conditions, which are defined in the parameter.ini). Periodic boundaries always have to be defined in the hopr.ini.

```
!================================================================================= !
! BOUNDARY CONDITIONS
!================================================================================= !
BoundaryName = BC_periodicx+ ! Periodic (+vv1)
BoundaryType = (/1,0,0,1/)   ! Periodic (+vv1)
BoundaryName = BC_periodicx- ! Periodic (-vv1)
BoundaryType = (/1,0,0,-1/)  ! Periodic (-vv1)
BoundaryName = BC_periodicy+ ! Periodic (+vv2)
BoundaryType = (/1,0,0,2/)   ! Periodic (+vv2)
BoundaryName = BC_periodicy- ! Periodic (-vv2)
BoundaryType = (/1,0,0,-2/)  ! Periodic (-vv2)
BoundaryName = BC_periodicz+ ! Periodic (+vv3)
BoundaryType = (/1,0,0,3/)   ! Periodic (+vv3)
BoundaryName = BC_periodicz- ! Periodic (-vv3)
BoundaryType = (/1,0,0,-3/)  ! Periodic (-vv3)

VV = (/6.2831 , 0.  , 0./)   ! Displacement vector 1 (x-direction)
VV = (/0.     , 0.2 , 0./)   ! Displacement vector 2 (y-direction)
VV = (/0.     , 0.  , 0.2/)  ! Displacement vector 3 (z-direction)
```

In this case a fully periodic setup is chosen by defining periodic boundaries on all six sides of the block, reflecting each positive and negative Cartesian coordinate. In x-direction,

```
BoundaryName = BC_periodicx+ ! Periodic (+vv1)
BoundaryType = (/1,0,0,1/)   ! Periodic (+vv1)
```

(continues on next page)

(continued from previous page)

```
BoundaryName = BC_periodicx- ! Periodic (-vv1)
BoundaryType = (/1,0,0,-1/)  ! Periodic (-vv1)
```

where for each of the six boundaries, a name `BoundaryName` and a type `BoundaryType` must be defined (in this order). The boundary name can be chosen by the user and will be used again in the parameter.ini. The first "1" in `BoundaryType` corresponds to the type "periodic" and the last entry, here, either "1" or "-1" corresponds to the first periodic vector that is defined via `VV=(/6.2831 , 0. , 0./)` that handles periodicity in the x-direction and gives the orientation on the boundary for the vector. Note that each periodic boundary must have one positive and one negative corresponding boundary for the same periodic vector.

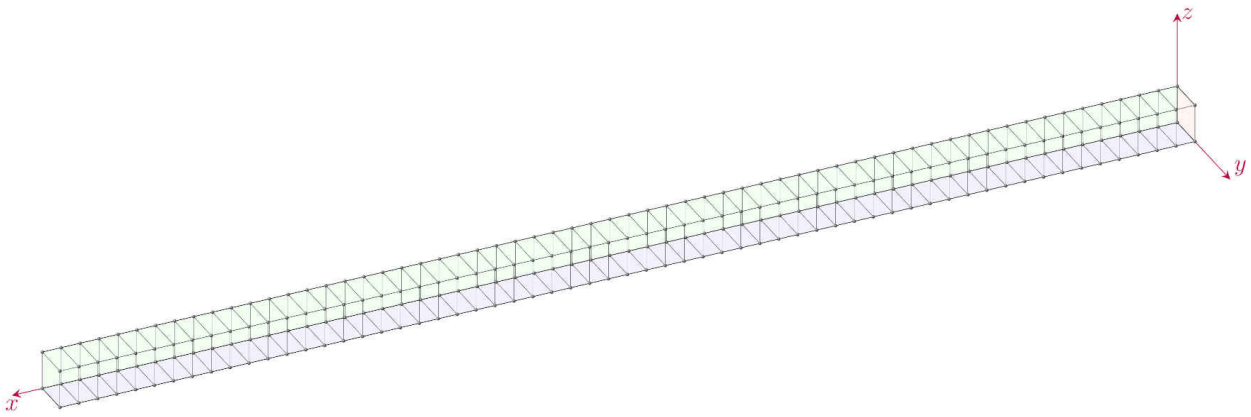

Fig. 1.1: Mesh with $60 \times 1 \times 1$ elements and a size of $[2\pi \times 0.2 \times 0.2]$ m^3.

PIC Simulation with PICLas

Install **piclas** by compiling the source code as described in Chapter *Installation* and make sure to set the correct compile flags

```
PICLAS_EQNSYSNAME     = poisson
PICLAS_TIMEDISCMETHOD = RK3
```

or simply run the following command from inside the *build* directory

```
cmake ../ -DPICLAS_EQNSYSNAME=poisson -DPICLAS_TIMEDISCMETHOD=RK3
```

to configure the build process and run `make` afterwards to build the executable. For this setup, we have chosen the Poisson solver and selected the three-stage, third-order low-storage Runge-Kutta time discretization method. An overview over the available solver and discretization options is given in Section *Solver settings*. To run the simulation and analyse the results, the *piclas* and *piclas2vtk* executables have to be run. To avoid having to use the entire file path, you can either set aliases for both, copy them to your local tutorial directory or create a link to the files via.

```
ln -s $PICLAS_PATH/build/bin/piclas
ln -s $PICLAS_PATH/build/bin/piclas2vtk
```

The simulation setup is defined in *parameter.ini*. For a specific electron number density, the plasma frequency of the system is given by

$$\omega_p = \omega_e = \sqrt{\frac{e^2 n_e}{\varepsilon_0 m_e}} \, ,$$

which is the frequency with which the charge density of the electrons oscillates, where ε_0 is the permittivity of vacuum, e is the elementary charge, n_e and m_e are the electron density and mass, respectively. For the standard PIC method, the plasma frequency yields the smallest time step that has to be resolved numerically. The Debye length

$$\lambda_D = \sqrt{\frac{\varepsilon_0 k_B T_e}{e^2 n_e}} \, ,$$

where ε_0 is the permittivity of vacuum, k_B is the Boltzmann constant, e is the elementary charge and T_e and n_e are the electron temperature and density, respectively, gives a spatial resolution constraint. In this test case, however, the electron temperature is not the defining factor for the spatial resolution because of the 1D nature of the setup. Therefore, the resolution that is required is dictated by the gradient of the electric potential solution, i.e., the electric field, which accelerates the charged particles and must be adequately resolved. The restriction on the spatial resolution is simply the number of elements (and polynomial degree N) that are required to resolve the physical properties of the PIC simulation. If the temporal and spatial constraints are violated, the simulation will not yield physical results and might even result in a termination of the simulation.

The physical parameters for this test case are summarized in Table 1.6.

Table 1.6: Physical properties

Property	Value
electron number density n_e	$8e11m^{-3}$
electron mass m_e	$9.1093826E-31kg$
ion number density n_i	$8e11m^{-3}$
ion mass m_i	$1.672621637E-27kg$
electron/ion charge $q_{i,e}$	$\pm 1.60217653E-19C$

General numerical setup

The general numerical parameters are selected by the following

```
! ================================================================================= !
! DISCRETIZATION
! ================================================================================= !
N             = 5  ! Polynomial degree of the DG method (field solver)

! ================================================================================= !
! MESH
! ================================================================================= !
MeshFile      = plasma_wave_mesh.h5 ! Relative path to the mesh .h5 file

! ================================================================================= !
! General
! ================================================================================= !
ProjectName       = plasma_wave ! Project name that is used for naming state files
ColoredOutput     = F           ! Turn ANSI terminal colors ON/OFF
doPrintStatusLine = T           ! Output live of ETA
TrackingMethod   = refmapping
```

where, among others, the polynomial degree N, the path to the mesh file `MeshFile`, project name and the option to print the ETA to the terminal output in each time step.

The temporal parameters of the simulation are controlled via

```
! ================================================================================= !
! CALCULATION
! ================================================================================= !
ManualTimeStep  = 5e-10 ! Fixed pre-defined time step only when using the Poisson solver.
↪ Maxwell solver calculates dt that considers the CFL criterion
tend            = 40e-9 ! Final simulation time
Analyze_dt      = 4e-9  ! Simulation time between analysis
IterDisplayStep = 50    ! Number of iterations between terminal output showing the␣
↪current time step iteration
```

where the time step for the field and particle solver is set via `ManualTimeStep`, the final simulation time `tend`, the time between restart/checkpoint file output `Analyze_dt` (also the output time for specific analysis functions) and the number of time step iterations `IterDisplayStep` between information output regarding the current status of the simulation that is written to std.out. The remaining parameters are selected for the field and particle solver as well as run-time analysis.

Boundary conditions

As there are no walls present in the setup, all boundaries are set as periodic boundary conditions for the field as well as the particle solver. The particle boundary conditions are set by the following lines

```
! ================================================================================= !
! PARTICLE Boundary Conditions
! ================================================================================= !
Part-nBounds              = 6             ! Number of particle boundaries
Part-Boundary1-SourceName = BC_periodicx+ ! Name of 1st particle BC
Part-Boundary1-Condition  = periodic      ! Type of 1st particle BC
Part-Boundary2-SourceName = BC_periodicx- ! ...
Part-Boundary2-Condition  = periodic      ! ...
Part-Boundary3-SourceName = BC_periodicy+ ! ...
Part-Boundary3-Condition  = periodic      ! ...
Part-Boundary4-SourceName = BC_periodicy- ! ...
Part-Boundary4-Condition  = periodic      ! ...
Part-Boundary5-SourceName = BC_periodicz+ ! ...
Part-Boundary5-Condition  = periodic      ! ...
Part-Boundary6-SourceName = BC_periodicz- ! ...
Part-Boundary6-Condition  = periodic      ! ...

Part-nPeriodicVectors = 3 ! Number of periodic boundary (particle and field) vectors

Part-FIBGMdeltas = (/6.2831 , 0.2 , 0.2/) ! Cartesian background mesh (bounding box␣
↪around the complete simulation domain)
Part-FactorFIBGM = (/60     , 1   , 1/)   ! Division factor that is applied t the "Part-
↪FIBGMdeltas" values to define the dx, dy and dz distances of the Cartesian background␣
↪mesh
```

where, the number of boundaries `Part-nBounds` (6 in 3D cuboid) is followed by the names of the boundaries (given by the hopr.ini file) and the type `periodic`. Furthermore, the periodic vectors must be supplied and the size of the Cartesian background mesh `Part-FIBGMdeltas`, which can be accompanied by a division factor (i.e. number of background cells) in each direction given by `Part-FactorFIBGM`. Here, the size and number of cells of the background mesh correspond to the actual mesh.

Field solver

The settings for the field solver (HDGSEM) are given by

```
! ================================================================================= !
! Field Solver: HDGSEM
! ================================================================================= !
epsCG                = 1e-6 ! Stopping criterion (residual) of iterative CG solver␣
↪(default that is used for the HDGSEM solver)
maxIterCG            = 1000 ! Maximum number of iterations
IniExactFunc         = 0    ! Initial field condition. 0: zero solution vector
```

where `epsCG` sets the abort residual of the CG solver, `maxIterCG` sets the maximum number of iterations within the CG solver and `IniExactFunc` set the initial solution of the field solver (here 0 says that nothing is selected).

The numerical scheme for tracking the movement of all particles throughout the simulation domain can be switched by

```
! ================================================================================= !
! Particle Solver
! ================================================================================= !
TrackingMethod    = refmapping  ! Particle tracking method
```

The PIC parameters for interpolation (of electric fields to the particle positions) and deposition (mapping of charge properties from particle locations to the grid) are selected via

```
! ================================================================================= !
! PIC: Interpolation/Deposition
! ================================================================================= !
PIC-DoInterpolation       = T                ! Activate Lorentz forces acting on␣
↪charged particles
PIC-Interpolation-Type    = particle_position ! Field interpolation method for Lorentz␣
↪force calculation

PIC-Deposition-Type           = shape_function_adaptive ! Particle-field coupling␣
↪method. shape_function_adaptive determines the cut-off radius of the shape function␣
↪automatically
PIC-shapefunction-dimension   = 1                       ! Shape function specific␣
↪dimensional setting
PIC-shapefunction-direction   = 1                       ! Shape function specific␣
↪coordinate direction setting
PIC-shapefunction-alpha       = 4                       ! Shape function specific␣
↪parameter that scales the waist diameter of the shape function
PIC-shapefunction-adaptive-DOF = 10                     ! Scaling factor for the␣
↪adaptive shape function radius (average number of DOF that are within the shape␣
↪function sphere in case of a Cartesian mesh)
```

where the interpolation type `PIC-Interpolation-Type = particle_position` is currently the only option for specifying how electro(-magnetic) fields are interpolated to the position of the charged particles. For charge and current deposition, a polynomial shape function with the exponent `PIC-shapefunction-alpha` of the type `PIC-Deposition-Type = shape_function_adaptive` is selected. The size of the shape function radius relative to the element size can be scaled via `PIC-shapefunction-adaptive-DOF`. The higher this value is, the more field DOF are within the shape function sphere. This increases the accuracy of the deposition method at the cost of computational resources. The dimension `PIC-shapefunction-dimension`, here 1D and direction `PIC-shapefunction-direction`, are selected specifically for the one-dimensional setup that is simulated here. The different available deposition types are described in more detail in Section *Charge and Current Deposition*.

Particle solver

For the treatment of particles, the maximum number of particles `Part-maxParticleNumber` that each processor can hold has to be supplied and the number of particle species `Part-nSpecies` that are used in the simulation (created initially or during the simulation time through chemical reactions).

```
! ============================================================================== !
! PARTICLE Emission
! ============================================================================== !
Part-maxParticleNumber    = 4000 ! Maximum number of particles (per processor/thread)
Part-nSpecies             = 2    ! Number of particle species
```

The inserting (sometimes labelled emission or initialization) of particles at the beginning or during the course of the simulation is controlled via the following parameters. Here, only the parameters for the electrons are shown, however, the parameters for the ions are set analogously and included in the supplied parameter.ini. For each species, the mass (`Part-SpeciesX-MassIC`), charge (`Part-SpeciesX-ChargeIC`) and weighting factor (`Part-SpeciesX-MacroParticleFactor`) have to be defined.

```
! -------------------------------------
! Electrons 1
! -------------------------------------
Part-Species1-ChargeIC            = -1.60217653E-19 ! Electric charge of species #1
Part-Species1-MassIC              = 9.1093826E-31   ! Rest mass of species #1
Part-Species1-MacroParticleFactor = 5e8             ! Weighting factor for species #1
Part-Species1-nInits              = 1               ! Number of initialization/emission␣
↪regions for species #1
```

The number of initialization sets is defined by `Part-Species1-nInits`, where each initialization set is accompanied by a block of parameters that starts from `Part-Species1-Init1-SpaceIC` up to `Part-Species1-Init1-VeloVecIC` and are preceded by the corresponding `-InitX` counter. In this example we have a single initialization set per species definition. The `Part-Species1-Init1-SpaceIC = sin_deviation` flag defines the type of the initialization set, here, the distribution the particles equidistantly on a line and sinusoidally dislocates them (representing an initial stage of a plasma wave in 1D). Each type of the initialization set might have a different set of parameters and an overview is given in Section *Particle Initialization & Emission*.

```
Part-Species1-Init1-ParticleNumber        = 400           ! Number of simulation␣
↪particles for species #1 and initialization #1
Part-Species1-Init1-maxParticleNumber-x   = 400           ! Number of simulation␣
↪particles in x-direction for species #1 and initialization #1
Part-Species1-Init1-SpaceIC               = sin_deviation ! Sinusoidal distribution is␣
↪space
Part-Species1-Init1-velocityDistribution  = constant      ! Constant velocity␣
↪distribution
Part-Species1-Init1-maxParticleNumber-y   = 1             ! Number of particles in y
Part-Species1-Init1-maxParticleNumber-z   = 1             ! Number of particles in z
Part-Species1-Init1-Amplitude             = 0.01          ! Specific factor for the␣
↪sinusoidal distribution is space
Part-Species1-Init1-WaveNumber            = 2.            ! Specific factor for the␣
↪sinusoidal distribution is space
Part-Species1-Init1-VeloIC                = 0.            ! Velocity magnitude [m/s]
Part-Species1-Init1-VeloVecIC             = (/1.,0.,0./)  ! Normalized velocity vector
```

To calculate the number of simulation particles of, e.g. electrons, defined by `Part-Species1-Init1-ParticleNumber`, the given number density n_e in Table 1.6, the selected weighting

factor w_e and the volume of the complete domain ($V = 2\pi \cdot 0.2 \cdot 0.2m^3$) are utilized.

$$N_{e,sim} = \frac{n_e V}{w_e}$$

In this case, however, the number of particles are pre-defined and the weighting factor is derived from the above equation. The extent of dislocation is controlled by `Part-Species1-Init1-Amplitude`, which is only set for the electron species as the ion species is not dislocated (they remain equidistantly distributed). The parameter `Part-Species1-Init1-WaveNumber` sets the number of sine wave repetitions in the `x`-direction of the domain. In case of the `SpaceIC=sin_deviation`, the number of simulation particles must be equal to the multiplied values given in `Part-Species1-Init1-maxParticleNumber-x/y/z` as this emission type allows distributing the particles not only in one, but in all three Cartesian coordinates, which is not required for this 1D example.

Analysis setup

Finally, some parameters for run-time analysis are chosen by setting them `T` (true). Further, with `TimeStampLength = 13`, the names of the output files are shortened for better postprocessing. If this is not done, e.g. Paraview does not sort the files correctly and will display faulty behaviour over time.

```
! ================================================================================= !
! Analysis
! ================================================================================= !
TimeStampLength         = 13 ! Reduces the length of the timestamps in filenames for␣
↪better postprocessing
CalcCharge               = T ! writes rel/abs charge error to PartAnalyze.csv
CalcPotentialEnergy      = T ! writes the potential field energy to FieldAnalyze.csv
CalcKineticEnergy        = T ! writes the kinetic energy of all particle species to␣
↪PartAnalyze.csv
PIC-OutputSource         = T ! writes the deposited charge (RHS of Poisson's equation to␣
↪XXX_State_000.0000XXX.h5)
CalcPICTimeStep          = T ! writes the PIC time step restriction to XXX_State_000.
↪0000XXX.h5 (rule of thumb)
CalcPointsPerDebyeLength = T ! writes the PIC grid step restriction to XXX_State_000.
↪0000XXX.h5 (rule of thumb)
CalcTotalEnergy          = T ! writes the total energy of the system to PartAnalyze.csv␣
↪(field and particle)
```

The function of each parameter is given in the code comments. Information regarding every parameter can be obtained from running the command

```
piclas --help "CalcCharge"
```

where each parameter is simply supplied to the *help* module of **piclas**. This help module can also output the complete set of parameters via `piclas --help` or a subset of them by supplying a section, e.g., `piclas --help "HDG"` for the HDGSEM solver.

Running the code

The command

```
./piclas parameter.ini | tee std.out
```

executes the code and dumps all output into the file *std.out*. To reduce the computation time, the simulation can be run using the Message Passing Interface (MPI) on multiple cores, in this case 4

```
mpirun -np 4 piclas parameter.ini | tee std.out
```

If the run has completed successfully, which should take only a brief moment, the contents of the working folder should look like

```
4.0K drwxrwxr-x  4.0K Jun 28 13:07 ./
4.0K drwxrwxr-x  4.0K Jun 25 23:56 ../
8.0K -rw-rw-r--  5.8K Jun 28 12:51 ElemTimeStatistics.csv
120K -rw-rw-r--  113K Jun 28 12:51 FieldAnalyze.csv
4.0K -rw-rw-r--  2.1K Jun 26 16:49 hopr.ini
8.0K -rw-rw-r--  5.0K Jun 28 13:07 parameter.ini
156K -rw-rw-r--  151K Jun 28 12:51 PartAnalyze.csv
 32K -rw-rw-r--   32K Jun 26 16:43 plasma_wave_mesh.h5
1.6M -rw-rw-r--  1.6M Jun 28 12:44 plasma_wave_State_000.000000000.h5
1.6M -rw-rw-r--  1.6M Jun 28 12:45 plasma_wave_State_000.000000004.h5
1.6M -rw-rw-r--  1.6M Jun 28 12:45 plasma_wave_State_000.000000008.h5
1.6M -rw-rw-r--  1.6M Jun 28 12:46 plasma_wave_State_000.000000012.h5
1.6M -rw-rw-r--  1.6M Jun 28 12:47 plasma_wave_State_000.000000016.h5
1.6M -rw-rw-r--  1.6M Jun 28 12:48 plasma_wave_State_000.000000020.h5
1.6M -rw-rw-r--  1.6M Jun 28 12:49 plasma_wave_State_000.000000024.h5
1.6M -rw-rw-r--  1.6M Jun 28 12:50 plasma_wave_State_000.000000028.h5
1.6M -rw-rw-r--  1.6M Jun 28 12:50 plasma_wave_State_000.000000032.h5
1.6M -rw-rw-r--  1.6M Jun 28 12:51 plasma_wave_State_000.000000036.h5
1.6M -rw-rw-r--  1.6M Jun 28 12:51 plasma_wave_State_000.000000040.h5
 72K -rw-rw-r--   71K Jun 28 12:51 std.out
```

Multiple additional files have been created, which are are named **Projectname_State_Timestamp.h5**. They contain the solution vector of the equation system variables at each interpolation nodes at the given time, which corresponds to multiples of **Analyze_dt**. If these files are not present, something went wrong during the execution of **piclas**. In that case, check the `std.out` file for an error message.

After a successful completion, the last lines in this file should look as shown below:

```
-------------------------------------------------------------------------------------------
↪---
Sys date  :    03.07.2021 14:34:26
PID: CALCULATION TIME PER TSTEP/DOF: [ 5.85952E-05 sec ]
EFFICIENCY: SIMULATION TIME PER CALCULATION in [s]/[Core-h]: [ 2.38587E-06 sec/h ]
Timestep  :    5.0000000E-10
#Timesteps :    8.0000000E+01
WRITE STATE TO HDF5 FILE [plasma_wave_State_000.000000040.h5] ...DONE  [.008s]
#Particles :    8.0000000E+02
-------------------------------------------------------------------------------------------
↪---
===========================================================================================
```

(continues on next page)

(continued from previous page)

```
PICLAS FINISHED! [           60.42 sec ] [     0:00:01:00]
================================================================================================
```

Visualization (post-processing)

To visualize the solution, the *State*-files must be converted into a format suitable for **ParaView**, **VisIt** or any other visualisation tool for which the program **piclas2vtk** is used.

The parameters for **piclas2vtk** are stored in the **parameter.ini** file under

```
! ============================================================================== !
! piclas2vtk
! ============================================================================== !
NVisu         = 10 ! Polynomial degree used for the visualization when the .h5 file is␣
↪converted to .vtu/.vtk format. Should be at least N+1
VisuParticles = T  ! Activate the conversion of particles from .h5 to .vtu/.vtk format.␣
↪Particles will be displayed as a point cloud with properties, such as velocity,␣
↪species ID, etc.
```

where `NVisu` is the polynomial visualization degree on which the field solution is interpolated. Depending on the used polynomial degree `N` and subsequently the degree of visualization `NVisu`, which should always be higher than `N`, the resulting electric potential Φ and its derivative the electric field strength $\mathbf{E}$ might show signs of oscillations. This is because the PIC simulation is always subject to noise that is influenced by the discretization (number of elements and polynomial degree as well as number of particles) and is visible in the solution as this is a snapshot of the current simulation.

Additionally, the flag `VisuParticles` activates the output of particle position, velocity and species to the *vtk*-files.

Run the command

```
./piclas2vtk parameter.ini plasma_wave_State_000.000000*
```

to generate the corresponding *vtk*-files, which can then be loaded into the visualisation tool.

The electric potential field can be viewed, e.g., by opening `plasma_wave_Solution_000.000000040.vtu` and plotting the field `Phi` along the x-axis, which should look like the following

1.7.2 Adiabatic Box/Reservoir (DSMC, Relaxation/Chemistry)

An essential feature of DSMC simulations is their ability to treat thermal and chemical non-equilibrium in a physically correct manner. A simple example for such a use case are adiabatic box/reservoir simulations, which start in a non-equilibrium state. The subsequent simulation should lead to the correct relaxation towards equilibrium. Hence, this tutorial provides an example case at thermal non-equilibrium with disabled chemistry and another based on chemical non-equilibrium with chemistry enabled.

Before beginning with the tutorial, copy the `dsmc-reservoir` directory from the tutorial folder in the top level directory to a separate location

```
cp -r $PICLAS_PATH/tutorials/dsmc-reservoir .
cd dsmc-reservoir
```

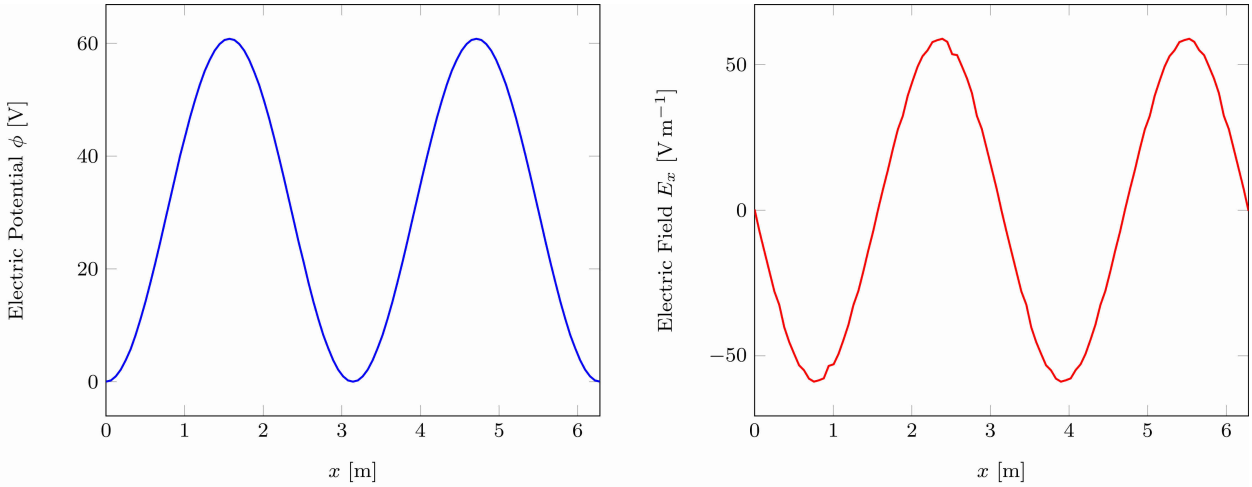

Fig. 1.2: Resulting electric potential and field.

Mesh Generation with HOPR (pre-processing)

Before the actual simulation is conducted, a mesh file in the HDF5 format has to be supplied. The mesh files used by **piclas** are created by supplying an input file *hopr.ini* with the required information for a mesh that has either been created by an external mesh generator or directly from block-structured information in the hopr.ini file itself. Here, a block-structured grid is created directly from the information in the hopr.ini file. To create the .h5 mesh file, simply run

```
hopr hopr.ini
```

This creates the mesh file *dsmc_reservoir_mesh.h5* in HDF5 format, which is depicted in Fig. 1.3. Alternatively, if you do not want to run **hopr** yourself, you can also use the provided mesh. After this step, the mesh needs to be copied in the simulation directories

```
cp dsmc_reservoir_mesh.h5 ./chemistry-off/
cp dsmc_reservoir_mesh.h5 ./chemistry-on/
```

The size of the simulation domain is set to $[4.64e-6 \times 4.64e-6 \times 4.64e-6]$ m^3 and is defined by the single block information, where each node of the hexahedral block is defined

```
Corner          =   (/0.0,0.0,0.0,, 1.0,0.0,0.0,, ... /)
```

Fig. 1.3: Order of the corners to define the used mesh. The first node is placed at the origin.

Afterwards this element is scaled via

```
postScaleMesh  = T
meshScale      = 4.64E-6
```

The number of mesh elements for the block in each direction can be adjusted by changing the line

```
nElems         = (/2,2,1/)
```

However, for an adiabatic box/reservoir simulation, a single element would be sufficient/optimal. Each side of the block has to be assigned a boundary index, which corresponds to the boundaries defined in the next steps. Due to the fact, that all boundaries in this example should behave similar, only one index is needed.

```
BCIndex        = (/1,1,1,1,1,1/)
```

The field boundaries can directly be defined in the hopr.ini file (contrary to the particle boundary conditions, which are defined in the parameter.ini). For particle-based methods without electromagnetic fields, the boundary type is set in the parameter.ini.

```
!================================================================================= !
! BOUNDARY CONDITIONS
!================================================================================= !
  BoundaryName = BC_wall
  BoundaryType = (/4,0,0,0/)
```

For more information about hopr, visit https://github.com/hopr-framework/hopr.

Simulation: Chemistry disabled

Install **piclas** by compiling the source code as described in Chapter *Installation* and make sure to set the correct compile flags (ie. chose the correct simulation method)

```
PICLAS_TIMEDISCMETHOD = DSMC
```

or simply run the following command from inside the *build* directory

```
cmake ../ -DPICLAS_TIMEDISCMETHOD=DSMC
```

to configure the build process and run `make` afterwards to build the executable. It is recommended to either utilize a separate build folder (e.g. build_DSMC/) or to delete the contents of the folder beforehand to avoid conflicts between different compiler options (e.g. the setting `PICLAS_EQNSYSNAME = poisson` from the plasma wave tutorial is in conflict with the DSMC method). An overview over the available solver and discretization options is given in Section *Solver settings*. The physical parameters for this test case are summarized in Table 1.7.

Table 1.7: Physical properties at the simulation start

Property	Value
Species	CO_2
Molecule mass m_{CO_2}	$7.306e-26kg$
Number density n_{CO_2}	$1e23m^{-3}$
Translational temperature T_{trans}	$10000K$
Rotational temperature T_{rot}	$7500K$
Vibrational temperature T_{vib}	$5000K$

General numerical setup

The general numerical parameters are selected by the following

```
! ================================================================================= !
! MESH
! ================================================================================= !
MeshFile   = dsmc_reservoir_mesh.h5
! ================================================================================= !
! PROJECT
```

(continues on next page)

(continued from previous page)

```
! ================================================================================= !
ProjectName     = dsmc_reservoir_chemisty_off
TrackingMethod  = triatracking
```

where, the path to the mesh file `MeshFile`, project name and particle tracking method `TrackingMethod` are chosen. The temporal parameters of the simulation are controlled via

```
! ================================================================================= !
! CALCULATION
! ================================================================================= !
! Time
TEnd                   = 2E-6
ManualTimeStep         = 1.0E-8
Analyze_dt             = 5E-6
IterDisplayStep        = 100
Particles-HaloEpsVelo = 5000
```

where the final simulation time `tend` [s], the time step for the field and particle solver is set via `ManualTimeStep` [s]. The time between restart/checkpoint file output is defined via `Analyze_dt` (which is also the output time for specific analysis functions in the field solver context). The number of time step iterations `IterDisplayStep` defines the interval between information output regarding the current status of the simulation, which is written to std.out. The `Particles-HaloEpsVelo` [m/s] determines the size of the halo region for MPI communication and should not be smaller than the fastest particles in the simulation.

Analysis setup

For this case our focus is on the run-time analysis to investigate the transient behavior of the reservoir. The first parameter `Part-AnalyzeStep` allows to perform the output every N^{th} iteration to reduce the size of the output file and to increase the computational speed. Different parameters for run-time analysis can be enabled, in this case the number of particles per species (`CalcNumSpec`) and the temperature output (`CalcTemp`). It is also recommended to enable `Particles-DSMC-CalcQualityFactors`, which provides outputs to evaluate the quality of the simulation results such as the mean and maximum collision probabilities. The parameter `TimeStampLength = 13` reduces the output filename length. It can be needed for postprocessing, as e.g. ParaView sometimes does not sort the files correctly if the timestamps are too long. The displayed time solution would then be faulty.

```
! ================================================================================= !
! Particle Analysis
! ================================================================================= !

Part-AnalyzeStep = 1
CalcNumSpec = T
CalcTemp    = T
Particles-DSMC-CalcQualityFactors = T
TimeStampLength  = 13
```

All available options with a short description can be displayed using the help of PICLas:

```
piclas --help 'Particle Analyze'
```

Boundary conditions

The boundary conditions are set by the following lines

```
! ============================================================================== !
! Boundaries
! ============================================================================== !
Part-nBounds              = 1
Part-Boundary1-SourceName = BC_wall
Part-Boundary1-Condition  = reflective
```

where, the number of boundaries `Part-nBounds` is followed by the names of the boundaries (given by the hopr.ini file) and the type `reflective`.

Particle solver

For the treatment of particles, the maximum number of particles `Part-maxParticleNumber` that each processor can hold has to be supplied and the number of particle species `Part-nSpecies` that are used in the simulation (created initially or during the simulation time through chemical reactions).

```
! ============================================================================== !
! PARTICLES
! ============================================================================== !
Part-maxParticleNumber = 420000
Part-nSpecies          = 1
Part-FIBGMdeltas       = (/4.64E-6,4.64E-6,4.64E-6/)
```

The inserting (sometimes labelled emission or initialization) of particles at the beginning or during the course of the simulation is controlled via the following parameters. For each species, the mass (`Part-Species[$]-MassIC`), charge (`Part-Species[$]-ChargeIC`) and weighting factor w (`Part-Species[$]-MacroParticleFactor`) have to be defined.

```
! ============================================================================== !
! Species1 - CO2
! ============================================================================== !
Part-Species1-MassIC              = 7.306E-26
Part-Species1-ChargeIC            = 0
Part-Species1-MacroParticleFactor = 5E2
```

The number of initialization sets is defined by `Part-Species[$]-nInits`, where each initialization set is accompanied by a block of parameters that is preceded by the corresponding `-Init[$]` counter. In this example we have a single initialization set. The `Part-Species[$]-Init[$]-SpaceIC = cuboid` flag defines the type of the initialization set. Here, the particles are placed in a cuboid which is spanned by its base plane (`Part-Species[$]-Init[$]-BasePointIC`, `Part-Species[$]-Init[$]-BaseVector1IC`, `Part-Species[$]-Init[$]-BaseVector2IC`), a normal (`Part-Species[$]-Init[$]-NormalIC`) and its height (`Part-Species[$]-Init[$]-CuboidHeightIC`). Each type of the initialization set might have a different set of parameters and an overview is given in Section *Particle Initialization & Emission*. Here, simulation particles are inserted at a translational temperature (`MWTemperatureIC`) of $10000K$ using a Maxwellian velocity distribution (`velocityDistribution`), a vibrational temperature (`TempVib`) of $5000K$, a rotational temperature (`TempRot`) of $7500K$ at a zero flow velocity, which is defined through a velocity vector (`VeloVecIC`, will be normalized internally) and its magnitude (`VeloIC`).

```
Part-Species1-nInits = 1

Part-Species1-Init1-SpaceIC              = cuboid
Part-Species1-Init1-velocityDistribution = maxwell_lpn
Part-Species1-Init1-MWTemperatureIC      = 10000
Part-Species1-Init1-TempVib              = 5000
Part-Species1-Init1-TempRot              = 7500
Part-Species1-Init1-PartDensity          = 1E23
Part-Species1-Init1-BasePointIC          = (/0.,0.,0./)
Part-Species1-Init1-BaseVector1IC        = (/4.64E-6,0.,0./)
Part-Species1-Init1-BaseVector2IC        = (/0.,4.64E-6,0./)
Part-Species1-Init1-NormalIC             = (/0.,0.,1./)
Part-Species1-Init1-CuboidHeightIC       = 4.64E-6
Part-Species1-Init1-VeloIC               = 0
Part-Species1-Init1-VeloVecIC            = (/0.,0.,1./)
```

To calculate the number of simulation particles defined by `Part-Species[$]-Init[$]-PartDensity`, the selected weighting factor w_{CO_2} and the volume of the complete domain $V = (4.64e-6m)^3$ are utilized. This value can be used to chose the maximum particle number per processor accordingly.

$$N_{CO_2,\text{sim}} = \frac{n_{CO_2} V}{w_{CO_2}}$$

DSMC setup

Finally, DSMC has to be enabled (`UseDSMC = T`) and the particle movement is disabled via `Particles-DSMCReservoirSim = T` to reduce the computational effort. Besides these settings `Particles-DSMC-CollisMode` is an important parameter. If set to 1, only elastic collisions are performed, if set to 2 relaxation processes are allowed and if set to 3 chemistry is enabled. Additionally, constant values for the rotational (`Particles-DSMC-RotRelaxProb`) and vibrational (`Particles-DSMC-VibRelaxProb`) relaxation probabilities are defined.

```
! ============================================================================= !
! DSMC
! ============================================================================= !
UseDSMC                     = T
Particles-DSMCReservoirSim  = T
Particles-DSMC-CollisMode   = 2
Particles-DSMC-RotRelaxProb = 0.2
Particles-DSMC-VibRelaxProb = 0.02
```

Besides the data given in the **parameter.ini**, a proper DSMC simulation needs additional species information, which is defined in the **DSMC.ini**. The species numeration needs to be the same in both files.

```
! ============================================================================= !
! Species1, CO2
! ============================================================================= !
Part-Species1-SpeciesName    = CO2
Part-Species1-InteractionID  = 2
Part-Species1-PolyatomicMol  = T
Part-Species1-NumOfAtoms     = 3
Part-Species1-LinearMolec    = T
```

(continues on next page)

(continued from previous page)

```
Part-Species1-Tref           = 273
Part-Species1-dref           = 5.10E-10
Part-Species1-omega          = 0.24
Part-Species1-CharaTempVib1  = 959.66
Part-Species1-CharaTempVib2  = 959.66
Part-Species1-CharaTempVib3  = 1918.6
Part-Species1-CharaTempVib4  = 3382
Part-Species1-CharaTempRot   = 0.6
Part-Species1-Ediss_eV       = 5.45
```

The first block from `Part-Species[$]-InteractionID` to `Part-Species[$]-LinearMolec` declares the structure of the species. Available species types set by `Part-Species[$]-InteractionID` are listed in Section *Species Definition*. The second one from `Part-Species[$]-Tref` to `Part-Species[$]-omega` are the basis of the collision model utilized, in this case the Variable Hard Sphere (VHS) model. It is important to keep in mind that the ω in this file differs from the ω used by Bird. $\omega = \omega_{\text{bird1994}} - 0.5$. The last block from `Part-Species[$]-CharaTempVib1` to `Part-Species[$]-CharaTempRot` defines the vibrational excitation modes and sets the characteristic rotational temperature (utilized for the partition function). Finally, `Part-Species1-Ediss_eV` defines the dissociation energy of the molecule in [eV].

Running the code

The command

```
./piclas parameter.ini DSMC.ini | tee std.out
```

executes the code and dumps all output into the file *std.out*. If the run has completed successfully, which should take only a brief moment, the contents of the working folder should look like

```
drwxrwxr-x 4,0K Okt 21 07:59 ./
drwxrwxr-x 4,0K Dez  4 11:09 ./
drwxrwxr-x 4,0K Dez  3 01:02 ../
-rw-rw-r-- 1,4K Dez  4 11:05 DSMC.ini
-rw-rw-r-- 8,7K Dez  4 11:09 dsmc_reservoir_chemisty_off_DSMCState_000.000005000.h5
-rw-rw-r-- 1,8M Dez  4 11:09 dsmc_reservoir_chemisty_off_State_000.000000000.h5
-rw-rw-r-- 1,8M Dez  4 11:09 dsmc_reservoir_chemisty_off_State_000.000005000.h5
-rw-rw-r-- 6,9K Nov  1 05:05 dsmc_reservoir_mesh.h5
-rw-rw-r--  931 Dez  4 11:09 ElemTimeStatistics.csv
-rw-rw-r-- 7,5K Dez  4 10:49 parameter.ini
-rw-rw-r-- 587K Dez  4 11:09 PartAnalyze.csv
-rw-rw-r--  58K Dez  4 11:09 std.out
-rw-rw-r--  15K Dez  4 11:09 userblock.tmp
```

Multiple additional files have been created, which are are named **Projectname_State_Timestamp.h5**. They contain the particle information (position, velocity, energy) and the solution vector of the equation system variables (if a field solver had been utilized) at each interpolation node at the given time, which corresponds to multiples of **Analyze_dt**. If these files are not present, something went wrong during the execution of **piclas**. In that case, check the `std.out` file for an error message and feel free to open an issue.

After a successful completion, the last lines in this file should look as shown below:

```
-------------------------------------------------------------------------------
 Sys date  :    04.12.2021 11:05:43
```

(continues on next page)

(continued from previous page)

```
 PID: CALCULATION TIME PER TSTEP/DOF: [ 1.77323E-04 sec ]
 EFFICIENCY: SIMULATION TIME PER CALCULATION in [s]/[Core-h]: [ 1.19627E-03 sec/h ]
 Timestep   :    2.0000000E-09
#Timesteps :    2.5000000E+03
 WRITE STATE TO HDF5 FILE [dsmc_reservoir_chemisty_off_State_000.000005000.h5] ...DONE ␣
↪[.005s]
#Particles :    1.9979000E+04
-------------------------------------------------------------------------------
===============================================================================
 PICLAS FINISHED! [            3.82 sec ] [     0:00:00:03]
===============================================================================
```

Visualization, chemistry disabled (post-processing)

In addition to `std.out`, which contains information about the simulation process, three other important file types were created. The simulation results are stored in `Projectname_State_Timestamp.h5`, `Projectname_DSMCState_Timestamp.h5` and `PartAnalyze.csv`. The latter stores the data generated during each the time step (or every `Part-AnalyzeStep`) and can be analyzed with e.g. gnuplot, any spreadsheet tool or ParaView. It allows the visualization of the transient development of the selected parameters (see Section *Analysis setup*). As an example, the process of the examined relaxation, i.e. the convergence of translational, rotational and vibrational temperature, is shown in Fig. 1.4.

Fig. 1.4: Temperature relaxation process towards thermal equilibrium.

Simulation: Chemistry enabled

In the next step, we add more species and chemical reactions to the simulation, focusing on the different input parameters. Besides changes at simulation and output generation times the most important modification concerning the **parameter.ini** and **DSMC.ini** is related to chemistry. The definition of more than one species is needed to capture the products of the chemical reactions. The input of new species is similar to the first species as shown in in Section *Particle solver* and *DSMC setup*. The values of the general physical properties are listed in Table 1.8. As can be seen the number density was slightly reduced and the species are initialized in thermal equilibrium.

Table 1.8: Physical properties at the simulation start

Property	Value
Species	CO_2, CO, O
Molecule mass m_{CO_2}	$7.306E-26kg$
Molecule mass m_{CO}	$4.65100E-26kg$
Molecule mass m_{O}	$2.65700E-26kg$
Number density $n_{CO_2} = n_{CO} = n_{O}$	$1e22m^{-3}$
Translational temperature $T_{\text{trans, CO}_2} = T_{\text{trans, CO}} = T_{\text{trans, O}}$	$10000K$
Rotational temperature $T_{\text{rot, CO}_2} = T_{\text{rot, CO}}$	$10000K$
Vibrational temperature $T_{\text{vib, CO}_2} = T_{\text{vib, CO}}$	$10000K$

Reactions

The parameter `Particles-DSMC-CollisMode = 3` mentioned in *DSMC setup* must be changed to enable the use of the chemistry module. `Particles-DSMC-BackwardReacRate = T` activates the calculation of the backward reaction to every considered reaction. The reaction rates are added to the output by setting `CalcReacRates`.

```
Particles-DSMC-CollisMode        = 3
Particles-DSMC-BackwardReacRate  = T
CalcReacRates = T
```

While the electronic model is disabled by `Particles-DSMC-ElectronicModel = 0`, a `Particles-DSMCElectronicDatabase` has to be provided for the calculation of the partition functions for the equilibrium constant required for the calculation of the backward reaction rates. It contains the species-specific electronic energy levels, for more information see Secion *Electronic Relaxation*.

```
Particles-DSMC-ElectronicModel   = 0
Particles-DSMCElectronicDatabase = DSMCSpecies_electronic_state_full_Data.h5
```

Copy one of the databases, which are used for the regression testing, to your current folder, e.g. from the path below

```
cp $PICLAS_PATH/regressioncheck/WEK_Reservoir/CHEM_EQUI_diss_CH4/DSMCSpecies_electronic_
↪state_full_Data.h5 ./
```

The definition of the reactions needs to be given in the **DSMC.ini** file. Additionally, the heat of formation of each species has to be provided for the calculation of the heat of reaction [K].

```
Part-Species1-HeatOfFormation_K = -47328.35
Part-Species2-HeatOfFormation_K = -13293.70
Part-Species3-HeatOfFormation_K =  29969.45
```

As with the species, the number of reactions must first be defined by `DSMC-NumOfReactions`. Next, a model has to be chosen for each reaction via `DSMC-Reaction[$]-ReactionModel`. In this example the TCE model is used. Thus the connected parameters of the Arrhenius equation `DSMC-Reaction[$]-Arrhenius-Prefactor`, `DSMC-Reaction[$]-Arrhenius-Powerfactor` and `DSMC-Reaction[$]-Activation-Energy_K` must be set. The reactants of each reaction are defined as follows: `DSMC-Reaction[$]-Reactants` contains the species number of up to three reactants and `DSMC-Reaction[$]-Products` contains the species number of up to four products. Nonreactive parts are mentioned in another list, the length of which is given by `DSMC-Reaction[$]-NumberOfNonReactives`. The list `DSMC-Reaction[$]-NonReactiveSpecies` is filled with the species numbers. If the reaction rate depends on the non-reaction partner (e.g. higher dissociation rate if the non-reactive partner is an atom), each reaction can be defined separately by simply defining the second reactant and product, respectively.

```
DSMC-NumOfReactions = 1
DSMC-Reaction1-ReactionModel         = TCE
DSMC-Reaction1-Reactants             = (/1,0,0/)
DSMC-Reaction1-Products              = (/2,0,3,0/)
DSMC-Reaction1-Arrhenius-Prefactor   = 1.15E-08
DSMC-Reaction1-Arrhenius-Powerfactor = -1.5
DSMC-Reaction1-Activation-Energy_K   = 63280
DSMC-Reaction1-NumberOfNonReactives  = 3
DSMC-Reaction1-NonReactiveSpecies    = (/1,2,3/)
```

Therefore, in this example with one reaction and each of the three species as possible non-reactive partner as well as the corresponding backward reaction lead to a number of six reactions in total. For more information see Section *Chemistry & Ionization*.

Running the code (Parallel computing)

In order to investigate the transient behavior, a longer simulation time was chosen. This results in comparatively long computing times, which is why the use of several computing cores is recommended. The number of cores may not exceed the number of cells. This results in a maximum of 4 cores for the described simulation. Another important note is that bash does not understand aliases which are not at the start of a line. Thus a copy of the **piclas** binary must be located in the current folder

```
cp $PICLAS_PATH/build/bin/piclas .
```

or the whole path to the binary must be used instead. Assuming a run with 4 cores is desired and the **piclas** binary is located at the current directory, the command

```
mpirun -np 4 piclas parameter.ini DSMC.ini | tee std.out
```

executes the code and dumps all output into the file *std.out*.

Visualization, chemistry enabled (post-processing)

To visualize the solution, the *State*-files must be converted into a format suitable for **ParaView**, **VisIt** or any other visualisation tool for which the program **piclas2vtk** is used.

The parameters for **piclas2vtk** are stored in the **parameter.ini** file under

```
! ============================================================================== !
! piclas2vtk
! ============================================================================== !
NVisu         = 1
VisuParticles = T
```

where `NVisu` is the polynomial visualization degree on which the field solution is interpolated. The flag `VisuParticles` activates the output of particle position, velocity and species, which is disabled per default due to the usually large number of particles.

Run the command

```
./piclas2vtk parameter.ini dsmc_reservoir_chemisty_on_State_000.00000*
```

to convert the HDF5 file to the binary VTK format (`*.vtu`), which can then be opened with e.g. ParaView.

The `Projectname_visuPart_Timestamp.vtu` files contain simulation particle specific data like their position, temperature and species. The figure below illustrates the initiated species and the resulting change of species. Additionaly, the particle information can be used for e.g. the determination of the velocity and energy distribution function of each species. It should be mentioned that the particle movement was disabled and therefore each reaction product stays at the position of the given reactant.

While the figure above is only capable of giving a general overview about the processes in the reservoir, an analysis of the `PartAnalyze.csv` shows, that nearly all carbon dioxide is dissociated at the end of the simulation. During this process the overall temperature and the reaction probabilities are decreasing.

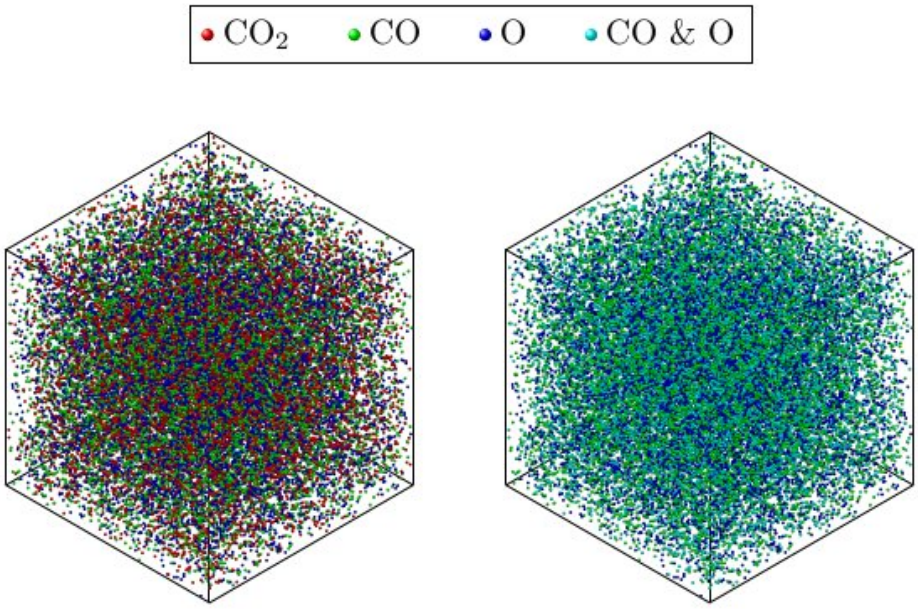

Fig. 1.5: Comparison of the present species (simulation particles) at start (left) and end (right) time of the simulation.

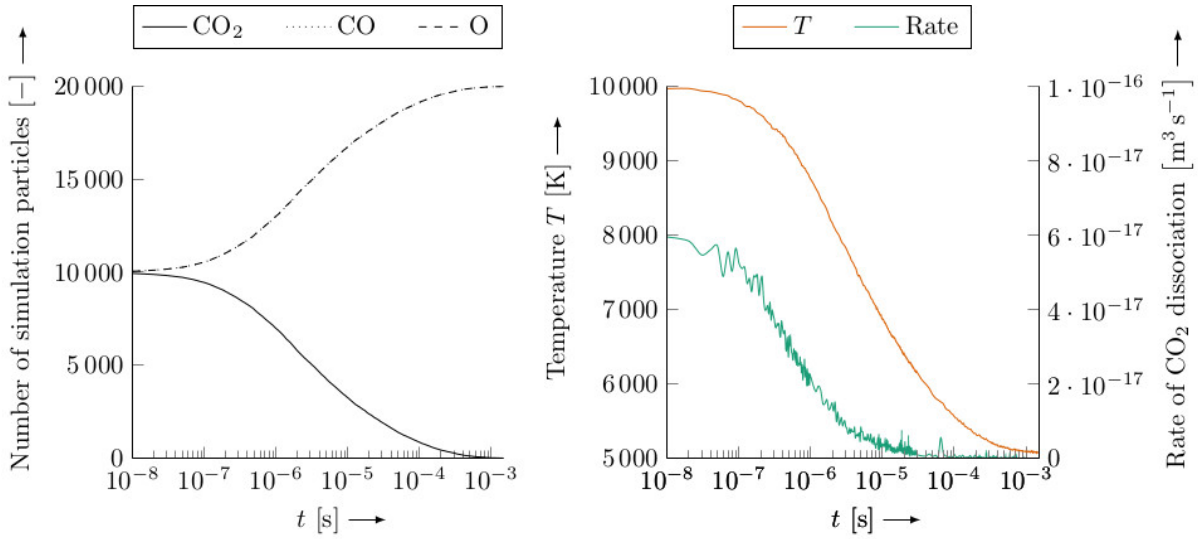

Fig. 1.6: Development of species composition (left) and translational temperature and dissociation rate (right) over time.

1.7.3 Hypersonic Flow around the 70° Cone (DSMC) - 2D Mesh

A widespread validation case for rarefied gas flows is the wind tunnel test of the 70° blunted cone in a hypersonic, diatomic nitrogen flow [72], [73]. Before beginning with the tutorial, copy the `dsmc-cone` directory from the tutorial folder in the top level directory to a separate location

```
cp -r $PICLAS_PATH/tutorials/dsmc-cone-2D .
cd dsmc-cone-2D
```

The general information needed to setup a DSMC simulation is given in the previous tutorial *Adiabatic Box/Reservoir (DSMC, Relaxation/Chemistry)*. The following focuses on case-specific differences.

Mesh Generation with HOPR (pre-processing)

Before the actual simulation is conducted, a mesh file in the HDF5 format has to be supplied. The mesh files used by **piclas** are created by supplying an input file `hopr.ini` with the required information for a mesh that has either been created by an external mesh generator or directly from block-structured information in the `hopr.ini` file itself. Here, a conversion from an external mesh generator is required. The mesh and the corresponding boundaries are depicted in Fig. 1.7.

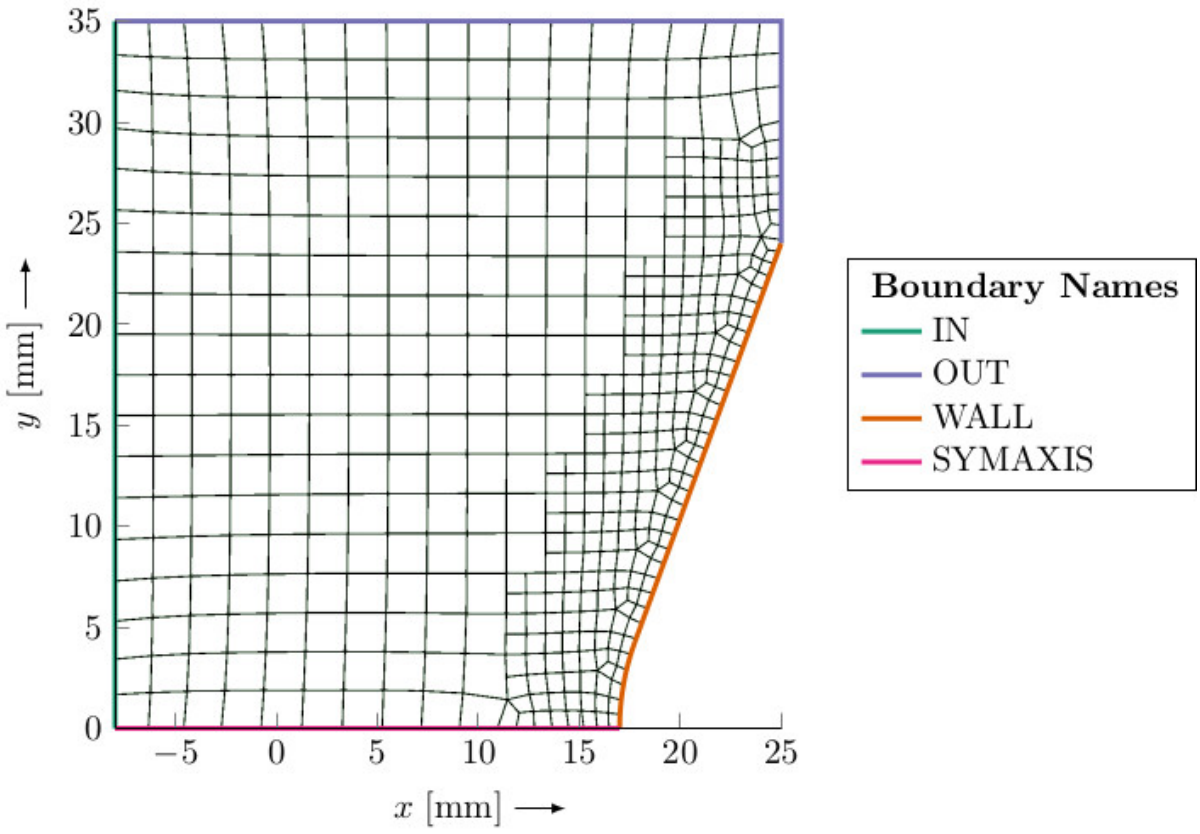

Fig. 1.7: Mesh of the 70° Cone.

Mesh generated by HEXPRESS (CGNS format)

An example setup for such a conversion is given in `hopr.ini`. In difference to a direct generation the `FileName` of the original mesh has to be specified and the `Mode` has to be set to 3, to read a CGNS mesh. For unstructured ANSA CGNS meshes the `BugFix_ANSA_CGNS` needs to be active.

```
!================================================================================== !
! MESH
!================================================================================== !
FileName         = 70degCone_2D_mesh.cgns
Mode             = 3
BugFix_ANSA_CGNS = T
```

The `BoundaryName` needs to be the same as in the CGNS file.

```
!================================================================================== !
! BOUNDARY CONDITIONS
!================================================================================== !
BoundaryName = IN
BoundaryType = (/3,0,0,0/)
BoundaryName = OUT
BoundaryType = (/3,0,0,0/)
BoundaryName = WALL
BoundaryType = (/4,0,0,0/)
BoundaryName = SYMAXIS
BoundaryType = (/4,0,0,0/)
BoundaryName = ROTSYM
BoundaryType = (/4,0,0,0/)
```

To create the `.h5` mesh file, you would simply run

```
hopr hopr_cgns.ini
```

This would create the mesh file `70degCone_2D_mesh.h5` in HDF5 format. For more information see directly at https://github.com/hopr-framework/hopr.

Mesh generated by Gmsh (msh format)

Instead of using the given `.cgns` mesh file, the mesh can be generated by using the open-source mesh generator Gmsh.

First, create a new file in gmsh: `70DegCone_2DSurf.geo`. In general, the mesh can be generated using the GUI or by using the `.geo` script environment. In the GUI, the script can be edited via `Edit script` and loaded with `Reload script`. This tutorial focuses on the scripting approach.

After opening the `.geo` script file, select the OpenCASCADE CAD kernel and open the provided `70DegCone_2DSurf.step` file with the following commands:

```
SetFactory("OpenCASCADE");
v() = ShapeFromFile("70degCone_2DSurf.step");
```

Two-dimensional and axisymmetric simulations require a mesh in the xy-plane, where the x-axis is the rotational axis and y ranges from zero to a positive value. Additionally, the mesh shall be centered around zero in the z-direction with a single cell row. As the loaded file only consists of a surface, which is meshed with Gmsh and then extruded using Hopr, the surface needs to be translated:

```
Translate {0, 0, -1} {
    Surface{1};
}
```

Physical groups are used to define the boundary conditions at all curves:

```
Physical Curve("SYMAXIS") = {3};
Physical Curve("WALL") = {1, 2};
Physical Curve("IN") = {4};
Physical Curve("OUT") = {5, 6};
```

The mesh options can be set with the following commands:

```
Mesh.MeshSizeMin = 0;
Mesh.MeshSizeMax = 10;
Field[1] = MathEval;
Field[1].F = "0.2";
Field[2] = Restrict;
Field[2].EdgesList = {1, 2};
Background Field = 2;
Mesh.Algorithm = 1;
Mesh.RecombinationAlgorithm = 2;
```

The commands `Mesh.MeshSizeMin` and `Mesh.MeshSizeMax` define the minimum and maximum mesh element sizes. With the prescribed `Field` options, the size of the mesh can be specified using an explicit mathematical function using `MathEval` and restriced to specific surfaces with `Restrict`. In this tutorial, a mesh refinement at the frontal wall of the cone is enabled with this. `Background Field = 2` sets `Field[2]` as background field. Different meshing algorithms for creating the 2D and 3D meshes can be chosen within Gmsh.

Next, the 2D mesh is created and all cells are recombined to quads with the following commands:

```
Mesh.RecombineAll = 1;
Mesh 2;
```

The following commands are required to save all elements even if they are not part of a physical group and to use the ASCII format, before saving the mesh as `70degCone_2D.msh`:

```
Mesh.SaveAll = 1;
Mesh.Binary = 0;
Mesh.MshFileVersion = 4.1;
Save "70degCone_2D.msh";
```

The mesh can be created by simplying executing Gmsh from the terminal:

```
gmsh 70degCone_2DSurf.geo
```

The resulting mesh shall consist of quad elements and not triangles. Finally, it has to be converted to the file format used by **piclas** using HOPR by supplying an input file `hopr.ini` using the corresponding mode:

```
Mode = 5
```

To extrude the 2D quadrangular mesh to a 3D hexahedral mesh with only one element in the third direction, the following commands next to be set in the `hopr.ini`:

```
MeshDim      = 2
zLength      = 1
nElemsZ      = 1
lowerZ_BC    = (/3,0,0,0/)
upperZ_BC    = (/3,0,0,0/)
```

While `zLength` specifies the length of the mesh in z-direction, `nElemsZ` ensures a single cell row. `lowerZ_BC` and `upperZ_BC` are the boundary conditions on the surfaces parallel to the xy-plane, that need to be defined as stated.

Again, to create the `.h5` mesh file, you would simply run

```
hopr hopr_gmsh.ini
```

This would create the mesh file `70degCone_2D_mesh.h5` in HDF5 format.

Flow simulation with DSMC

Install **piclas** by compiling the source code as described in Section *Installation* and make sure to set the correct compile flags. For this setup, we are utilizing the regular Direct Simulation Monte Carlo (DSMC) method

```
PICLAS_TIMEDISCMETHOD = DSMC
```

or simply run the following command from inside the *build* directory

```
cmake ../ -DPICLAS_TIMEDISCMETHOD=DSMC
```

to configure the build process and run `make` afterwards to build the executable. It is recommended to either utilize a separate build folder (e.g. build_DSMC/) or to delete the contents of the folder beforehand to avoid conflicts between different compiler options (e.g. the setting `PICLAS_EQNSYSNAME = poisson` from the plasma wave tutorial is in conflict with the DSMC method). An overview over the available solver and discretization options is given in Section *Solver settings*. The values of the general physical properties are listed in Table 1.9.

Table 1.9: Physical properties at the simulation start

Property	Value (initial and surfaceflux)
Species	N_2
Molecule mass m_{N_2}	$4.65e-26kg$
Number density n	$3.715e+20m^{-3}$
Translational temperature T_{trans}	$13.3K$
Rotational temperature T_{rot}	$13.3K$
Vibrational temperature T_{vib}	$13.3K$
Velocity v_{x}	$1502.57\frac{m}{s}$

To define an incoming, continuous flow, the procedure is similar to that for initialization sets. For each species, the number of inflows is specified via `Part-Species[$]-nSurfaceFluxBCs`. Subsequently, the boundary from which it should flow in is selected via `Part-Species[$]-Surfaceflux[$]-BC`. The rest is identical to the initialization explained in Section *Particle solver*.

```
Part-Species1-nSurfaceFluxBCs = 1

Part-Species1-Surfaceflux1-BC                    = 1
Part-Species1-Surfaceflux1-velocityDistribution = maxwell_lpn
Part-Species1-Surfaceflux1-MWTemperatureIC       = 13.3
```

(continues on next page)

(continued from previous page)

```
Part-Species1-Surfaceflux1-TempVib              = 13.3
Part-Species1-Surfaceflux1-TempRot              = 13.3
Part-Species1-Surfaceflux1-PartDensity          = 3.715E+20
Part-Species1-Surfaceflux1-VeloIC               = 1502.57
Part-Species1-Surfaceflux1-VeloVecIC            = (/1.,0.,0./)
```

Exploiting symmetry

In axially symmetrical cases, the simulation effort can be greatly reduced. For this, 2D must first be activated via `Particles-Symmetry-Order = 2`. `Particles-Symmetry2DAxisymmetric = T` enables axisymmetric simulations.

```
! Symmetry
Particles-Symmetry-Order         = 2
Particles-Symmetry2DAxisymmetric = T
```

First of all, certain requirements are placed on the grid. The y-axis acts as the symmetry axis, while the x-axis defines the radial direction. Therefore grid lies in the xy-plane and should have an extension of one cell in the z-direction, the extent in z-direction is irrelevant whilst it is centered on $z = 0$. In addition, the boundary at $y = 0$ must be provided with the condition `symmetric_axis` and the boundaries parallel to the xy-plane with the condition `symmetric`.

```
Part-Boundary4-SourceName  = SYMAXIS
Part-Boundary4-Condition   = symmetric_axis
```

For the `.cgns` mesh, the following commands need to be enabled:

```
Part-nBounds               = 5
Part-Boundary5-SourceName  = ROTSYM
Part-Boundary5-Condition   = symmetric
```

For the `.msh` mesh instead, the following commands need to be enabled:

```
Part-nBounds               = 6
Part-Boundary5-SourceName  = lowerZ_BC
Part-Boundary5-Condition   = symmetric
Part-Boundary6-SourceName  = upperZ_BC
Part-Boundary6-Condition   = symmetric
```

To fully exploit rotational symmetry, a radial weighting can be enabled via `Particles-RadialWeighting = T`, which will linearly increase the weighting factor towards y_{max}, depending on the current y-position of the particle. Thereby the `Particles-RadialWeighting-PartScaleFactor` multiplied by the `MacroParticleFactor` is the weighting factor at y_{max}. Since this position based weighting requires an adaptive weighting factor, particle deletion and cloning is necessary. `Particles-RadialWeighting-CloneDelay` defines the number of iterations in which the information of the particles to be cloned are stored and `Particles-RadialWeighting-CloneMode = 2` ensures that the particles from this list are inserted randomly after the delay.

```
! Radial Weighting
Particles-RadialWeighting                = T
Particles-RadialWeighting-PartScaleFactor = 60
Particles-RadialWeighting-CloneMode      = 2
Particles-RadialWeighting-CloneDelay     = 5
```

For further information see *2D/Axisymmetric Simulations*.

Octree

By default, a conventional statistical pairing algorithm randomly pairs particles within a cell. Here, the mesh should resolve the mean free path to avoid numerical diffusion. To circumvent this requirement, an octree-based sorting and cell refinement can be enabled by `Particles-DSMC-UseOctree = T`. The resulting grid is defined by the maximum number `Particles-OctreePartNumNode` and minimum number `Particles-OctreePartNumNodeMin` of particles in each subcell. Furthermore, the search for the nearest neighbour can be enabled by `Particles-DSMC-UseNearestNeighbour = T`.

```
! Octree
Particles-DSMC-UseOctree           = T
Particles-DSMC-UseNearestNeighbour = T
Particles-OctreePartNumNode        = 40
Particles-OctreePartNumNodeMin     = 28
```

For further information see *Pairing & Collision Modelling*.

Sampling

The outputs of the *Projectname_DSMCState_Timestamp.h5* (data in domain) and *Projectname_DSMCSurfState_Timestamp.h5* (surface data) files are based on sampling over several time steps. There are two different approaches that can not be used at the same time. The first method is based on specifying the sampling duration via `Part-TimeFracForSampling` as the fraction of the simulation end time (as defined by `TEnd`) between 0 and 1, where 0 means that no sampling occurs and 1 that sampling starts directly at the beginning of the simulation

$t_{\text{samp,start}} = T_{\text{end}} \cdot (1 - f_{\text{samp}})$

`Particles-NumberForDSMCOutputs` then indicates how many samples are written in this period. The data is not discarded after the respective output and the sampling is continued. In other words, the last output contains the data for the entire sampling period.

```
Part-TimeFracForSampling           = 0.5
Particles-NumberForDSMCOutputs     = 2
```

The second method is activated via `Part-WriteMacroValues = T`. In this approach, `Part-IterationForMacroVal` defines the number of iterations that are used for one sample. After the first sample has been written, the data is discarded and the next sampling process is started.

```
Part-WriteMacroValues              = T
Part-IterationForMacroVal          = 1250
```

For further information see *Particle Flow and Surface Sampling*.

Run the simulation

Finally, you can start the simulation using the Message Passing Interface (MPI) on multiple cores

```
mpirun -np 8 piclas parameter.ini DSMC.ini | tee std.out
```

To continue a simulation after a successful run, you have to provide the state file (`Projectname_State_Timestamp.h5`) you want to restart from

```
mpirun -np 8 piclas parameter.ini DSMC.ini Projectname_State_Timestamp.h5 | tee std.out
```

The restart also redistributes the computational load and can thus significantly reduce the time to solution. In the following, additional automatic load balancing during the run-time is described.

MPI and Load Balancing

When using several cores, piclas divides the computing load by distributing the computing cells to the various cores. If a particle leaves the boundaries of a core, it is necessary that the surrounding grid is also known. This region is defined by the `Particles-HaloEpsVelo` and the time step. In general, it can be said that this velocity should be greater than or equal to the maximum velocity of any particle in the simulation. This prevents a particle from completely flying through this halo region during a time step.

```
Particles-HaloEpsVelo = 8.0E+4
```

If the conditions change, it could make sense to redistribute the computing load. An example is the build-up of a bow shock during the simulation time: While all cells have the same particle density during initialization, an imbalance will develop after a short time. The cores with cells in the area of the bow shock have significantly more computational effort, since the particle density is significantly higher. As mentioned at the beginning, **piclas** redistributes the computing load each time it is started.

The parameter `Particles-MPIWeight` indicates whether the distribution should be oriented more towards a uniform distribution of the cells (values less than 1) or a uniform distribution of the particles (values greater than 1). There are options in piclas to automate this process by defining load balancing steps during a single program call. For this, load balancing must have been activated when compiling piclas (which is the default). To activate load balancing based on the number of particles, `DoLoadBalance = T` and `PartWeightLoadBalance = T` must be set. **piclas** then decides after each `Analyze_dt` whether a redistribution is required. This is done using the definable `Load DeviationThreshold`. Should the maximum relative deviation of the calculation load be greater than this value, a load balancing step is carried out. If `DoInitialAutoRestart = T` and `InitialAutoRestart-PartWeightLoadBalance = T` are set, a restart is carried out after the first `Analyze_dt` regardless of the calculated imbalance. To restrict the number of restarts, `LoadBalanceMaxSteps` limits the number of all load balancing steps to the given number.

```
! Load Balancing
Particles-MPIWeight                      = 1000
DoLoadBalance                            = T
PartWeightLoadBalance                    = T
DoInitialAutoRestart                     = T
InitialAutoRestart-PartWeightLoadBalance = T
LoadBalanceMaxSteps                      = 2
```

Information about the imbalance are shown in the *std.out* and the *ElemTimeStatistics.csv* file. The default load balancing scheme will exchange the required data internally, but there is also the possibility to perform the re-balancing step via HDF5, which will create a state file and restart from this file by activating

```
UseH5IOLoadBalance = T ! default is False
```

Visualization (post-processing)

Ensuring physical simulation results

After running a simulation, especially if done for the first time it is strongly recommended to ensure the quality of the results. For this purpose, the `Particles-DSMC-CalcQualityFactors = T` should be set, to enable the calculation of quality factors such as the maximum collision probability and the mean collision separation distance over the mean free path. All needed datasets can be found in the `*_DSMCState_*.h5` or the converted `*_visuDSMC_*.vtu`.

First of all, it should be ensured that a sufficient number of simulation particles were available for the averaging, which forms the basis of the shown data. The value `*_SimPartNum` indicates the average number of simulated particles in the respective cell. For a sufficient sampling size, it should be guaranteed that at least 10 particles are in each cell, however, this number is very case-specific. The value `DSMC_MCSoverMFP` is an other indicator for the quality of the particle discretization of the simulation area. A value above 1 indicates that the mean collision separation distance is greater than the mean free path, which is a signal for too few simulation particles. For 3D simulations it is sufficient to adjust the `Part-Species[$]-MacroParticleFactor` accordingly in **parameter.ini**. In 2D axisymmetric simulations, the associated scaling factors such as `Particles-RadialWeighting-PartScaleFactor` can also be optimized (see Section *2D/Axisymmetric Simulations*).

Similarly, the values `DSMC_MeanCollProb` and `DSMC_MaxCollProb` should be below 1 in order to avoid nonphysical values. While the former indicates the averaged collision probability per timestep, the latter stores the maximum collision probability. If this limit is not met, more collisions should have ocurred within a time step than possible. A refinement of the time step `ManualTimeStep` in **parameter.ini** is therefore necessary. If a variable timestep is also used in the simulation, there are further options (see Section *Variable Time Step*).

Table 1.10: Target value to ensure physical results and a connected input parameter

Property	Target	Connected Input Parameter
*_SimPartNum	10	Part-Species[$]-MacroParticleFactor
DSMC_MCSoverMFP	1	Part-Species[$]-MacroParticleFactor
DSMC_MaxCollProb	1	ManualTimeStep

Finally, the time step and particle discretization choice is a trade-off between accuracy and computational time. For further information see Section *Ensuring Physical Simulation Results*.

Visualizing flow field variables (DSMCState)

To visualize the data which represents the properties in the domain (e.g. temperatures, velocities, ...) the *DSMCState*-files are needed. They are converted using the program **piclas2vtk** into the VTK format suitable for **ParaView**, **VisIt** or many other visualisation tools. Run the command

```
./piclas2vtk dsmc_cone_DSMCState_000.00*
```

to generate the corresponding VTK files, which can then be loaded into your visualization tool. The resulting translational temperature and velocity in the domain are shown in Fig. 1.8. The visualized variables are `Total_TempTransMean`, which is mean translational temperature and the magnitude of the velocities `Total_VeloX`, `Total_VeloX`, `Total_VeloX` (which is automatically generated by ParaView). Since the data is stored on the original mesh (and not the internally refined octree mesh), the data initially looks as shown in the two upper halves. **ParaView**

offers the possibility to interpolate this data using the `CellDatatoPointData` filter. The data visualized in this way can be seen in the lower half of the image.

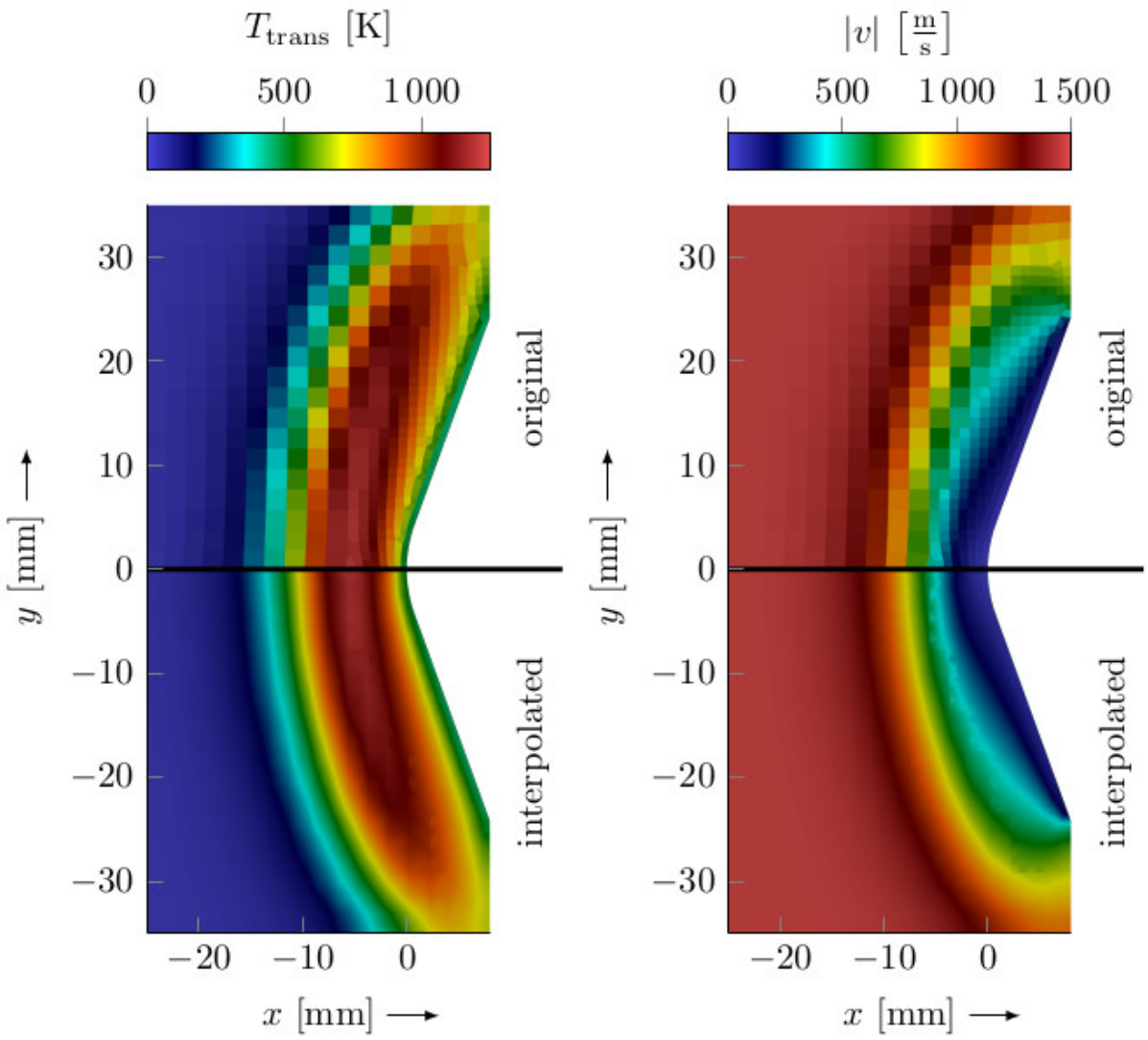

Fig. 1.8: Translational temperature and velocity in front of the 70° Cone, top: original data; bottom: interpolated data.

Visualizing surface variables (DSMCSurfState)

For postprocessing and visualization, the parameter `TimeStampLength = 13` is set in*parameter.ini* . This limits the output filename length. This can be needed, as e.g. Paraview may sort the files incorrectly and display a faulty time solution. To visualize the data which represents the properties at closed boundaries (e.g. heat flux, force per area, etc. the *DSMCSurfState*-files are needed. They are converted using the program **piclas2vtk** into the VTK format suitable for **ParaView**, **VisIt** or many other visualization tools. Run the command

```
./piclas2vtk dsmc_cone_DSMCSurfState_000.00*
```

to generate the corresponding VTK files, which can then be loaded into your visualization tool. A comparison between experimental data by [72] and the simulation data stored in `dsmc_cone_visuSurf_000.002000000.vtu` is shown at Fig. 1.9. Further information about this comparison can be found at [74].

Fig. 1.9: Experimental heat flux data compared with simulation results from PICLas.

1.7.4 Hypersonic Flow around the 70° Cone (DSMC) - 3D Mesh with Gmsh

With the validation case of a 70° blunted cone already used in the previous tutorial (*Hypersonic Flow around the 70° Cone (DSMC) - 2D Mesh*), the 3D mesh generation using Gmsh is presented in greater detail in this tutorial. Before starting, copy the `dsmc-cone-gmsh` directory from the tutorial folder in the top level directory to a separate location

```
cp -r $PICLAS_PATH/tutorials/dsmc-cone-3D .
cd dsmc-cone-3D
```

The general information needed to setup a DSMC simulation is given in the previous tutorials *Adiabatic Box/Reservoir (DSMC, Relaxation/Chemistry)* and *Hypersonic Flow around the 70° Cone (DSMC) - 2D Mesh*. The following focuses on the mesh generation with Gmsh and case-specific differences for the DSMC simulation.

Mesh generation with Gmsh

First, create a new file in gmsh: `70DegCone_3D.geo`. In general, the mesh can be generated using the GUI or by using the `.geo` script environment. In the GUI, the script can be edited via `Edit script` and loaded with `Reload script`. This tutorial focuses on the scripting approach.

After opening the `.geo` script file, select the OpenCASCADE CAD kernel and open the provided `70DegCone_3D_model.step` file with the following commands:

```
SetFactory("OpenCASCADE");
v() = ShapeFromFile("70degCone_3D_model.step");
```

The simulation domain is created next by adding a cylindrical section and subtracting the volume of the cone.

```
Cylinder(2) = {-50, 0, 0, 100, 0, 0, 50, Pi/6};
BooleanDifference(3) = { Volume{2}; Delete; }{ Volume{1}; Delete; };
```

Physical groups are used to define the boundary conditions at all surfaces:

```
Physical Surface("IN", 29) = {4, 1};
Physical Surface("SYM", 30) = {3, 5};
Physical Surface("OUT", 31) = {2};
Physical Surface("WALL", 32) = {7, 8, 9, 10, 11, 6};
```

The mesh options can be set with the following commands:

```
Mesh.MeshSizeMin = 1;
Mesh.MeshSizeMax = 10;
Field[1] = MathEval;
Field[1].F = "0.2";
Field[2] = Restrict;
Field[2].SurfacesList = {7, 8, 9};
Background Field = 2;
Mesh.Algorithm = 1;
Mesh.Algorithm3D = 7;
Mesh.SubdivisionAlgorithm = 2;
Mesh.OptimizeNetgen = 1;
```

The commands `Mesh.MeshSizeMin` and `Mesh.MeshSizeMax` define the minimum and maximum mesh element sizes. With the prescribed `Field` options, the size of the mesh can be specified using an explicit mathematical function using `MathEval` and restriced to specific surfaces with `Restrict`. In this tutorial, a mesh refinement at the frontal wall of the cone is enabled with this. `Background Field = 2` sets `Field[2]` as background field. Different meshing algorithms for creating the 2D and 3D meshes can be chosen within Gmsh. The command `Mesh.SubdisionAlgorithm = 2` enables the generation of a fully hexahedral mesh by subdivision of cells. `Mesh.OptimizeNetgen` improves the mesh quality additionally.

Next, the 3D mesh is created:

```
Mesh 3;
```

The following commands are required to save all elements even if they are not part of a physical group and to use the ASCII format, before saving the mesh as `70degCone_3D.msh`:

```
Mesh.SaveAll = 1;
Mesh.Binary = 0;
Mesh.MshFileVersion = 4.1;
Save "70degCone_3D.msh";
```

The mesh file in the file format `.h5` used by **piclas** has to be converted using HOPR by supplying an input file `hopr.ini` using the corresponding mode:

```
Mode = 5
```

As another possibility, the `SplitToHex` option can be enabled in the `hopr.ini` file instead of using the `SubdivionAlgorithm` command in Gmsh. The expected result for the 3D mesh is shown in Fig. 1.10.

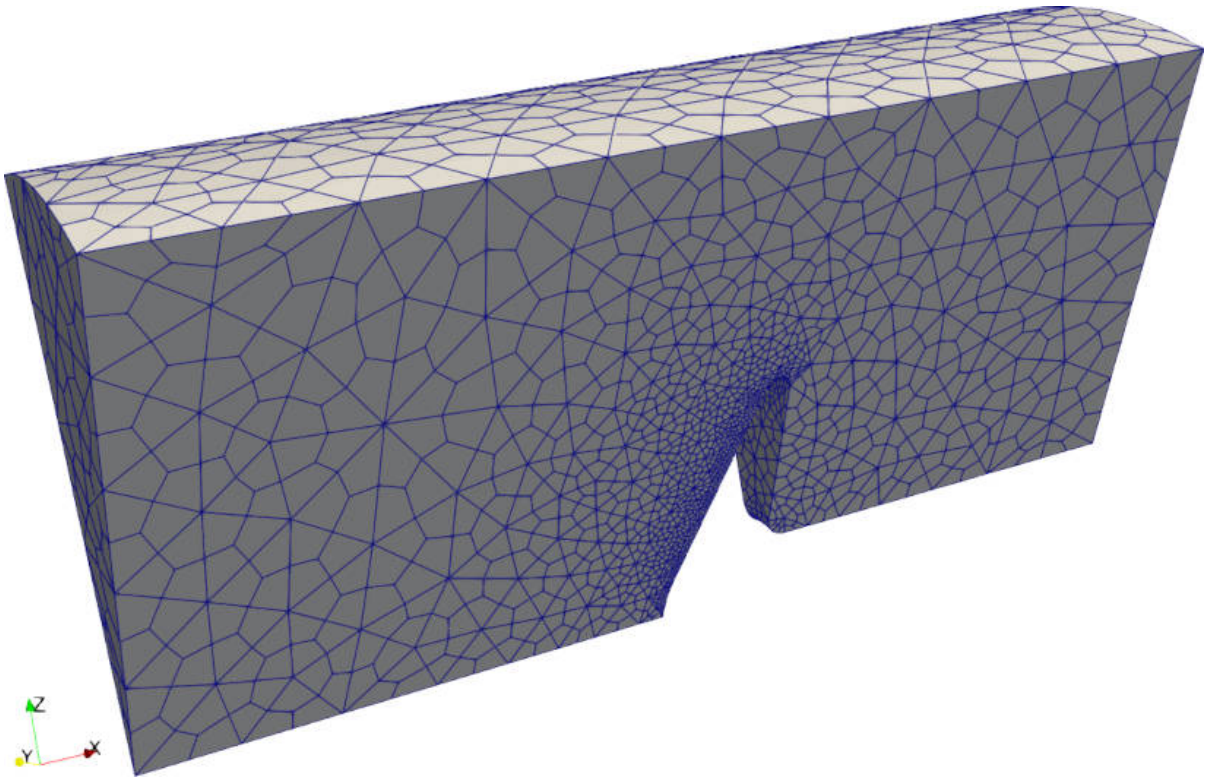

Fig. 1.10: 3D mesh of the 70° cone.

Flow simulation with DSMC

For the general information on the setup of the DSMC simulation, please see the previous tutorial *Hypersonic Flow around the 70° Cone (DSMC) - 2D Mesh*. In this tutorial, only the changes in the `parameter.ini` file for the 3D simulation compared to the 2D simulation are explained further.

First, the mesh file name is adapted. The number of boundaries is reduced from five to four, as only one symmetrical boundary is used in this 3D simulation. The maximum particle number per processor is increased due to the changed simulation domain. Additionally, the octree is adapted to appropriate values for a 3D simulation.

```
MeshFile                        = 70degCone_3D_mesh.h5
Part-nBounds                    = 4
Part-maxParticleNumber          = 1500000
Particles-OctreePartNumNode     = 80
Particles-OctreePartNumNodeMin  = 60
```

Compared to the `parameter.ini` for the 2D simulation, the symmetrical boundaries, the commands for the 2D axisymmetric simulation and the radial weighting are deleted:

```
Part-Boundary4-SourceName                   = SYMAXIS
Part-Boundary4-Condition                    = symmetric_axis
Part-Boundary5-SourceName                   = ROTSYM
Part-Boundary5-Condition                    = symmetric
Particles-Symmetry-Order                    = 2
Particles-Symmetry2DAxisymmetric            = T
Particles-RadialWeighting                   = T
Particles-RadialWeighting-PartScaleFactor   = 60
Particles-RadialWeighting-CloneMode         = 2
Particles-RadialWeighting-CloneDelay        = 5
```

Instead, a new symmetrical boundary is added:

```
Part-Boundary4-SourceName  = SYM
Part-Boundary4-Condition   = symmetric
```

An exemplary simulation result using the 3D mesh generated with Gmsh is shown in Fig. 1.11.

1.8 Cluster Guidelines

PICLas has been tested on numerous different Linux- and CPU-based clusters and is continuously been developed to run efficiently on high-performance systems.

1.8.1 Simulating at HLRS

Unfortunately, the GitHub and GitLab servers are not available on machines at the HLRS, such as the Hawk, due to restricted internet access. The workaround is to use ssh tunneling and proxy jump or remote forwarding to access the repositories.

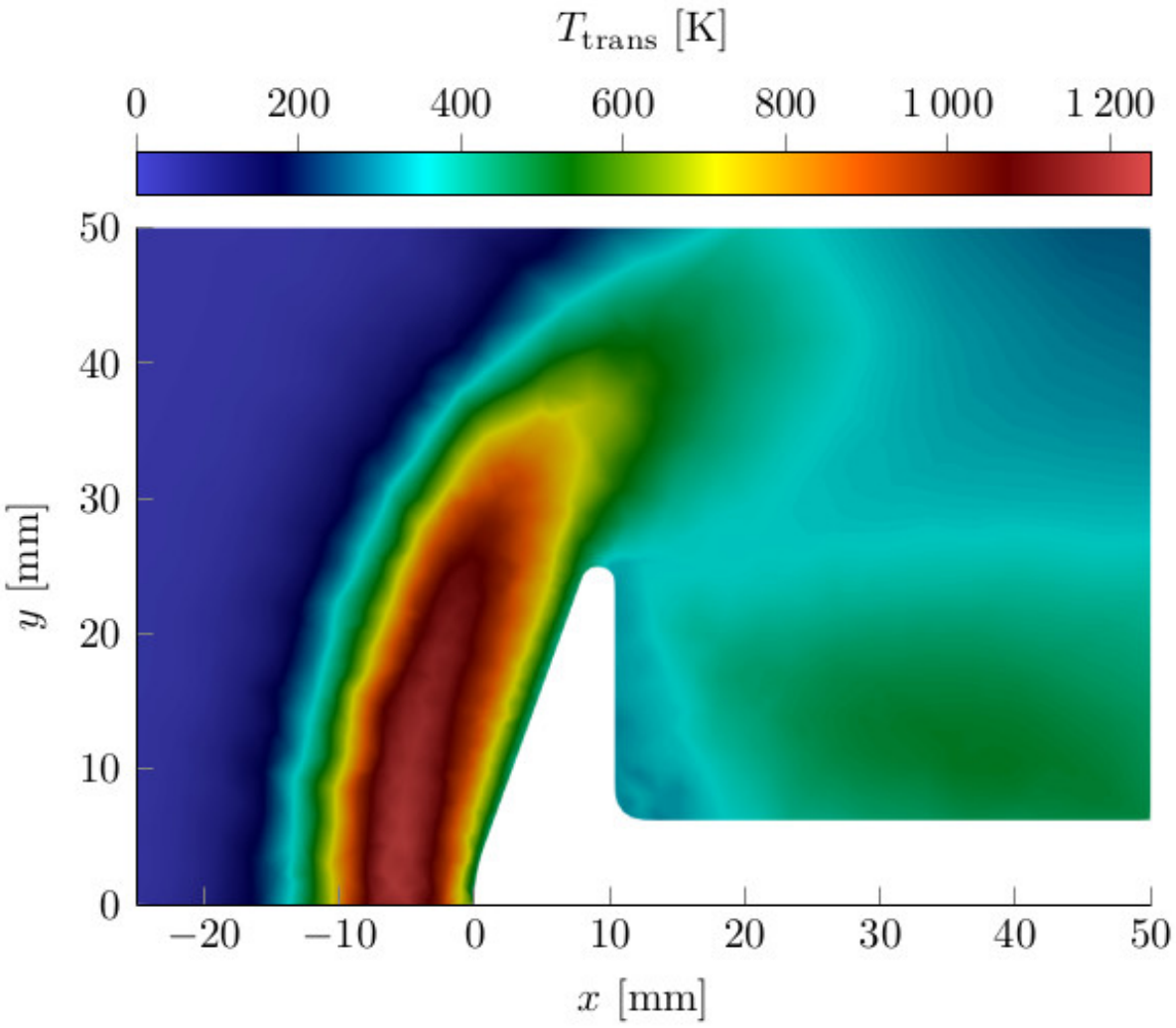

Fig. 1.11: Translational temperature around the 70° cone.

Cloning with the SSH protocol

Two methods for checking out the code are described in the following.

Method 1 (Proxy Jump)

To clone a repository from, e.g., github.com on a HLRS system, the ssh proxy jump must first be set up. Simply connect to the system via ssh and add the following lines to the ssh configuration file under `~/.ssh/config` on the cluster

```
Host github.com
   HostName   github.com
   ProxyJump  RemoteHost
```

where the *RemoteHost* has internet access and can be accessed via ssh from the HLRS system. It is also defined

```
Host RemoteHost
   User         username
   HostName     hostname.something.com
   ForwardX11   yes
```

Then simply clone the repository via

```
git clone git@github.com:mygroup/myproject.git
```

and all the ssh traffic via `github.com` is automatically re-routed over the *RemoteHost*.

Method 2 (Remote Forwarding)

You can use the SSH protocol to clone the repository. You have to connect to the cluster with the `RemoteForward` option

```
ssh -R 7777:github.com:22 username@hazelhen.hww.de
```

To avoid using the above command every time, you can add the following to your `.ssh/config` file:

```
host hlrs
   hostname        hazelhen.hww.de
   user            username
   RemoteForward   7777 github.com:22
```

and login with `ssh hlrs`. Now you can clone the repository when logged onto the cluster by

```
git clone ssh://git@localhost:7777/piclas/piclas.git
```

Note that if you experience problems when connecting, e.g., when the warning

```
Warning: remote port forwarding failed for listen port 7777
```

is issued, you have to choose a different port, e.g., 1827 (or any other 4-digit number) and re-connect. If the code has already been cloned using the original port, the number of the port must be changed in `./git/config` to the new number for git fetch/pull operations, which would then look like

```
[remote "origin"]
  url = ssh://git@localhost:1827/piclas/piclas.git
  fetch = +refs/heads/*:refs/remotes/origin/*
```

Compiling and executing PICLas

For building on the hazelhen cluster, certain modules have to be loaded and included in the .bashrc or .profile:

```
module unload PrgEnv-cray
module load PrgEnv-intel
module load cray-hdf5-parallel
```

An example submit script for the test queue is then

```
#!/bin/bash
#PBS -N Testcase
#PBS -M email@university.de
#PBS -m abe
#PBS -l nodes=4:ppn=24
#PBS -l walltime=00:24:59
#PBS -q test

# number of cores per node
nodecores=24

# switch to submit dir
cd $PBS_O_WORKDIR

# get number of total cores
ncores=$(cat $PBS_NODEFILE | wc -l)

module unload craype-hugepages16M

# restart
aprun -n $ncores -N $nodecores -j 1 ./piclas parameter.ini DSMC.ini restart.h5 1>log 2>
↪log.err
```

More information on using the queue system can be found in the HLRS wiki.

Section last updated: 27.03.2019

1.9 Appendix

1.9.1 Tested compiler combinations

The following list summarizes all **tested combinations** of the required libraries (HDF5, OpenMPI, CMake etc.)

Dev	PICLas Version	System	Compiler	HDF5	MPI	CMake
SC	2.3.0 (Nov 2021)	PC	gcc11.2.0	1.12.1	openmpi-4.1.1	3.21.3
SC	2.2.0 (Nov 2021)	PC	gcc10.1.0	1.10.5	openmpi-4.0.2	3.17.0
AM	2.1.0 (Nov 2021)	PC	gcc9.3.0	1.10.6	openmpi-3.1.6	3.17.0
SC	2.0.0 (Nov 2021)	boltzhawk	gcc9.3.0	1.10.5	openmpi-3.1.6	3.17.0
SC	2.0.0 (Nov 2021)	boltzreggie	gcc9.2.0	1.10.5	openmpi-4.0.2	3.15.0
SC	2.0.0 (Nov 2021)	boltzplatz	gcc9.2.0	1.10.5	openmpi-3.1.6	3.17.0
SC	2.0.0 (Nov 2021)	hawk	gcc9.2.0	1.10.5	mpt2.23	3.16.4
SC	2.0.0 (Nov 2021)	fh1	intel18.1	1.10	impi2018	3.17
SC	2.0.0 (Nov 2021)	fh2	intel19.1	1.10	impi2019	3.17
PN	1.4.0 (Nov 19)	boltzplatz	gnu7.4.0	1.10.5	openmpi-3.1.3	3.15.3-d
SC	1.4.0 (Nov 19)	boltzreggie	gnu9.2.0	1.10.5	openmpi-4.0.2	3.15.3-d

Combinations that can cause problems are listed in the following table

Dev	PICLas Version	System	Compiler	HDF	MPI	CMak	Notes
SC	1.4.0 (Nov 19)	boltzreggie	gnu9.2	1.10..	openmpi-4.0.1	3.15.3d	Does not work for more than 3 processors probably due to a problem with the OpenMPI version

1.9.2 Unified Species Database (USD)

This section documents the initial creation process for the database, but since it is updated all the time it is recommended that modifications and maintenance are performed using the scripts in the *tools* folder: `piclas/tools/species_database/`. These scripts are meant to provide a controlled and uniform approach to database management. All files to create the original database were moved to `piclas/tools/archive/`.

A tool to create a database containing cross-section, electronic states, Arrhenius rates, and species data can be found in the *tools* folder: `piclas/tools/species_database/`. The Python script (python3.7) `create_species_database.py` creates a new database or expands an existing one combining all necessary parameters (formerly read-in through ini files). The script uses the `numpy`, `h5py`, `argparse`,`datetime`, `cmath`, and `matplotlib.rcsetup` packages. To create the species database run the command:

```
python3.7 create_species_database.py
```

If electronic states or cross-section data should be added to the species database, an electronic states `Electronic-State-Database.h5` and a cross-section database `XSec-Database.h5` need to be built before.

If nothing additionally is specified, the following filenames are called: `DSMC.ini` for the parameter, gas-phase reaction input and `Species_Database.h5` for the final output. For custom file names and for the inclusion of electronic and cross-section data, the following options can be added:

```
python3 create_species_database.py --parameter parameter-filename --electronic␣
↪electronic_statefile --crosssection crosssection_statefile --output output_filename --
↪reference reference-name
```

or

```
python3 create_species_database.py -p parameter-filename -e electronic_statefile -c␣
↪crosssection_statefile -o output_filename -r reference-name
```

The data is grouped in the output file, as shown in the following example:

```
Cross-Sections (group)
    H2-H2Ion1 (dataset)
Reaction (group)
    CH3_CH2+H (dataset)
    Chemistry model (attributes)
    Arrhenius parameters (attributes)
    O2+M_O+O+M (dataset)
    Chemistry model (attributes)
    Arrhenius parameters (attributes)
    Fe_FeIon1+electron (dataset)
    Chemistry model (attributes)
    Arrhenius parameters (attributes)
Species (group)
    H2 (group)
        Electronic levels (dataset)
        Species parameters (attributes)
    H2Ion1 (group)
        Electronic levels (dataset)
        Species parameters (attributes)
    electron (group)
        Electronic levels (dataset)
        Species parameters (attributes)
```

For cross-sections, reactions and species data, the former `DSMC.ini` files are used to create the database. However, every species must be defined to create the database and to run simulations.

```
Part-Species1-SpeciesName=CO2
```

The name of the reaction is optional, if none is given, the reaction name is created automatically from the reactants and products, following the naming convention defined below:

```
Reac1+Reac2_Prod1+Prod2+Prod3
Reac1+Reac2_Prod1+Prod2+Prod3#2
```

Non-reacting partners can be given as A (atoms or similar) or M (molecules or similar) in the reaction name. If a name is defined in the input file, the programm automatically renames it according to the set convention. If multiple sets of parameters or multiple models exist for the same reaction, a counter variable f.e. '#5' is added at the end of the reaction name.

For reactions a chemistry model is defined in all cases. The name of the given parameter-file is automatically taken as the model name. To have a clear distinction, the following naming convention should be used for the parameter-filename and thus the chemistry model for the reactions:

```
PlanetAtmosphere_XSpec_XReac_Source (Titan_18Spec_30Reac_Gokcen2007)
TestCase_XSpec_XReac_Source (CO2_6Spec_10Reac_Johnston2014)
```

In addition to creating a new database, the same script can be used to extend an existing version. For this, only the name of the existing database needs to be defined in the call to create_species_database.py. The function automatically tests if the provided data is already included and adds them if not.

After creating the database the script `extend_reactions.py` in `piclas/tools/archive/` is used to allow one reaction to be part of more than one chemistry model. In this script there are some hard coded changes to ensure a consistent documentation, so no changes by hand had to be made, e.g. changing the reaction model to 'QK' for these reactions 'O2+electron_O2Ion1+electron+electron#1','O+electron_OIon1+electron+electron#1' and 'CO+electron_COIon1+electron+electron#1'.

Electronic database

This function was also moved to the *tools* folder: `piclas/tools/species_database/`. If a new atom species is added the electronic excitation states are added to the database with the data from the NIST database (https://physics.nist.gov/cgi-bin/ASD/energy1.pl). The folder `piclas/tools/electronic_database/` was removed to enable a uniform handling of electronic excitation states, but the section is kept for clarity.

A tool to create a database containing electronic excitation states can be found in the *tools* folder: `piclas/tools/electronic_database/`. The Python script (python3.7) `create_electronic_database_atoms.py` can be used to populate a PICLas-compatible cross-section database, using the `pandas`, `h5py`, `io`, `re`, `datetime` and `requests` packages. It can be excuted with

```
python3.7 `create_electronic_database_atoms.py`
```

The script gets the data from the NIST database (https://physics.nist.gov/cgi-bin/ASD/energy1.pl) and stores it in an h5-database. Additional species can be added by adapting the `species-list` parameter.

Reactions

This function was also moved to the *tools* folder: `piclas/tools/species_database/`. If a new reaction is added the necessary data is read in from user inputs or from the name of the reaction.

Reactions can be defined in a separate group as shown above, named by the product species, and added manually using h5py or HDF View (Make sure to re-open the file as `Read/Write` to be able to modify and create the dataset.). The ionization process from the LXCat database is not added as a reaction, however, the cross-section can be used. An additional source for cross-sectional data for reactions is the NIFS database.

This user guide is organized to both guide the first steps as well as provide a complete overview of the simulation code's features from a user and a developer point of view.

- Chapter *Installation* contains step by step instructions from obtaining the source code up to running a first simulation and visualizing the simulation results. In addition, it provides an overview of the whole simulation framework and the currently implemented features.
- Chapter *Mesh Generation* describes the preprocessing step of creating mesh files via the in-house tool HOPR that also handles mesh formats created with external mesh generators
- Chapter *Workflow* outlines the workflow and the visualization of results produced with **PICLas**.
- Chapter *Features & Models* shall serve as a reference for the models and features implemented in **PICLas**.
- Chapter *Visualization & Output* presents the options and parameters for the output of particle data, field and flow variables.
- Chapter *Tools* lists tools within the **PICLas** repository, including the post-processing tools.
- Simulation tutorials are contained in Chapter *Tutorials*.
- Cluster-specific user guidelines are given in Chapter *Cluster Guidelines*.

CHAPTER

TWO

DEVELOPER GUIDE

The developer guide is intended to be for people who come in more close contact with PICLas, i.e., code developers and performance analysts as well as people who are tasked with working or extending the documentation of PICLas.

2.1 GitLab Workflow

Code development is performed on the GitLab platform, with the protected `master` and `master.dev` branches. The actual development is performed on feature branches, which can be merged to `master.dev` following a merge request and the completion of a merge request checklist. After a successful pass of the nightly and weekly regression test, the `master.dev` can be merged into the `master`. A merge of the `master.dev` to the `master` should be associated with a release tag, where the changes to previous version are summarized.

In the following the envisioned development process using issues and milestones, the release & deploy procedure as well as other developer relevant issues are discussed.

2.1.1 Issues & Milestones

Issues are created for bugs, improvements, features, regression testing and documentation. The issues should be named with a few keywords. Try to avoid describing the complete issue already in the title. The issue can be assigned to a certain milestone (if appropriate).

Milestones are created based on planned releases (e.g. Release 1.2.1) or as a grouping of multiple related issues (e.g. Documentation Version 1, Clean-up Emission Routines). A deadline can be given if applicable. The milestone should contain a short summary of the work performed (bullet-points) as its contents will be added to the description of the releases. Generally, merge requests should be associated with a milestone containing a release tag, while issues should be associated with the grouping milestones.

As soon as a developer wants to start working on an issue, she/he shall assign himself to the issue and a branch and merge request denoted as work in progress (`WIP: ...`) should be created to allow others to contribute and track the progress. For this purpose, it should be created directly from the web interface within the issue (`Create merge request`). This creates a branch, naming it automatically, leading with the issue number (e.g. `60-fix-boundary-condition`) and associates the branch and merge request to the issue (visible in the web interface below the description). To start working on the issue, the branch can be checked out as usually.

Ideally, issues should be created for every code development for documentation purposes. Branches without an issue should be avoided to reduce the number of orphaned/stale branches. However, if branches are created outside of the issue context, they should be named with a prefix indicating the purpose of the branch, according to the existing labels in GitLab. Examples are given below:

```
feature.chemistry.polyatomic
improvement.tracking.curved
bug.compiler.warnings
reggie.chemistry.reservoir
documentation.pic.maxwell
```

Progress tracking, documentation and collaboration on the online platform can be enabled through creating a merge request with the WIP prefix for this branch instead of an issue. An issues created afterwards cannot be associated with an already created branch, without renaming the branch to include the issue number at the beginning. However, this should be avoided.

2.1.2 Merge Request

Merge requests that are not WIP are discussed every Monday by the developer group to be considered for a merge. The following checklist has to be completed before a merge request should be approved. For bugs only the first points have to be considered, while for features and improvements the complete list has to be completed. The **Feature** merge request template considers the following bullet points

```
## Related Issue

Closes #number

## Merge Request Checklist

* [ ] Style Guide
* [ ] Maximum of 10 compile warnings via *./tools/test_max_warnings.sh*. How many␣
↪warning were found?
* [ ] No large files via *./tools/test_max_file_size.sh*. What is the largest file?
* [ ] Descriptions for new/changed routines
  * [ ] Short header description (do not just spell out the name of the subroutine,␣
↪units for important variables if applicable)
  * [ ] Workflow
    * [ ] Short summary in the header
    * [ ] Inside the routine at the appropriate positions
* [ ] Reggie
  * [ ] Add small test setup
  * [ ] Add entry in REGGIE.md table
  * [ ] Check automatic restart functionality of reggie example via Load Balance (checks␣
↪correct allocation and deallocation for the test case)
* [ ] New feature description in appropriate documentation (user/developer guide)
* [ ] Check that no large files were added to the repository
```

For this purpose, the developer can select the respective template for his merge request (**Bug**: only first two to-do's or **Feature**: all to-do's, Improvements can utilize either depending on the nature of the improvement). The appropriate checklist will then be displayed as the merge request description. Merge requests generated automatically through the Issues interface have already `Closes #55` as a description. When editing the merge request, the description gets overwritten by the template. Thus, the issue number has to be added manually after the template is chosen. The templates for merge requests are stored in *./.gitlab/merge_request_templates/*.

2.1.3 Release and deploy

A new release version of PICLas is created from the **master** repository, which requires a merge of the current **master.dev** branch into the **master** branch. The corresponding merge request should be associated with a release milestone (e.g. *Release 1.X.X* within which the merge request is referenced, e.g., "See merge request !283"). Within this specific merge request, the template `Release` is chosen, which contains the to-do list as well as the template for the release notes as given below. After the successful completion of all to-do's and regression checks (check-in, nightly, weekly), the **master.dev** branch can be merged into the **master** branch.

Release Tag

A new release tag can be created through the web interface (Repository -> Tags -> New tag) and as the `Tag name`, the new version number is used, e.g.,

```
v1.X.X
```

The tag is then created from the **master** branch repository and the `Message` is left empty. The release notes, which were used within the corresponding milestone, shall be given in the following format

```
## Release 1.X.X

### Documentation

* Added section about particle emission

### Reggie

* Added a regression test of the chemistry routine

### Features

* Skipping field update for the HDG solver for user-defined number of iterations

### Improvements

* Speed-up by skipping/cycle over neutral particles in deposition

### Fixes

* Treatment of non-linear polyatomic molecules during analyze and wall interaction
```

Headlines without changes/additions within a release can be omitted.

Collaborative Numerics Group

The collaborative numerics group for PICLas is located at *https://gitlab.com/collaborative-numerics-group/piclas*. There are two possible ways of sharing code with other people and are explained in the following.

Single master branch repository

Method 1 involves a single master branch that is updated from the internal PICLas repository, similar to the repository of PICLas at *github.com*.

The master branch of development group can be merged after the successful regression check with the master of the collaborative group. For this purpose, the collaborative repository can be added as a remote (this step has only to be performed once)

```
git remote add remote_name git@gitlab.com:collaborative-numerics-group/piclas/piclas.git
```

First, make sure to have the most recent version of the master branch (of the development repository)

```
git checkout master && git pull
```

Now you can checkout the most recent version of the master branch of the collaborative-numerics-group and create a local branch with that version (performing only a simple checkout will create a detached HEAD state)

```
git fetch
git checkout -b branch_name remote_name/master
```

The master branch of the development repository can now be merged into the newly created branch

```
git merge origin/master
```

Finally, the changes can be pushed from the local branch *branch_name* to the master of collaborative-numerics-group

```
git push remote_name master
```

If a tag has also been created, it should be pushed separately.

```
git push remote_name tag_name
```

Afterwards, the local branch *branch_name* can either be deleted or utilized for future merges

```
git branch -d branch_name
```

Pushing each branch to a separate repository

Method 2 involves pushing each development branch from the internal PICLas repository to a separate repository in the PICLas CRG separately.

Extract solely a single branch by cloning only *myfeaturebranch* via

```
git clone -b myfeaturebranch --single-branch git@gitlab.com:piclas/piclas.git piclas_new_
↪CRG ;
```

Navigate to the newly created directory

```
cd piclas_new_CRG
```

Push this repository that only contains one branch to the CRG via

```
git push --mirror git@gitlab.com:collaborative-numerics-group/piclas/piclas.
↪myfeaturebranch.git
```

which created a new repository that only contains the desired feature branch. When sharing this new repository with new people, simply add them as members of this new repository (not the complete CRG!).

GitHub

Finally, the release tag can be deployed to GitHub. This can be achieved by running the `Deploy` script in the CI/CD -> Schedules web interface. At the moment, the respective tag and the release have to be created manually on GitHub through the web interface with **piclas-framework** account. The releases are accessed through Releases and a new release (including the tag) can be created with `Draft a new release`. The tag version should be set as before (`v1.X.X`) and the release title accordingly (`Release 1.X.X`). The release notes can be copied from the GitLab release while omitting the `## Release 1.X.X` headline as it was given with the release title before.

2.2 Documentation

2.2.1 Building documentation

The user and developer guides are built automatically via Read The Docs. When changing the guides, build the html and pdf files locally before committing changes to the repository.

Prerequisites

The following shows how to create the html and pdf files for the user/developer guide. First, install the required prerequisites. Install *python3* and make sure that pip (third-party Python packages) is installed. If you have an old version of, e.g., Ubuntu visit this website.

```
sudo apt install python3-pip
```

Navigate to the documentation folder from the PICLas top level directory

```
cd docs/documentation
```

Run pip to install the required extensions and packages for compiling the user guide (only once)

```
python3 -m pip install --exists-action=w --no-cache-dir -r requirements.txt
```

Make sure that *latexmk* is installed on the system for compiling the PDF version of the user guide. For Ubuntu, follow this link for installation.

```
sudo apt-get install latexmk
```

HTML Version

Compile the html version of the user guide via

```
python3 -m sphinx -T -E -b html -d _build/doctrees -D language=en . _build/html
```

Check that no errors occur during compilation and then navigate to the created html files

```
cd _build/html
```

Open index.html to see if everything has worked out correctly (e.g. with your favourite browser). Note that you can simply run the script *buildHTML.sh* in the *documentation* directory for this task.

PDF Version

Next, create the pdf output.

```
python3 -m sphinx -b latex -D language=en -d _build/doctrees . _build/latex
```

and switch into the output directory

```
cd _build/latex
```

Finally, compile the pdf file

```
latexmk -r latexmkrc -pdf -f -dvi- -ps- -jobname=piclas -interaction=nonstopmode
```

and check if the pdf exists

```
ls _build/latex/piclas.pdf
```

Note that you can simply run the script *buildPDF.sh* in the *documentation* directory for this task.

2.2.2 Writing documentation

Figures

Graphics are supported as long as their size is not above 1 MB, recommended size is below 100 KB. The conversion of PDF plots/graphics produced with pgfplots/tikz can be performed via terminal using the `libvips` package available for most distributions

```
sudo apt install libvips-dev
vips copy example.pdf[dpi=150] example.jpg[Q=90,strip]
```

Modify `dpi=150` to scale the PDF and `Q=90` (between 0 and 100) to change the quality of the JPEG. Vector graphics might give better quality using SVG as the end format by converting from PDF

```
pdf2svg example.pdf example.svg
```

2.3 Style Guide

- Why do we need a style guide?
  - It creates a unified appearance and coding structure
  - It makes the code more understandable and therefore important information is understood more easily
  - It forces the developers to think more actively about their work
- General rules
  - Coding language: English
  - A maximum of 132 characters are allowed per line (incl. Comments)
  - Indentation: 2 spaces (no tabs!)
  - Line breaks in comments -> the following line must be indented appropriately
  - Comments of modules and input-/output variables: Doxygen style
  - Comments of preprocessor directives in C-Style

2.3.1 Header of Functions and Subroutines

Function calls must always supply the variable name of optional arguments. Always use `USE` statements with `ONLY`

```
USE MODULE, ONLY: ...
```

this accounts for variables and function/subroutines. An exception are the initialization and finalization routines.

```
!===================================================================
!> \brief Fills the solution array U with a initial solution.
!>
!> Fills the solution array U with a initial solution provided by the ExactFunc␣
↪subroutine through interpolation. Function is
!> specified with the IniExactFunc paramter.
!===================================================================
SUBROUTINE FillIni(NLoc,xGP,U)
!-------------------------------------------------------------------
! MODULES
USE MOD_PreProc
USE MOD_Equation_Vars ,ONLY: IniExactFunc
USE MOD_Exactfunc     ,ONLY: ExactFunc
USE MOD_Mesh_Vars     ,ONLY: nElems
IMPLICIT NONE
!-------------------------------------------------------------------
! INPUT/OUTPUT VARIABLES
INTEGER,INTENT(IN)              :: NLoc                                      !<␣
↪Polynomial degree of solution
REAL,INTENT(IN)                 :: xGP(3,    0:NLoc,0:NLoc,0:NLoc,nElems)    !<␣
↪Coordinates of Gauss-points
REAL,INTENT(OUT)                :: U(1:PP_nVar,0:NLoc,0:NLoc,0:NLoc,nElems)  !< Solution␣
↪array
!-------------------------------------------------------------------
! LOCAL VARIABLES
INTEGER                         :: i,j,k,iElem
!===================================================================

! Evaluate the initial solution at the nodes and fill the solution vector U.
DO iElem=1,nElems
  DO k=0,NLoc; DO j=0,NLoc; DO i=0,NLoc
    CALL ExactFunc(IniExactFunc,0.,xGP(1:3,i,j,k,iElem),U(:,i,j,k,iElem))
  END DO; END DO; END DO
END DO
END SUBROUTINE FillIni
```

The separators `!====` and `!----` are exactly 132 characters long (here they have been shortened for visualization purposes).

2.3.2 Variables

- Preprocessor variables: PP_$var

```
PP_nVar
```

  Note that

  - USE MOD_Preproc cannot be used with ONLY because the pro-processor flags sometimes result in constants and not variables
  - PP_N and other pre-processor variables that may be constants cannot be assigned in ASSOCIATE constructs

- Counters: the counting variable (lower case) + description (the first character is capital case)

```
DO iVar=1,PP_nVar
```

- Variables generally begin with a capital letter (composite words also)

```
ActualElem
```

- Dimension allocations must be specified by giving both a lower and an upper boundary

```
ALLOCATE(U(1:PP_nVar,0:NLoc,0:NLoc,0:NLoc,nElems))
```

- When using single characters: small at the beginning when using composite words otherwise in capital letters. Both is possible when purely single characters are used. Exceptions are allowed in special cases, but they are not recommended.

```
hTilde, TildeH, (Elem%U)
```

2.3.3 Functions and Control Structures

- User-defined functions and subroutines should carry meaning in their name. If their name is composed of multiple words, they are to be fused together without underscores (_) and the first letter of each words should be capital.

```
GetParticleWeight(), isChargedParticle()
```

  An exception to this rule is the usage of underscores (_) for shared memory arrays, where _Shared indicates that the property is available to all processors on the same shared-memory domain (generally a node). Furthermore, the words Get, get, Is, is, Do, do indicate the intention of the function/subroutine in supplying or acquiring specific properties or values. User-defined functions and subroutines should not, in general, be named in all-capital letters.

- FORTRAN intrinsics generally in capital letters

```
ALLOCATE(), DO, MAX(), SQRT(), INT(), etc.
```

- END-X is to be separated by a space

```
END DO, END IF, END SUBROUTINE
```

- For loops and IF statements etc. comments are to be inserted at the end (and in-between, e.g. when ELSE IF is used)

```
DO iVar=1,PP_nVar
  IF (a.EQ.b) THEN
...
  ELSE ! a.NE.b
...
  END IF ! a.EQ.b
...
END DO ! PP_nVar
```

2.3.4 Workflow Description

Additionally to the header description, a short workflow table of contents at the beginning of the subroutine or function is required for longer subroutines in which multiple tasks are completed. Example:

```
! ----------------------------------------------------------------------------------
! MAIN STEPS        []=FV only
! ----------------------------------------------------------------------------------
! 1.  Filter solution vector
! 2.  Convert volume solution to primitive
! 3.  Prolong to face (fill U_master/slave)
! 4.  ConsToPrim of face data (U_master/slave)
![5.] Second order reconstruction for FV
! 6.  Lifting
! 7.  Volume integral (DG only)
![8.] FV volume integral
! 9.  IF EDDYVISCOSITY: Prolong muSGS to face and send from slave to master
! 10. Fill flux (Riemann solver) + surface integral
! 11. Ut = -Ut
! 12. Sponge and source terms
! 13. Perform overintegration and apply Jacobian
! ----------------------------------------------------------------------------------
```

Furthermore, the steps are required to be found at the appropriate position within the code. It is not allowed to just incorporate the corresponding number of the step within the code.

```
! (0. Nullify arrays)
! NOTE: UT and U are nullified in DGInit, and Ut is set directly

! 1. Filter the solution vector if applicable, filter_pointer points to cut-off
IF(FilterType.GT.0) CALL Filter_Pointer(U,FilterMat)

! 2. Convert Volume solution to primitive
CALL ConsToPrim(PP_N,UPrim,U)

! X. Update mortar operators and neighbour connectivity for the sliding mesh
CALL PrepareSM()

! 3. Prolong the solution to the face integration points for flux computation
! ----------------------------------------------------------------
! General idea: The slave sends its surface data to the master
! where the flux is computed and sent back to the slaves.
! Steps:
```

(continues on next page)

(continued from previous page)

```
! (these steps are done for all slave MPI sides and then for all remaining sides):
! 3.1)  Prolong solution to faces and store in U_master/slave.
!       Use them to build mortar data (split into 2/4 smaller sides).
![3.2)] The information which element is a DG or FV subcells element is stored
!       in FV_Elems per element.
![3.3)] The reconstruction of slopes over element interfaces requires,
!       besides U_slave and FV_Elems_slave, some more information that
!       has to be transmitted from the slave to the master MPI side.
! 3.4)  Finish all started MPI communications (after step 2. due to latency hiding)

#if USE_MPI
! Step 3 for all slave MPI sides
! 3.1) Prolong solution to faces and store in U_master/slave.
!      Use them to build mortar data (split into 2/4 smaller sides).
CALL StartReceiveMPIData(U_slave,DataSizeSide,1,nSides,MPIRequest_U(:,SEND),SendID=2) !␣
↪Receive MINE / U_slave: slave -> master
CALL StartReceiveSM_MPIData(PP_nVar,U_MorRot,MPIRequestSM_U,SendID=2) ! Receive MINE / U_
↪slave: slave -> master
CALL ProlongToFaceCons(PP_N,U,U_master,U_slave,L_Minus,L_Plus,doMPISides=.TRUE.)
CALL U_MortarCons(U_master,U_slave,doMPISides=.TRUE.)
CALL U_MortarConsSM(U_master,U_slave,U_MorStat,U_MorRot,doMPISides=.TRUE.)
CALL StartSendMPIData(   U_slave,DataSizeSide,1,nSides,MPIRequest_U(:,RECV),SendID=2) !␣
↪SEND YOUR / U_slave: slave -> master
CALL StartSendSM_MPIData(   PP_nVar,U_MorRot,MPIRequestSM_U,SendID=2) ! SEND YOUR / U_
↪slave: slave -> master
#if FV_ENABLED
! 3.2) The information which element is a DG or FV subcells element is stored
!      in FV_Elems per element.
CALL FV_Elems_Mortar(FV_Elems_master,FV_Elems_slave,doMPISides=.TRUE.)
CALL StartExchange_FV_Elems(FV_Elems_slave,1,nSides,MPIRequest_FV_Elems(:,SEND),
↪MPIRequest_FV_Elems(:,RECV),SendID=2)
#endif /* FV_ENABLED */
```

2.3.5 Special Rules

CALL Allocate_Shared()

Subroutine calls for *Allocate_Shared()* must be placed in a single line, i.e., no line break is allowed due to the compile flag *PICLAS_DEBUG_MEMORY*, which changes the number of arguments of the call for this routine by adding a debugging parameter.

USE MOD_Preproc

Using variables via *USE MOD_Preproc, ONLY: PP_N* is not possible due to the compile flag *PICLAS_POLYNOMIAL_DEGREE*, which switches the polynomial degree between a constant values, e.g., 1, 2, 3 .. or the default value of *N*. Therefore, the *ONLY* statement is not allowed here.

2.4 Best Practices

The following collection of best practice guidelines are intended to prevent bugs and improve the computational performance.

2.4.1 MPI

The general rules can be summarized as follows:

1. The first rule of MPI is: You do not send subsets of arrays, only complete continuous data ranges.
2. The second rule of MPI is: You do not send subsets of arrays, only complete continuous data ranges.
3. Third rule of MPI: Someone sends non-continuous data, the simulation is over.
4. Fourth rule: Only two processors to a single send-receive message.
5. Fifth rule: Only one processor access (read or write) to a shared memory region.

Please also read the general implementation information and, e.g., mappings used for elements, sides and nodes in the chapter *MPI Implementation*.

2.4.2 Shared Memory Windows

The following principles should always be considered when using shared memory windows

- Only the node root process initializes the shared memory array

```
! Allocate the shared memory window
CALL Allocate_Shared((/nUniqueGlobalNodes/), NodeVolume_Shared_Win, NodeVolume_
↪Shared)

! Lock the window
CALL MPI_WIN_LOCK_ALL(0, NodeVolume_Shared_Win, IERROR)

! Set pointer
NodeVolume => NodeVolume_Shared

! Only CN root nullifies
IF (myComputeNodeRank.EQ.0) NodeVolume = 0.0

! This sync/barrier is required as it cannot be guaranteed that the zeros have been
! written to memory by the time the MPI_REDUCE is executed (see MPI specification).
! Until the Sync is complete, the status is undefined, i.e., old or new value or␣
↪utter
! nonsense.
CALL BARRIER_AND_SYNC(NodeVolume_Shared_Win, MPI_COMM_SHARED)
```

- When all processes on a node write exclusively to their separate region in the shared memory array, using designated elements IDs which are assigned to a single process only

```
! Get offset
! J_N is only built for local DG elements. Therefore, array is only filled for␣
↪elements on the same compute node
offsetElemCNProc = offsetElem - offsetComputeNodeElem

! Allocate shared array
CALL Allocate_Shared((/nComputeNodeElems/),ElemVolume_Shared_Win,ElemVolume_Shared)
...

! Calculate element volumes
DO iElem = 1,nElems
  CNElemID=iElem+offsetElemCNProc
  !--- Calculate and save volume of element iElem
  J_N(1,0:PP_N,0:PP_N,0:PP_N)=1./sJ(:,:,:,iElem)
  DO k=0,PP_N; DO j=0,PP_N; DO i=0,PP_N
    ElemVolume_Shared(CNElemID) = ElemVolume_Shared(CNElemID) +␣
↪wGP(i)*wGP(j)*wGP(k)*J_N(1,i,j,k)
  END DO; END DO; END DO
END DO
```

- When all processes on a node write to all regions in the shared memory array, an additional local array is required, which has to be reduced to the shared array at the end

```
CALL Allocate_Shared((/nSpecies,4,nSurfSample,nSurfSample,
↪nComputeNodeSurfTotalSides/),SampWallImpactEnergy_Shared_Win,SampWallImpactEnergy_
↪Shared)
CALL MPI_WIN_LOCK_ALL(0,SampWallImpactEnergy_Shared_Win,IERROR)
IF (myComputeNodeRank.EQ.0) SampWallImpactEnergy_Shared = 0.
CALL BARRIER_AND_SYNC(SampWallImpactEnergy_Shared_Win,MPI_COMM_SHARED)
ALLOCATE(SampWallImpactEnergy(1:nSpecies,1:4,1:nSurfSample,1:nSurfSample,
↪1:nComputeNodeSurfTotalSides))
SampWallImpactEnergy = 0.
SampWallImpactEnergy(SpecID,1,SubP,SubQ,SurfSideID) = SampWallImpactEnergy(SpecID,1,
↪SubP,SubQ,SurfSideID) + ETrans * MPF
CALL MPI_REDUCE(SampWallImpactEnergy,SampWallImpactEnergy_Shared,MessageSize,MPI_
↪DOUBLE_PRECISION,MPI_SUM,0,MPI_COMM_SHARED,IERROR)
CALL BARRIER_AND_SYNC(SampWallImpactEnergy_Shared_Win,MPI_COMM_SHARED)
```

- When possible, never read from the shared memory array in a round robin manner, as shown in this commit [eaff78c]. Split the work and then use `MPI_REDUCE` or `MPI_ALLREDUCE`. Instead of

```
CNVolume = SUM(ElemVolume_Shared(:))
```

where all processes traverse over the same memory addresses, which slows down the computation, use

```
offsetElemCNProc = offsetElem - offsetComputeNodeElem
CNVolume = SUM(ElemVolume_Shared(offsetElemCNProc+1:offsetElemCNProc+nElems))
CALL MPI_ALLREDUCE(MPI_IN_PLACE,CNVolume,1,MPI_DOUBLE_PRECISION,MPI_SUM,MPI_COMM_
↪SHARED,iError)
```

to split the operations and use MPI to distribute the information among the processes.

- Atomic MPI operations on shared memory
  - Example 1: Store the min/max extent when building the CN FIBGM [6350cc2]
  - Example 2: Use atomic MPI operations to read/write from contested shared memory [772c371]
  - The main idea is to access and change parts of a shared array with multiple processes to, e.g., sum up numbers from different processes and guarantee that in the end the sum is correct without having a predefined order in which the numbers are added to the entry in the shared array.

    In the example in [772c371], get the memory window while bypassing local caches

    ```
    CALL MPI_FETCH_AND_OP(ElemDone,ElemDone,MPI_INTEGER,0,INT(posElem*SIZE_INT,MPI_
    ↪ADDRESS_KIND),MPI_NO_OP,ElemInfo_Shared_Win,iError)
    ```

    Flush only performs the pending operations (getting the value)

    ```
    CALL MPI_WIN_FLUSH(0,ElemInfo_Shared_Win,iError)
    ```

    Using `MPI_REPLACE` makes sure that the correct value is written in the end by one of the processes in an undefined order.

    ```
    MPI_FETCH_AND_OP(haloChange,dummyInt,MPI_INTEGER,0,INT(posElem*SIZE_INT,MPI_
    ↪ADDRESS_KIND),MPI_REPLACE,ElemInfo_Shared_Win,iError)
    CALL MPI_WIN_FLUSH(0,ElemInfo_Shared_Win,iError)
    ```

2.4.3 Hawk

Before running a simulation, check out the HLRS Wiki pages Batch System PBSPro (Hawk).

Striping

Always use user-defined striping in the simulation case folders that are on the work spaces as the default stiping setting (dynamic striping) has caused massive problems in the past. Add the following code to your submit script

```
# Set fixed striping to avoid problems with the progressive Lustre file layout
# - Region 1 [0, 1GiB): Stripe-Size=1 MiB, Stripe-Count=1
#lfs setstripe -c 1 -S 1M $PBS_O_WORKDIR
# - Region 2 [1GiB, 4GiB): Stripe-Size=1 MiB, Stripe-Count=4
#lfs setstripe -c 4 -S 1M $PBS_O_WORKDIR
# - Region 3 [4 GiB, EOF): Stripe-Size=4 MiB, Stripe-Count=8
lfs setstripe -c 8 -S 4M $PBS_O_WORKDIR
```

Note that the correct line should be commented in and the other lines should be commented out, all depending on the size of your output files. Also consider the stripe settings for large mesh files just to be sure.

Species-zero bug

It has repeatedly occurred that particles with species index zero have been produced on hawk. This might be due to the output to .h5, which could reflect the previous section regarding the striping settings, but could also lie deeper the Lustre file system itself. If this problem occurs, the corrupted particles must be removed from the .h5 file by hand if a restart from such a corrupted file is performed in order to prevent piclas from crashing.

2.5 Troubleshooting

The following examples are the result of lengthy debugging sessions and can be checked if problems arise during the development process. Especially when using MPI, the debugging process can be quite cumbersome.

2.5.1 WriteArrayToHDF5() and the collective flag

Error Description

- One or more array elements in a .h5 file are corrupt, e.g., the element contains the value `1.16828195e-314` instead of the expected value of `7960`

Necessary Conditions

- `LIBS_USE_MPI=ON`: the error only occurs when using more than 1 process (a multi-node run with a large number of processes might yield a high chance to trigger this problem)

Sufficient Conditions

- `CALL WriteArrayToHDF5()` with `collective=.TRUE.` but not all processes enter this function

Finding the Bug

- This error can be found with a regression test that runs with `LIBS_USE_MPI=OFF` or one process and again with multiple processes at best using the multi-node feature `PICLAS_SHARED_MEMORY=OMPI_COMM_TYPE_CORE`

Resolution

- `CALL WriteArrayToHDF5()` with collective=.FALSE. when it is not 100% certain that all processes enter this routine

Explanation

- Setting `collective=.TRUE.` triggers the usage of `H5FD_MPIO_COLLECTIVE_F` (`collective=.FALSE.` uses `H5FD_MPIO_INDEPENDENT_F`) in `H5PSET_DXPL_MPIO_F(PList_ID, H5FD_MPIO_INDEPENDENT_F, iError)`, which configures the "transfer mode" in the hdf5 output. Collective MPI output requires that all processes take part in the operation!

git hashes

- One of these bugs was specifically fixed in 0b2f7b12ecdf84d095caeb8c4b35e08a8484ce42

2.5.2 Seemingly meaningless change in code triggers segmentation fault or slow down of the code

Error Description

- A user-defined read-in parameter is commented in/out which triggers the bug due to different values that are read into the parameter, e.g., setting `1e-14` or `10e-15` (which is basically the same value!) The suspected problem is that something is read from the memory, which should not be read, e.g., an uninitialized variable (that therefore points to a possibly random location in the memory) This causes an undefined state (non-deterministic outcome, different compilers/different machines yield different effects). The code might not crash but hang for a certain amount of time (this can be used to find the bug).

Necessary Conditions

- An integer variable that is used for (indirect) allocation of an array is uninitialized

Sufficient Conditions

- A function or subroutine is called that declares an array depending on an uninitialized integer variable

Finding the Bug

- When the code does not crash but instead hangs for a certain amount of time, the following print statements can be used (when not utilizing a debugger)

```
IPWRITE(UNIT_StdOut,'(I0,A,I0)') ": v "//TRIM(__FILE__)//" +",__LINE__
```

  to see the exact position in the code where the code hangs (due to a possible function call and subsequent allocation process) for a short period of time

Resolution

- Nullify all integer variables that are used for allocation per default

Explanation

- The following variable was declared in a subroutine

```
REAL :: DistriOld(SpecDSMC(PartSpecies(iPart1))%MaxElecQuant)
```

  but the integer `SpecDSMC(PartSpecies(iPart1))%MaxElecQuant` was not initialized because the corresponding model is not used in this specific case and therefore `DistriOld` is never used. It is however allocated with an undefined state, with undefined outcome! In this case the bug was fixed by using the "assignment to an allocatable array"

```
REAL,ALLOCATABLE              :: DistriOld(:)
...
DistriOld = ElectronicDistriPart(iPart1)%DistriFunc
```

  For gfortran, Fortran 2003's assignment to an allocatable array was introduced in 4.6, 2011-01-28.

git hashes

- One of these bugs was specifically fixed in 8d1129be95abc91bf56e94bf7f12987e4c214666

2.5.3 Possible memory leak detection when using MPICH

Error Description

- The error output looks like this

```
================================================================================================================================
PICLAS FINISHED! [ 2.41 sec ] [ 0:00:00:02 ]
================================================================================================================================
Abort(810645775): Fatal error in internal_Finalize: Other MPI error, error stack:
internal_Finalize(50)...........: MPI_Finalize failed
MPII_Finalize(400)..............:
MPID_Finalize(652)..............:
MPIDI_OFI_mpi_finalize_hook(856):
destroy_vni_context(1094).......: OFI domain close failed (ofi_init.c:1094:destroy_
↪vni_context:Device or resource busy)
[WARNING] yaksa: 4 leaked handle pool objects
```

and shows that piclas finishes successfully, but an MPI error is invoked afterwards. Note that last line containing "[WARNING] yaksa: 4 leaked handle pool objects" might not be there and sometimes reflects the number of processes minus one.

Necessary Conditions

- MPICH must be used instead of OpenMPI. The problem even occurs when only one single process is used.

Finding the Bug

- Activate `PICLAS_DEBUG_MEMORY=ON` and check all the pairs of, e.g.,

```
myrank=      0               Allocated ElemBaryNGeo_Shared_Win with WIN_SIZE =     ␣
↪             240
```

with

```
myrank=      0                         Unlocking ElemBaryNGeo_Shared_Win with MPI_
↪WIN_UNLOCK_ALL()
myrank=      0                     Freeing window ElemBaryNGeo_Shared_Win with     ␣
↪ MPI_WIN_FREE()
```

to find the missing `CALL UNLOCK_AND_FREE(ElemBaryNGeo_Shared_Win)` by running piclas and storing the output in, e.g., `std.out` and then running the following command

```
STDOUT='std.out'; dashes='----------------------------------------'; for line in
↪$(grep -o -P '(?<=Allocated).*(?=with)' ${STDOUT} | sort -u | xargs); do printf
↪'Checking [%s] %s' "$line" "${dashes:${#line}}"; NbrOfA=$(grep -iw "${line}" $
↪{STDOUT} | grep -ic "Allocated"); printf ' allocated %sx' "$NbrOfA"; NbrOfDA=
↪$(grep -iw "${line}" ${STDOUT} | grep -ic "Unlocking"); printf ' deallocated %sx'
↪"$NbrOfDA"; if [[ $NbrOfDA -lt $NbrOfA ]]; then echo " ---> Could not find MPI_
↪WIN_UNLOCK_ALL() and MPI_WIN_FREE() for this variable"; else echo "... okay"; fi;␣
↪done
```

Replace `std.out` in the command if a different file name is used. If a variable is not correctly freed, the output of the script should look like this

```
Checking [ElemSideNodeID_Shared_Win] --------------- allocated 2x deallocated 2x...␣
↪okay
```

(continues on next page)

(continued from previous page)

```
Checking [ElemMidPoint_Shared_Win] ----------------- allocated 2x deallocated 2x...␣
↪okay
Checking [ElemNodeID_Shared_Win] ------------------- allocated 2x deallocated 1x ---
↪> Could not find all required MPI_WIN_UNLOCK_ALL() and MPI_WIN_FREE() for this␣
↪variable
Checking [ElemBaryNGeo_Shared_Win] ----------------- allocated 2x deallocated 2x...␣
↪okay
Checking [ElemRadius2NGeo_Shared_Win] -------------- allocated 2x deallocated 2x...␣
↪okay
Checking [ElemCurved_Shared_Win] ------------------- allocated 2x deallocated 2x...␣
↪okay
```

Resolution

- Add the missing `CALL UNLOCK_AND_FREE(MYNAME_Win)`, where `MYNAME` is the name of the shared memory window.

Explanation

- `MPI_WIN_UNLOCK_ALL()` and `MPI_WIN_FREE()` must be applied to shared memory windows before `CALL MPI_FINALIZE(iError)` is called.

git hashes

- One of these bugs was specifically fixed in 1a151c24bab7ea22809d3d7554ff5ddf18379cf1

2.6 Code Extension

This section shall describe how to extend the code in a general way e.g. to implement new input and output parameters.

2.6.1 Surface Sampling & Output

Location: *piclas/src/particles/boundary/particle_boundary_sampling.f90*

The surface sampling values are stored in the `SampWallState` array and the derived output variables in `MacroSurfaceVal`, which can be extended to include new **optional** variables and to exploit the already implemented MPI communication. The variables `SurfSampSize` and `SurfOutputSize` define the size of the arrays is set in `InitParticleBoundarySampling` with default values as given in `piclas.h`

```
SurfSampSize = SAMPWALL_NVARS+nSpecies
SurfOutputSize = MACROSURF_NVARS
```

To add optional variables, you need to increase the sampling and/or output indices as shown in the example

```
IF(ANY(PartBound%SurfaceModel.EQ.1)) THEN
  SurfSampSize = SurfSampSize + 1
  SWIStickingCoefficient = SurfSampSize
  SurfOutputSize = SurfOutputSize + 1
END IF
```

To be able to store the new sampling variable at the correct position make sure to define the index (SWI: SampWallIndex) as well. The index variable `SWIStickingCoefficient` is defined in the `MOD_Particle_Boundary_Vars` and can be later utilized to write and access the `SampWallState` array at the correct position, e.g.

```
SampWallState(SWIStickingCoefficient,SubP,SubQ,SurfSideID) =␣
↪SampWallState(SWIStickingCoefficient,SubP,SubQ,SurfSideID) + Prob
```

The calculation & output of the sampled values is performed in `CalcSurfaceValues` through the array `MacroSurfaceVal`. In a loop over all `nComputeNodeSurfSides` the sampled variables can be averaged (or manipulated in any other way). The variable `nVarCount` guarantees that you do not overwrite other variables

```
IF(ANY(PartBound%SurfaceModel.EQ.1)) THEN
  nVarCount = nVarCount + 1
  IF(CounterSum.GT.0) MacroSurfaceVal(nVarCount,p,q,OutputCounter) =␣
↪SampWallState(SWIStickingCoefficient,p,q,iSurfSide) / CounterSum
END IF
```

Finally, the `WriteSurfSampleToHDF5` routine writes the prepared `MacroSurfaceVal` array to the `ProjectName_DSMCSurfState_Timestamp.h5` file. Here, you have define a variable name, which will be shown in the output (e.g. in ParaView)

```
IF (ANY(PartBound%SurfaceModel.EQ.1)) CALL AddVarName(Str2DVarNames,nVar2D_Total,
↪nVarCount,'Sticking_Coefficient')
```

The order of the variable names and their position in the `MacroSurfaceVal` array has to be the same. Thus, make sure to place the `AddVarName` call at the same position, where you placed the calculation and writing into the `MacroSurfaceVal` array, otherwise the names and values will be mixed up.

2.6.2 Arrays with size of PDM%maxParticleNumber

If an array is to store particle information, it is usually allocated with

```
ALLOCATE(ParticleInformation(1:PDM%maxParticleNumber))
```

But since PDM%maxParticleNumber is dynamic, this behavior must also be captured by this array. Therefore its size has to be changed in the routines `IncreaseMaxParticleNumber` and `ReduceMaxParticleNumber` in `src/particles/particle_tools.f90`.

```
IF(ALLOCATED(ParticleInformation)) CALL ChangeSizeArray(ParticleInformation,PDM
↪%maxParticleNumber,NewSize, Default)
```

Default is an optional parameter if the new array memory is to be initialized with a specific value. The same must be done for TYPES of size PDM%maxParticleNumber. Please check both routines to see how to do it.

2.6.3 Insert new particles

To add new particles, first create a new particle ID using the GetNextFreePosition function contained in `src/particles/particle_tools.f90`

```
NewParticleID = GetNextFreePosition()
```

This directly increments the variable PDM%CurrentNextFreePosition by 1 and if necessary adjusts PDM%ParticleVecLength by 1. If this is not desired, it is possible to pass an offset. Then the two variables will not be incremented, which must be done later by the developer. This can happen if the particle generation process is divided into several functions, where each function contains a loop over all new particles (e.g. `src/particles/emission/particle_emission.f90`).

```
DO iPart=1,nPart
    NewParticleID = GetNextFreePosition(iPart)
END DO
PDM%CurrentNextFreePosition = PDM%CurrentNextFreePosition + nPart
PDM%ParticleVecLength = MAX(PDM%ParticleVecLength,GetNextFreePosition(0))
```

For the new particle to become a valid particle, the inside flag must be set to true and various other arrays must be filled with meaningful data. See SUBROUTINE CreateParticle in `src/particles/particle_operations.f90`. A basic example of the most important variables is given below:

```
newParticleID = GetNextFreePosition()
PDM%ParticleInside(newParticleID) = .TRUE.
PDM%FracPush(newParticleID) = .FALSE.
PDM%IsNewPart(newParticleID) = .TRUE.
PEM%GlobalElemID(newParticleID) = GlobElemID
PEM%LastGlobalElemID(newParticleID) = GlobElemID
PartSpecies(newParticleID) = SpecID
LastPartPos(1:3,newParticleID) = Pos(1:3)
PartState(1:3,newParticleID) = Pos(1:3)
PartState(4:6,newParticleID) = Velocity(1:3)
```

2.7 Useful Functions

This chapter contains a summary of useful functions and subroutines that might be re-used in the future.

2.7.1 General Functions and Subroutines

Function	Module	Input	Output	Description
UNITVECT	MOD_Glob	3D vector	3D vector	Normalizes a given vector by dividing all vectors entries by the vector's magnitude
CROSSNOR	MOD_Glob	two 3D vectors	3D vector	Computes the cross product of two 3-dimensional vectors: cross=v1 x v2 and normalizes the resulting vector
CROSS	MOD_Glob	two 3D vectors	3D vector	Computes the cross product of two 3-dimensional vectors: cross=v1 x v2
VECNORM	MOD_Glob	3D vector	REAL	Computes the Euclidean norm (length) of a vector
DOTPROD	MOD_Glob	3D vector	REAL	Computes the dot product of a vector with itself

2.7.2 Particle Functions and Subroutines

Function (Module)	Input	Output	Description
isChargedParticle (MOD_part_tools)	particle ID	LOGICAL	Check if particle has charge unequal to zero
PARTISELECTRON (MOD_globals)	particle ID	LOGICAL	Check if particle is an electron by checking if the charge is equal to 1.602176634e-19 (division and nearest integer)
isDepositParticle (MOD_part_tools)	particle ID	LOGICAL	Check if particle is to be deposited on the grid
isPushParticle (MOD_part_tools)	particle ID	LOGICAL	Check if particle is to be pushed (integrated in time)
isInterpolatePart: (MOD_part_tools)	particle ID	LOGICAL	Check if the field at a particle's is to be interpolated (accelerated)
VeloFromDistribut: (MOD_part_tools)	distribution type, Tempergy	3D vector	**WIP**, Calculates a velocity vector from a defined velocity distribution and Tempergy (temperature [K] or energy [J] or velocity [m/s])
DiceUnitVector (MOD_part_tools)	None	3D vector	Calculates a normalized vector in 3D (unit space) in random direction
DiceDeflectedVelo (MOD_part_tools)	cRela2(post-collison), alphaVSS(iSpecA,iSpecB)	3D vector	Calculates scaled post-collision relative velocity vector in center-of-mass frame
	if alphaVSS>1 (VSS) also: , cRelaX,cRelaY,cRelaZ (pre-collision)		VSS case includes coordinate transformation due to anisotropic scattering
CreateParticle (MOD_part_tools)	species ID, position, element ID, velocity and internal energies	particle ID (optional)	Creates a new particle at a given position and energetic state and return the new particle ID (optional)
GetParticleWeight (MOD_part_tools)	particle ID	REAL	Determines the weighting factor of a particle

2.8 MPI Implementation

This chapter describes how PICLas subroutines and functions are parallelized. Please also read the general rules for using *MPI*.

2.8.1 General Remarks: Things to consider

Debug MPI

```
mpirun --mca mpi_abort_print_stack 1 -np 3 ~/piclas/build/bin/piclas parameter.ini DSMC.
↪ini
```

2.8.2 Construction of Halo Region (MPI 3.0 Shared Memory)

The general idea is to have the geometry information in a shared memory array on each node. This reduces the memory overhead and removes the need for halo regions between processors on the same node. An exemplary unstructured mesh is given in Fig. 2.1. The workflow is as follows:

1. Load the complete mesh into a shared memory window that is solely allocated by the compute-node root processor of size `1:nTotalElems` (only node coords, not derived properties, such as metric terms).
2. Load the elements that are assigned to processors on a single compute-node into an array of size `1:nSharedElems` (red elements in Fig. 2.1).
3. Find the compute-node halo elements and store in an array of size `1:nSharedHaloElems` (blue elements in Fig. 2.1).
    1. Background mesh (BGM) reduction
    2. Shared sides reduction
    3. Halo communicator

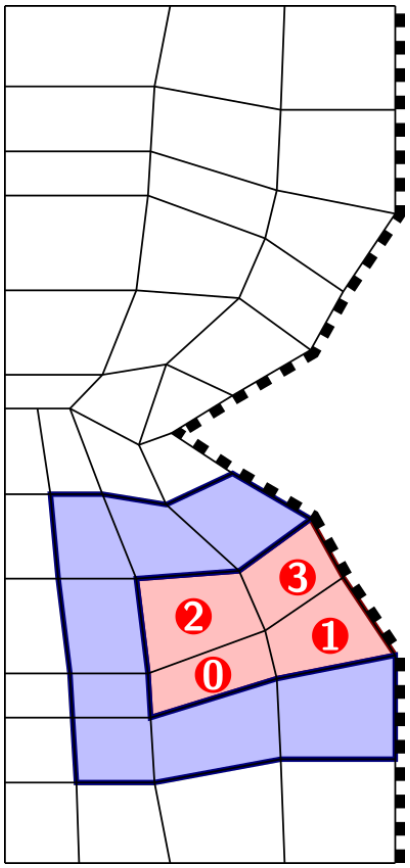

Fig. 2.1: Node local (red), halo (blue) and complete mesh (white). The red and blue geometry information is allocated once on each node and available to all processors on that node

Mesh Geometry

The complete mesh geometry is read-in by each compute-node and available through the ElemInfo_Shared, SideInfo_Shared, NodeInfo_Shared, NodeCoords_Shared arrays. Other derived variables and mappings (e.g. in `MOD_Particle_Mesh_Vars`) are usually only available on the compute-node.

ElemInfo

Element information is read-in and built in `ReadMeshElems`

```
ElemInfo_Shared(1:ELEMINFOSIZE,1:nGlobalElems)
! Possible preprocessor variables for first entry
! ElemInfo from mesh file
ELEM_TYPE         1    ! HOPR classification of element (currently not used)
ELEM_ZONE         2    ! Zone as defined in HOPR (currently not used)
ELEM_FIRSTSIDEIND 3    ! Index of the first side belonging to the element
ELEM_LASTSIDEIND  4    ! Index of the last side belonging to the element
ELEM_FIRSTNODEIND 5    ! Index of the first node belonging to the element
ELEM_LASTNODEIND  6    ! Index of the last node belonging to the element
! ElemInfo for shared array (only with MPI)
ELEM_RANK         7    ! Processor rank to which the element is assigned
ELEM_HALOFLAG     8    ! Element type:  1 = Regular element on compute node
                       !                2 = Halo element (non-periodic)
                       !                3 = Halo element (periodic)
```

SideInfo

Side information is read-in and built in `ReadMeshSides`

```
SideInfo_Shared(1:SIDEINFOSIZE+1,1:nNonUniqueGlobalSides)
! Possible preprocessor variables for first entry
SIDE_TYPE         1    ! HOPR classification of side
SIDE_ID           2    ! tbd
SIDE_NBELEMID     3    ! Neighbouring element ID
SIDE_FLIP         4    ! tbd
SIDE_BCID         5    ! Index of the boundary as given parameter.ini
SIDE_ELEMID       6    ! Element ID
SIDE_LOCALID      7    ! Local side ID
SIDE_NBSIDEID     8    ! Neighbouring side ID
SIDE_NBELEMTYPE   9    ! Neighbouring element type, 0 = No connection to another element
                       !                            1 = Neighbour element is on compute-
↪node
                       !                            2 = Neighbour element is outside of␣
↪compute node
```

NodeInfo

Node information is read-in and built in `ReadMeshNodes`

```
UniqueNodeID = NodeInfo_Shared(NonUniqueNodeID)
! Non-unique node coordinates per element
Coords(1:3) = NodeCoords_Shared(3,1:8*nGlobalElems)
```

Element/Side Mappings

To get the required element/side ID, utilize one of these functions from the module `MOD_Mesh_Tools`

```
! Global element
GlobalElemID = GetGlobalElemID(CNElemID)
! Compute-node element
CNElemID = GetCNElemID(GlobalElemID)
! Global side
GlobalSideID = GetGlobalSideID(CNSideID)
! Compute-node side
CNSideID = GetCNSideID(GlobalSideID)
```

If a loop is over the local number of elements, the compute-node and global element can be determined with the respective offsets.

```
DO iElem = 1, nElems
    CNElemID = iElem + offsetComputeNodeElem
    GlobalElemID = iElem + offsetElem
END DO
```

SurfSide

Additionally to conventional sides, mappings for the sides that belong to a boundary condition are available.

```
DO iSurfSide = 1,nComputeNodeSurfSides
  GlobalSideID = SurfSide2GlobalSide(SURF_SIDEID,iSurfSide)
END DO
```

Particle Element Mappings

```
PEM%GlobalElemID(iPart)         ! Global element ID
PEM%CNElemID(iPart)             ! Compute-node local element ID␣
↪(GlobalElem2CNTotalElem(PEM%GlobalElemID(iPart)))
PEM%LocalElemID(iPart)          ! Core local element ID (PEM%GlobalElemID(iPart) -␣
↪offsetElem)
```

2.8.3 Custom communicators

To limit the number of communicating processors, feature specific communicators can be built. In the following, an example is given for a communicator, which only contains processors with a local surface side (part of the `InitParticleBoundarySurfSides` routine). First, a global variable `SurfCOMM`, which is based on the `tMPIGROUP` type, is required:

```
TYPE tMPIGROUP
  INTEGER         :: UNICATOR=MPI_COMM_NULL     !< MPI communicator for surface sides
  INTEGER         :: nProcs                     !< number of MPI processes
  INTEGER         :: MyRank                     !< MyRank, ordered from 0 to nProcs - 1
END TYPE
TYPE (tMPIGROUP)    :: SurfCOMM
```

To create a subset of processors, a condition is required, which is defined by the `color` variable:

```
color = MERGE(1337, MPI_UNDEFINED, nSurfSidesProc.GT.0)
```

Here, every processor with the same `color` will be part of the same communicator. The condition `nSurfSidesProc.GT.0` in this case includes every processor with a surface side. Every other processor will be set to `MPI_UNDEFINED` and consequently be part of `MPI_COMM_NULL`. Now, the communicator itself can be created:

```
CALL MPI_COMM_SPLIT(MPI_COMM_PICLAS, color, MPI_INFO_NULL, SurfCOMM%UNICATOR, iError)
```

`MPI_COMM_PICLAS` denotes the global PICLas communicator containing every processor (but can also be a previously created subset) and the `MPI_INFO_NULL` entry denotes the rank assignment within the new communicator (default: numbering from 0 to nProcs - 1). Additional information can be stored within the created variable:

```
IF(SurfCOMM%UNICATOR.NE.MPI_COMM_NULL) THEN
  ! Stores the rank within the given communicator as MyRank
  CALL MPI_COMM_RANK(SurfCOMM%UNICATOR, SurfCOMM%MyRank, iError)
  ! Stores the total number of processors of the given communicator as nProcs
  CALL MPI_COMM_SIZE(SurfCOMM%UNICATOR, SurfCOMM%nProcs, iError)
END IF
```

Through the IF clause, only processors that are part of the communicator can be addressed. And finally, it is important to free the communicator during the finalization routine:

```
IF(SurfCOMM%UNICATOR.NE.MPI_COMM_NULL) CALL MPI_COMM_FREE(SurfCOMM%UNICATOR,iERROR)
```

This works for communicators, which have been initialized with MPI_COMM_NULL, either initially during the variable definition or during the split call. If not initialized initially, you have to make sure that the freeing call is only performed, if the respective split routine has been called to guarantee that either a communicator exists and/or every (other) processor has been set to MPI_COMM_NULL.

Available communicators

Handle	Description	Derived from
MPI_COMM_WORLD	Default global communicator	-
MPI_COMM_PICLAS	Duplicate of MPI_COMM_WORLD	MPI_COMM_PICLAS
MPI_COMM_NODE	Processors on a node	MPI_COMM_PICLAS
MPI_COMM_LEADERS	Group of node leaders	MPI_COMM_PICLAS
MPI_COMM_WORKERS	All remaining processors, who are not leaders	MPI_COMM_PICLAS
MPI_COMM_SHARED	Processors on a node	MPI_COMM_PICLAS
MPI_COMM_LEADERS_SHAR	Group of node leaders (myComputeNodeRank = 0)	MPI_COMM_PICLAS
MPI_COMM_LEADERS_SURF	Node leaders with surface sides	MPI_COMM_LEADERS_SHARED

Feature-specific

Handle	Description	Derived from
PartMPIInitGroup(nInitRegions)%COMM	Emission groups	MPI_COMM_PICLAS
SurfCOMM%UNICATOR	Processors with a surface side (e.g. reflective), including halo sides	MPI_COMM_PICLAS
CPPCOMM%UNICATOR	Coupled power potential	MPI_COMM_PICLAS
EDC%COMM(iEDCBC)%UNICATC	Electric displacement current (per BC)	MPI_COMM_PICLAS
FPC%COMM(iUniqueFPCBC)%UNI	Floating potential (per BC)	MPI_COMM_PICLAS
EPC%COMM(iUniqueEPCBC)%UNI	Electric potential (per BC)	MPI_COMM_PICLAS
BiasVoltage%COMM%UNICATOR	Bias voltage	MPI_COMM_PICLAS

2.9 Regression Testing

The purpose of regression testing is summarized by the following quote:

> *Regression testing (rarely non-regression testing) is re-running functional and non-functional tests to ensure that previously developed and tested software still performs after a change.*

Wikipedia: https://en.wikipedia.org/wiki/Regression_testing

2.9.1 Reggie2.0 Tool

PICLas is continuously tested by utilizing a Python based regression testing environment, which is run by gitlab-runners. Therefore, the tool *reggie2.0* is used, which is found under https://github.com/piclas-framework/reggie2.0

Additionally, the different analysis methods that can be applied and the general structure of a regression test is described there.

Different tests are executed on check-in, during nightly or weekly testing. These tests are defined in the file *.gitlab-ci.yml* that is located in the top level repository directory of PICLas. In this file, various tests are defined, which are found under *regressioncheck* and a summary of the different tests is given under https://github.com/piclas-framework/piclas/blob/master/REGGIE.md

The automatic execution by *gitlab-runners* can be performed on any machine that is connected to the internet and in the following section, the setup of such a machine is described

2.9.2 Local Testing of GitLab CI

To locally test the GitLab CI (including a YAML verification), gitlab-ci-local can be used. An installation guide can be found here: Link. After a successful installation, you can view the available parameters through

```
gitlab-ci-local --help
```

To view the stages for the default check-in pipeline, execute in the main folder of piclas:

```
gitlab-ci-local --list
```

To view all stages and tests:

```
gitlab-ci-local --list-all
```

To execute the check-in pipeline locally (ie. the jobs that were shown with the `--list` command), use

```
gitlab-ci-local --shell-isolation
```

to avoid errors due to parallel writing of the ctags.txt file. An alternative is to limit the concurrent execution to one job, which is analogous to the current configuration on the Gitlab Runner (requires gitlab-ci-local in version 4.42.0)

```
gitlab-ci-local --concurrency=1
```

It should be noted that currently the cache creation & utilization does not seem to represent the remote execution, meaning that some errors might only be recognized after a push to the remote. A specific job can be executed simply by reference its name, and to also consider the dependencies (ie. the `needs:`), the following command can be utilized to execute, for example the DSMC check-in job:

```
gitlab-ci-local --needs CHE_DSMC
```

Another useful option to check the resulting configuration file is

```
gitlab-ci-local --preview preview.yml
```

which gives the expanded version of utilized `extends:` and `<<:` templates.

2.9.3 Regression Server *Gitlab Runner* Setup

In a first step, the required software packages for PICLas and Reggie2.0 are installed on a new system. In a second step, the *gitlab-runner* program is installed and the setup of runner is described.

Required Installation of Software on Clean Ubuntu Setup (18.04)

Latest tests on

- Ubuntu (18.04), 3 Jul 2019
- Ubuntu server (18.04.3 LTS), 19 Nov 2019

The following packages can be installed automatically by using the script located at `./tools/Setup_ModuleEnv/InstallPackagesReggie.sh`. The system inquiries can be skipped by forcing `yes` as input via

```
yes | ./InstallPackagesRe
```

The script contains the following packages

```
# Check for updates
sudo apt-get update

# compiler
sudo apt-get install make cmake cmake-curses-gui gfortran g++ gcc

# python
sudo apt-get install python python-numpy python3 python3-numpy python3-matplotlib python-
↪matplotlib

# editors
sudo apt-get install gedit vim vim-runtime vim-common vim-tiny gdb vim-gui-common

# git && versions
sudo apt-get install git gitg qgit subversion

# paraview
sudo apt-get install libpython-dev libboost-dev libphonon-dev libphonon4 libxt-dev mesa-
↪common-dev
#apt-get install qt4-default qt4-dev-tools libqt4-dev qt4-qmake libqt4-opengl-dev
sudo apt-get install qttools5-dev libqt5x11extras5-dev qt5-default libgl1-mesa-dev

# Tecplot
sudo apt-get install libstdc++5

# tools
sudo apt-get install gzip gimp htop meld gnuplot gnuplot-x11 vlc okular ddd gmsh unzip
sudo apt-get install openvpn openssl openssh-client

# for FLEXI/PICLas
sudo apt-get install liblapack3 liblapack-dev zlib1g-dev exuberant-ctags

# for documentation
sudo apt-get install texlive-base
sudo apt-get install texlive-latex-extra

# hdf5-file viewer
sudo apt-get install hdfview

# Install libs for reggie
```

(continues on next page)

(continued from previous page)

```
sudo apt-get install python-h5py
```

When no module environment is to be used on the server, the following packages are also required

```
# Further libs
sudo apt-get install hdf5-tools libhdf5-dev # this is maybe not required (do not install␣
↪them if it works without these packages)


# openMPI
wget https://download.open-mpi.org/release/open-mpi/v3.1/openmpi-3.1.3.tar.gz
tar -xvf openmpi-3.1.3.tar.gz
cd openmpi-3.1.3
mkdir -p build
cd build
../configure --prefix=/opt/openmpi/3.1.3
sudo make all install
export PATH="/opt/openmpi/3.1.3/bin:$PATH"
export LD_LIBRARY_PATH="/opt/openmpi/3.1.3/lib:$LD_LIBRARY_PATH"
cd ../..
rm openmpi-3.1.3.tar.gz


#HDF5
wget https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.10/hdf5-1.10.5/src/hdf5-1.10.
↪5.tar.bz2
tar -xjf hdf5-1.10.5.tar.bz2
cd hdf5-1.10.5
mkdir -p build
cd build
cmake -DBUILD_TESTING=OFF -DHDF5_BUILD_FORTRAN=ON -DHDF5_BUILD_CPP_LIB=OFF -DHDF5_BUILD_
↪EXAMPLES=OFF -DHDF5_ENABLE_PARALLEL=ON -DHDF5
sudo make && sudo make install
export HDF5_DIR=/opt/hdf5/1.10.5/share/cmake
cd ../..
rm hdf5-1.10.5.tar.bz2
```

otherwise a module environment can be installed at this point, see `~/piclas/tools/Setup_ModuleEnv/README.md`, which is explained in detail in Chapter *Developer Tools* under Section *Module Environment*.

When no module environment is to be used on the server, the following commands must be places in the *.gitlab-ci.yml* file:

```
# Export the paths on new reggie2@reggie2 (no module env any more)
before_script:
  - ulimit -s unlimited
  - export PATH=/opt/openmpi/3.1.3/bin:$PATH
  - export LD_LIBRARY_PATH=/opt/openmpi/3.1.3/lib/:$LD_LIBRARY_PATH
  - export CMAKE_PREFIX_PATH=/opt/openmpi/3.1.3/share/cmake:$CMAKE_PREFIX_PATH
  - export CMAKE_LIBRARY_PATH=/opt/openmpi/3.1.3/lib:$CMAKE_LIBRARY_PATH
  - export HDF5_DIR=/opt/hdf5/1.10.5/share/cmake/
  - export PATH=/opt/hdf5/1.10.5/bin:$PATH
```

(continues on next page)

(continued from previous page)

```
  - export LD_LIBRARY_PATH=/opt/hdf5/1.10.5/lib/:$LD_LIBRARY_PATH
  - export CMAKE_PREFIX_PATH=/opt/hdf5/1.10.5/:$CMAKE_PREFIX_PATH
  - export CMAKE_LIBRARY_PATH=/opt/hdf5/1.10.5/lib:$CMAKE_LIBRARY_PAT
```

otherwise, the correct environment must be loaded by adding the following in `/etc/profile`

```
```
Default modules
module load gcc/9.2.0 cmake/3.15.3-d openmpi/4.0.1/gcc/9.2.0 hdf5/1.10.5/gcc/9.2.0/
↪openmpi/4.0.1
```
```

or by loading the modules directly in the gilab script file, e.g.,

```
```
module load XX/XX
```
```

NOTE: The stack size limit has been removed here by `ulimit -s unlimited`, which might be required by memory consuming programs

Installation Steps for Gitlab Runners

Latest tests on

- Ubuntu (18.04) with gitlab-runner 10.5.0 (10.5.0), 3 Jul 2019
- Ubuntu server (18.04.3 LTS) with gitlab-runner 10.5.0 (10.5.0), 19 Nov 2019

1. Install gitlab-runner from ubuntu packages (choose old version to avoid problems https://gitlab.com/gitlab-org/gitlab-runner/issues/1379) This creates the user gitlab-runner and a home directory (for 10.5 in /var/lib/gitlab-runner/)

   ```
   sudo apt-get install gitlab-runner
   ```

2. Start the runner program

   ```
   sudo gitlab-runner start
   ```

3. Register a runner using a shell executor (follow the information in the official guideline as it can vary slightly from version to version), on gitlab see `Settings` → `CI / CD Settings` → `Runners` (registration token during setup)

   ```
   sudo gitlab-runner register
   ```

4. Restart the runner

   ```
   sudo gitlab-runner restart
   ```

5. create ssh keys for normal user and set up password free access to gitlab (https://piclas.boltzplatz.eu)

   ```
   ssh-keygen -t ecdsa -b 521
   ```

   Add key to `Enabled deploy keys`. If multiple codes are on gitlab, add the key to one repository and select the key on the other repositories via `Privately accessible deploy keys`.

Clone a code from each platform to create known hosts then

```
sudo cp .ssh/*  /var/lib/gitlab-runner/.ssh
```

Check rights in this folder. If necessary make gitlab-runner own the files with

```
sudo chown -R gitlab-runner:gitlab-runner /var/lib/gitlab-runner/.ssh/
```

If the runner is used to push to remote repositories, add the public key under *deploy keys* and execute, e.g.,

```
sudo -u gitlab-runner git clone git@github.com:piclas-framework/piclas.git piclas_
↪github
```

to establish the first connection with the new repository and add the repo IP to the list of known hosts.

6. Start pipeline in gitlab or github for testing of reggie

NOTE: Interesting information is found in `/etc/systemd/system/gitlab-runner.service`.

Configuration files

The runner services can be adjusted by changing the settings in the file

```
/etc/systemd/system/gitlab-runner.service
```

in which the runner configuration file is specified:

```
[Unit]
Description=GitLab Runner
After=syslog.target network.target
ConditionFileIsExecutable=/usr/bin/gitlab-runner

[Service]
StartLimitInterval=5
StartLimitBurst=10
ExecStart=/usr/bin/gitlab-runner "run" "--working-directory" "/var/lib/gitlab-runner/" "-
↪-config" "/etc/gitlab-runner/config.toml" "--service" "gitlab-runner" "--syslog" "--
↪user" "gitlab-runner"


Restart=always
RestartSec=120

[Install]
WantedBy=multi-user.target
```

The runner configuration settings can be edited by changing the settings in the file

```
/etc/gitlab-runner/config.toml
```

where the number of runners, the concurrency level and runner limits are specified:

```
concurrent = 2
check_interval = 0
```

(continues on next page)

(continued from previous page)

```
[[runners]]
  name = "myrunner1"
  url = "https://gitlab.com/"
  token = "XXXXXXXXXX"
  executor = "shell"
  limit = 1
  [runners.cache]

[[runners]]
  name = "myrunner2"
  url = "https://gitlab.com/"
  token = "XXXXXXXXXX"
  executor = "shell"
  limit = 1
  [runners.cache]

[[runners]]
  name = "myrunner3"
  url = "https://gitlab.com/"
  token = "XXXXXXXXXX"
  executor = "shell"
  limit = 1
  [runners.cache]
```

Automatic Deployment to other platforms

1. Add the required ssh key to the deploy keys on the respective platform (e.g. github)
2. Clone a code from the platform to update the list of known hosts. Do not forget to copy the information to the correct location for the runner to have access to the platform

   ```
   sudo cp~/.ssh/.ssh/known_hosts /var/lib/gitlab-runner/.ssh/known_hosts
   ```

   This might have to be performed via the gitlab-runner user, which can be accomplished by executing the following command

   ```
   sudo -u gitlab-runner git clone git@github.com:piclas-framework/piclas.git piclas_
   ↪github
   ```

3. PICLas deployment is performed by the gitlab runner in the *deployment stage*

   ```
   github:
     stage: deploy
     tags:
       - withmodules-concurrent
     script:
       - if [ -z "${DO_DEPLOY}" ]; then exit ; fi
       - rm -rf piclas_github || true ;
       - git clone -b master --single-branch git@piclas.boltzplatz.eu:piclas/piclas.
   ↪git piclas_github ;
       - cd piclas_github ;
   ```

(continues on next page)

(continued from previous page)

```
    - git remote add piclas-framework git@github.com:piclas-framework/piclas.git ;
    - git push --force --follow-tags piclas-framework master ;
```

This script clones the master branch of PICLas and deploys it on github.

2.10 Unit tests

Unit tests are used to test individual key units of the source code. Currently these key routines include:

- Functionality of Read-In tools.
- Functionality of matrix inversion via external library

2.10.1 Integration of unit test with CTest

These unit tests are integrated into the **PICLas** build process using the CTest tool. Usually CTest will be run every time you build **PICLas** and give you an overview on the exit status of each test that looks something like this:

```
Test project /home/piclas/build_
    Start 1: ReadInTools
1/2 Test #1: ReadInTools ......................   Passed    0.12 sec
    Start 2: MatrixInverse
2/2 Test #2: MatrixInverse ....................   Passed    0.12 sec

100% tests passed, 0 tests failed out of 2

Total Test time (real) =   0.24 sec
```

To manually run the tests after a build use the CTest command

```
ctest
```

in your build directory. The manual page of CTest can give you an overview of all available options.

If you don't want to run the test after each build there is a CMake option called PICLAS_UNITTESTS that can be used to turn the tests on and off. This is an advanced option that CCMake will only show if you enter the advanced mode by pressing the t key.

2.10.2 Implementation of unit tests

All unit tests are implemented in FORTRAN and can be found in the subdirectory unitTests in your **PICLas** directory alongside a seperate CMakeLists.txt and some binary input and reference files.

CMakeLists.txt

The CMakeLists.txt defines a custom function called add_unit_test which can be used in the CMakeLists.txt to add a single test to the CTest tool. The syntax is

```
add_unit_test(NAME SOURCEFILE.F90)
```

All tests are defined using this function. At the end of the CMakeLists.txt a custom target all_tests is defined which includes all unit tests and will run the ctest command after it has been build.

The whole CMakeLists.txt content is included in the main CMakeLists.txt if the option PICLAS_UNITTESTS is set to ON (default) by CMake.

General unit test structure

The general structure of the unit tests is the same in all cases. They are implemented as FORTRAN programs. The unit test will call a function or subroutine from the **PICLas** framework with input either set in the program itself or read from a binary file. The output of this call will then be compared to some precomputed reference results (also stored as binary files) with a certain tolerance to account for differences in e.g. compiler versions and system architecture. If the results are within the given tolerance, the test will be passed, otherwise it will fail by returning a value other than 0.

The programs usually also contain a command line option that can be uses to generate the reference solution from a code version that is known to work correctly.

Have a look at the source code of one of the already implemented unit tests if you want to have a more detailed idea about how to implement your own tests.

Generation of reference mesh data

Some of the unit tests require parts of the mesh data structure to be able to call the functions to be tested. For this purpose, a curved single element is created and all the mesh data stored as a binary file called `UnittestElementData.bin`. This binary file can then be read during runtime by the unit test programs.

To generate the curved single element mesh, run **HOPR** with the parameter file provided in the `unitTest` subdirectory of **PICLas**. To generate the binary file, run **PICLas** with the following command line argument and the parameter file provided in the `unitTest` subdirectory:

```
piclas --generateUnittestReferenceData parameter.ini
```

2.11 Compiler Options

Todo: scribed current compiler options and what their purpose is (what was encountered in the past in order for the options to be as they are now?)

- Release: optimized with -O3 for execution runs
- Debug: debugger options
- Sanitize: GNU sanitizer for further debugging

Compi Flag	Options	What does it do?
–ffpe-trap=lis	*invalid*	invalid floating point operation, such as SQRT(-1.0)
	zero	division by zero
	overflow	overflow in a floating point operation
	underflow	underflow in a floating point. **DO NOT USE**. Because a small value can occure, such as exp(-766.2). operation
	precision	loss of precision during operation
		Some of the routines in the Fortran runtime library, like **CPU_TIME**, are likely to trigger floating point exceptions when ffpe-trap=precision is used. For this reason, the use of ffpe-trap=precision is not recommended.
	denormal	operation produced a denormal value
-fbacktr		runtime error should lead to a backtrace of the error
-fcheck=	*all*	enable all run-time check
	array temp	Warns at run time when for passing an actual argument a temporary array had to be generated. The information generated by this warning is sometimes useful in optimization, in order to avoid such temporaries.
	boun	Enable generation of run-time checks for array subscripts and against the declared minimum and maximum values. It also checks array indices for assumed and deferred shape arrays against the actual allocated bounds and ensures that all string lengths are equal for character array constructors without an explicit typespec.
	do	Enable generation of run-time checks for invalid modification of loop iteration variables
	mem	Enable generation of run-time checks for memory allocation. Note: This option does not affect explicit allocations using theALLOCATE statement, which will be always checked.
	point	Enable generation of run-time checks for pointers and allocatables.
	recursion	Enable generation of run-time checks for recursively called subroutines and functions which are not marked as recursive. See also -frecursive. Note: This check does not work for OpenMP programs and is disabled if used together with -frecursiveand -fopenmp.
-fdump-core		Request that a core-dump file is written to disk when a runtime error is encountered on systems that support core dumps. This option is only effective for the compilation of the Fortran main program
-fstack-arrays		Adding this option will make the Fortran compiler put all local arrays, even those of unknown size onto stack memory. If your program uses very large local arrays it is possible that you will have to extend your runtime limits for stack memory on some operating systems. This flag is enabled by default at optimization level -Ofast.
-frepack arrays		In some circumstances GNU Fortran may pass assumed shape array sections via a descriptor describing a noncontiguous area of memory. This option adds code to the function prologue to repack the data into a contiguous block at runtime.This should result in faster accesses to the array. However it can introduce significant overhead to the function call, especially when the passed data is noncontiguous.
-finline-matmul limit=n		
-finit-local-zero		The -finit-local-zero option instructs the compiler to initialize local INTEGER, REAL, and COMPLEX variables to zero, LOGICALvariables to false, and CHARACTER variables to a string of null bytes

2.12 Developer Tools

This chapter gives an overview over the tools and scripts for developers contained in the **PICLas** repository It also provides references to the tutorials where their usage is explained.

2.12.1 WarningsCheck

The maximum number of allowed warnings is limited to 10 and can be checked by using the script *test_max_warnings.sh*. Navigate to the top-level directory of the repository and execute

```
./tools/test_max_warnings.sh
```

This creates a build directory *build_test_max_warnings* and launches *cmake*, which must be configured by the user (hit *c* for configure). The user must then supply the desired compilation flags and must complete configuration by hitting *c* again and then continue the script by generating the make files (hit *g* for generate). The number of warnings is then supplied.

2.12.2 Remove trailing white spaces

Script (RemoveTrailWhiteSpaces.sh) can be executed from any directory inside the project. Searches all files (*.f90, *.h) in gitroot/src directory and removes trailing white spaces. Before remove operation all files and number of changes are shown and user is asked wether action is to be performed.

2.12.3 Module Environment

A set of scripts given in `./tools/Setup_ModuleEnv/` can be used for setting up a module environment. The main steps are described in `./tools/Setup_ModuleEnv/README.txt`

Global settings to the appearance of the module env list can be changed in `/etc/profile`, e.g., by adding (sudo required)

```
# ------------------------------------------------------------------------
# Display module information in list form
alias module='module -l'

# ------------------------------------------------------------------------
```

FAQ: Common Problems

- After installing packages, cmake and GCC, the module environment is loaded by the script `./InstallMPIallCOMPILERS.sh` the first time. This can fail even though the environment can be loaded from a shell by hand (interactive shell).

2.13 Performance Analysis

This chapter describes different tools that can be utilized for measuring the computational performance of PICLas.

2.13.1 Extrae and Paraver

Extra is a performance instrumentation tool that generates Paraver trace files that is distributed under LGPL-2.1 License and can be downloaded from https://github.com/bsc-performance-tools/extrae. Paraver is a performance analysis GUI for visualizing the code tracing data: https://github.com/bsc-performance-tools/wxparaver They are part of the BSC tool set that is found under https://tools.bsc.es/downloads.

Note: `wxparavwer` can simply be downloaded as pre-compiled binary file as it is only used for viewing the results.

Installation

See the `README` and `INSTALL` file in the git repository of the package.

Code Instrumentation

Note: Tested with extrae version 3.8.3

In PICLas, the extrae code instrumentation for the very basic modules is already implemented, see the in-code statements, e.g.,

```
#ifdef EXTRAE
CALL extrae_eventandcounters(int(9000001), int8(1))
#endif

! Initialization

#ifdef EXTRAE
CALL extrae_eventandcounters(int(9000001), int8(0))
#endif
```

which spans the complete initialization phase. Other regions are the field and particle modules (pure DG, PIC, DSMC, etc.) and the instrumentation is activated by setting the PICLas compile flag

```
PICLAS_EXTRAE = ON
```

in the cmake settings.

Examples that are already instrumented are

Table 2.1: Examples of instrumented code blocks

Function	Intger Value	Source
Initialization	1	`./src/piclaslib.f90`
Load Balancing	2	`./src/loadbalance/loadbalance.f90`
Write State file to .h5	3	`./src/io_hdf5/hdf5_output.f90`
Field Solver (HDG with CG solver)	4	`./src/hdg/hdg.f90`
Particle Solver	5	`./src/timedisc/timedisc_TimeStepPoissonByBorisLeapfrog.f90`
Particle Push	5	`./src/timedisc/timedisc_TimeStep_BGK.f90`
`PerformTracking()`	50	`./src/particles/tracking/particle_tracking.f90`
`CALL UpdateNextFreePosition()`	51	`./src/timedisc/timedisc_TimeStep_BGK.f90`
`CALL BGK_DSMC_main()` or `BGK_main()`	52	`./src/timedisc/timedisc_TimeStep_BGK.f90`
Analysis	6	`./src/analyze/analyze.f90`

Tracing the code

Load the required Modules

On the target system, the extrae software packages must be installed and loaded via, e.g.,

```
module load extrae
```

Create *tracing.sh* and *extrae.xml* in the simulation directory

Create a shell script *tracing.sh* (must be executable) with the following content

```
#!/bin/bash

export EXTRAE_CONFIG_FILE=/path/to/extrae.xml
export LD_PRELOAD=${EXTRAE_HOME}/lib/libmpitracef.so

$*
```

where the path to the directory containing the *extrae.xml* file must be inserted.

Note: `LD_PRELOAD` might only required when no user-defined instrumentation is used. If `PICLAS_EXTRAE=ON` is used during compilation, the line with `LD_PRELOAD` can be commented out or removed.

Furthermore, a configuration file *extrae.xml* is required that defines which hardware counters, which should be traced

```
<?xml version='1.0'?>

<trace enabled="yes"
 home="/opt/hlrs/non-spack/performance/extrae/3.7.1-mpt-2.23-gcc-9.2.0"
 initial-mode="detail"
```

(continues on next page)

(continued from previous page)

```
 type="paraver"
>

  <openmp enabled="no" ompt="no">
    <locks enabled="no" />
    <taskloop enabled="no" />
    <counters enabled="yes" />
  </openmp>

  <pthread enabled="no">
    <locks enabled="no" />
    <counters enabled="yes" />
  </pthread>

  <counters enabled="yes">
    <cpu enabled="yes" starting-set-distribution="1">
      <set enabled="yes" domain="all" changeat-time="0">
        PAPI_TOT_INS,PAPI_TOT_CYC
      </set>
      <set enabled="no" domain="all" changeat-time="0">
        PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_VEC_SP,PAPI_SR_INS,PAPI_LD_INS,PAPI_FP_INS
        <sampling enabled="no" period="1000000000">PAPI_TOT_CYC</sampling>
      </set>
    </cpu>

    <network enabled="no" />

    <resource-usage enabled="no" />

    <memory-usage enabled="no" />
  </counters>

  <storage enabled="no">
    <trace-prefix enabled="yes">TRACE</trace-prefix>
    <size enabled="no">5</size>
    <temporal-directory enabled="yes">/scratch</temporal-directory>
    <final-directory enabled="yes">/gpfs/scratch/bsc41/bsc41273</final-directory>
  </storage>

  <buffer enabled="yes">
    <size enabled="yes">5000000</size>
    <circular enabled="no" />
  </buffer>

  <trace-control enabled="yes">
    <file enabled="no" frequency="5M">/gpfs/scratch/bsc41/bsc41273/control</file>
    <global-ops enabled="no"></global-ops>
  </trace-control>

  <others enabled="yes">
    <minimum-time enabled="no">10M</minimum-time>
    <finalize-on-signal enabled="yes"
```

(continues on next page)

(continued from previous page)

```
      SIGUSR1="no" SIGUSR2="no" SIGINT="yes"
      SIGQUIT="yes" SIGTERM="yes" SIGXCPU="yes"
      SIGFPE="yes" SIGSEGV="yes" SIGABRT="yes"
    />
    <flush-sampling-buffer-at-instrumentation-point enabled="yes" />
  </others>

  <sampling enabled="no" type="virtual" period="50m" variability="10m" />

  <dynamic-memory enabled="no" />

  <input-output enabled="no" />

  <syscall enabled="no" />

  <merge enabled="no"
    synchronization="default"
    tree-fan-out="16"
    max-memory="512"
    joint-states="yes"
    keep-mpits="yes"
    sort-addresses="yes"
    overwrite="yes"
  />

</trace>
```

Here, the MPI library with PAPI_TOT_INS and PAPI_TOT_CYC counters are traced. Note that the path to the extrae directory is defined under

```
home="/opt/hlrs/non-spack/performance/extrae/3.7.1-mpt-2.23-gcc-9.2.0"
```

User functions

Warning: This section is experimental!

Compile the code with

```
-finstrument-functions
```

and supply the names of the functions that are exclusively traced in a file `user-functions.dat` containing the hash and name of each function in a comma-separated list, e.g.,

```
000000000042d2e0#__mod_timedisc_MOD_timedisc
```

where the hash is acquired via

```
nm -a bin/piclas_extrae | grep -in timedisc
```

or from the lib via `nm -a lib/libpiclas.so` if the function is in the shared library. To use the `user-functions.dat` file in extrae, add the following block to the `extrae.xml` file.

```
<user-functions enabled="yes" list="/absolute/path/to/user-functions.dat" exclude-
↪automatic-functions="no">
  <counters enabled="yes" />
</user-functions>
```

where the absolute path to `user-functions.dat` is supplied.

Run the application

Run the application and convert the output to Paraver format

Note: The extrae instrumented executable has a different name, which ends on `_extrae`

Execute `mpirun` and pass the `tracing.sh` script

```
mpirun -np 32 tracing.sh piclas_extrae parameter.ini
```

The following command can be appended to the submit script directly after `mpirun`.

Convert the Extrae output for Paraver

The tracing output stored in *TRACE.mpits* is then converted to a Paraver file via

```
${EXTRAE_HOME}/bin/mpi2prv -f TRACE.mpits -o tracing.prv
```

e.g.,

```
/opt/hlrs/non-spack/performance/extrae/3.7.1-mpt-2.23-gcc-9.2.0/bin/mpi2prv -f TRACE.
↪mpits -o pilcas.32ranks.prv
```

or using mpirun

```
mpirun -np 64  ${EXTRAE_HOME}/bin/mpimpi2prv -f TRACE.mpits -o tracing.prv
```

which will create a file containing the tracing events (.prv), list of registered events (.pcf) and cluster topology description (.row).

Analysing the results with Paraver

Note: Tested with wxparaver version 4.9.2

Open Paraver

```
wxparaver
```

and load a trace file for Paraver

- Open the `.prv` file via *File* -> *Load Trace* and the possible quantities are already shown under *Workspaces*, e.g., *Useful+MPI+PAPI* . . . .

- To view one of these properties, go to *Hints* -> *Useful* -> *Useful Duration*, which opens a separate window displaying the data.
- It shows the MPI ranks vs. the wall time and shows the calculation time for each trace, i.e., how much of the wall time was actually spent for calculation (the useful part of the simulation).
- On the bottom left go to *Files & Window Properties* and select *Window Properties*.
- Under *Properties Mode*, change the value from *Basic* to *Full* and select the drop down box
  - Have a look at the field *values* -> ... -> `9000001` (piclas directives instrumented by hand) to see if they have been correctly used
  - *Filter* -> *Events* -> *Event type* and set *Function* to =
  - *Filter* -> *Events* -> *Event type* and set *Types* to `9000001`
  - *Filter* -> *Events* -> *Event value* and set *Function* to =
  - *Filter* -> *Events* -> *Event value* and set *Values* to `1`

  here, the actual tracing event number has to be used as defined in *Code Instrumentation*, e.g., `1` as for `int8(1)`. For a list of pre-defined settings, see *Examples of instrumented code blocks*
- Right-click into window *Useful Duration @ *.prv* -> *View* -> *Event Flags* to activate the user-instrumented events from *Code Instrumentation*

To synchronize the views between different windows, e.g., *Useful Duration* and *MPI call* or simply two windows *Useful Duration* that each display a different *Event value* to show where a function instrumentation starts and ends

- Right-click into window *Useful Duration @ *.prv* -> *Synchronize* -> *[ ] 1* (select a group)
- Right-click into window *MPI call @ *.prv* -> *Synchronize* -> *[ ] 1* (select a group)

2.13.2 Intel® VTune™

Intel® VTune™ is a performance analysis tool with GUI for applications running on x86 systems for Linux and Windows developed by Intel®.

VTune Installation

Download the Intel VTune Profiler Source files for Linux and extract the files:

- https://software.intel.com/en-us/vtune/choose-download#standalone
- https://software.intel.com/content/www/us/en/develop/articles/oneapi-standalone-components.html#vtune

A user guide can be found here:

- https://software.intel.com/content/www/us/en/develop/documentation/vtune-help/top/launch/getting-started.html

Install VTune via the command line script (or the GUI installer)

```
sudo ./install.sh
```

The installed environment is meant to run in a bash shell. The GUI can be started by

```
bash
source /opt/intel/vtune_profiler_2020.0.0.605129/vtune-vars.sh
vtune-gui
```

Compile PICLas with "-g" to produce debugging information in the operating system's native format, which is required for detailed analysis performed by VTune.

Batch jobs

VTune can also be run in batch mode without the GUI. For a list of available options, see

```
vtune -help
```

and

```
vtune -help collect
```

To run a simulation with 10 MPI threads and collect data, simply run

```
mpirun -np 10 vtune -collect memory-consumption -trace-mpi -result-dir ~/intel/vtune/
↪projects/feature_branch/r0001mc ~/piclas/particle.dev/build.vtune/bin/piclas parameter.
↪ini
```

and specify where the output data of vtune should be collected.

Usage

In the Vtune GUI, set the path to the executable, the parameters (parameter.ini DSMC.ini) and the working directory (where the executable is run).

Hit the "Start" button and wait. The piclas std-out is dumped directly into the shell where vtune-gui was launched. The output can be redirected to a shell that is displayed in VTune by Setting: Options → General → "Product output window"

2.13.3 Valgrind

Valgrind is a complete suite of tools for debugging/profiling licenced under GPL. The complete documentation can be found here.

Installation of Valgrind

Valgrind is provided through the repository of all major Linux distributions. Install it with the package manager of your choice.

Execution of Valgrind

Valgrind is composed of individual tools, each tailered to debug or profil a specific aspect. All tools need PICLas compiled with "-g" to produce debugging information in the operating system's native format.

Callgrind

Callgrind tracks the call graph and duration for each function.

```
valgrind --tool=callgrind ./piclas parameter.ini
```

The output file can be opened with KCacheGrind or converted using gprof2dot. The options `-n PERCENTAGE, --node-thres=PERCENTAGE / -e PERCENTAGE, --edge-thres=PERCENTAGE` eliminate nodes/edges below this threshold [default: 0.5].

```
gprof2dot -n0 -e0 ./callgrind.out.1992 -f callgrind > out.dot
dot -Tpng out.dot -o out.png
```

In both cases, make sure you have GraphViz installed.

Memcheck

Memcheck keeps track of every allocated memory and shows memory leaks or invalid accesses to memory/pointers. Run it with

```
valgrind --leak-check=full --show-leak-kinds=all --track-origins=yes -s --suppressions=/
↪share/openmpi/openmpi-valgrind.supp ./piclas parameter.ini
```

OpenMPI handles its memory independently, so memcheck will always report memory leaks due to OpenMPI. Using the provided suppressions hides these falls flags. Combining memcheck with the GCC sanitize flag should provide full memory coverage.

Massif

Massif keeps track of the current memory usage as well as the overall heap usage. It helps finding code segments that hold on to memory after they should. Run it with

```
valgrind --tool=massif ./piclas parameter.ini
```

The output file can be opened with massif-visualizer.

DHAT

DHAT tracks memory allocations and inspects every memory access to a block. It is exceptionally useful to find code segments where memory is allocated unused or rapidly re-allocated. Run it with

```
valgrind --tool=dhat ./piclas parameter.ini
```

Afterwards, open `//usr/lib/valgrind/dh_view.html` in a web browser and load the output file.

2.14 Building the AppImage Executable

2.14.1 piclas

Navigate to the piclas repository and create a build directory

```
mkdir build && cd build
```

and compile piclas using the following cmake flags

```
cmake .. -DCMAKE_INSTALL_PREFIX=/usr
```

and then

```
make install DESTDIR=AppDir
```

or when using Ninja run

```
DESTDIR=AppDir ninja install
```

Then create an AppImage (and subsequent paths) directory in the build folder

```
mkdir -p AppDir/usr/share/icons/
```

and copy the piclas logo into the icons directory

```
cp ../docs/logo.png AppDir/usr/share/icons/piclas.png
```

A desktop file should already exist in the top-level directory containing

```
[Desktop Entry]
Type=Application
Name=piclas
Exec=piclas
Comment=PICLas is a flexible particle-based plasma simulation suite.
Icon=piclas
Categories=Development;
Terminal=true
```

Next, download the AppImage executable

```
curl -L -O https://github.com/linuxdeploy/linuxdeploy/releases/download/continuous/
↪linuxdeploy-x86_64.AppImage
```

and make it executable

```
chmod +x linuxdeploy-x86_64.AppImage
```

Then run

```
./linuxdeploy-x86_64.AppImage --appdir AppDir --output appimage --desktop-file=../.
↪github/workflows/piclas.desktop
```

The executable should be created in the top-level directory, e.g.,

```
piclas-ad6830c7a-x86_64.AppImage
```

2.14.2 piclas2vtk and other tools

Other tools such as `piclas2vtk` and `superB` etc. are also included in the AppImage container and can be extracted via

```
./piclas-ad6830c7a-x86_64.AppImage --appimage-extract
```

The tools are located under `./squashfs-root/usr/bin/`. To make on those tools the main application of the AppImage, remove the AppDir folder

```
rm -rf AppDir
```

and then

```
make install DESTDIR=AppDir
```

or when using Ninja run

```
DESTDIR=AppDir ninja install
```

and change the following settings, e.g., for `piclas2vtk`

```
PROG='piclas2vtk'
cp ../.github/workflows/piclas.desktop ${PROG}.desktop
mkdir -p AppDir/usr/share/icons/
cp ../docs/logo.png AppDir/usr/share/icons/${PROG}.png
sed -i -e "s/Name=.*/Name=${PROG}/" ${PROG}.desktop
sed -i -e "s/Exec=.*/Exec=${PROG}/" ${PROG}.desktop
sed -i -e "s/Icon=.*/Icon=${PROG}/" ${PROG}.desktop
./linuxdeploy-x86_64.AppImage --appdir AppDir --output appimage --desktop-file=${PROG}.
↪desktop
```

This should create

```
piclas2vtk-ad6830c7a-x86_64.AppImage
```

2.14.3 Troubleshooting

If problems occur when executing the AppImage, check the troubleshooting section for possible fixes.

2.15 Markdown Examples

2.15.1 hyperlinks

one with a title. Unclear how to enforce new window.

2.15.2 Code environment

Either use fenced style (tildes)

```
if (a > 3) {
  moveShip(5 * gravity, DOWN);
}
```

or indented style (4 whitespaces)

```
if (a > 3) {
  moveShip(5 * gravity, DOWN);
}
```

Both works with pandoc and wordpress. Also see pandoc verbatim code.

2.15.3 Equations

(@gleichung1) $a = b * c$ As (@gleichung1) shows, blabla.

2.15.4 Bibtex, cite

Hindenlang [@Hindenlang2015]. Only works with pandoc!

[bibshow file=references.bib]

Hindenlang [bibcite key=Hindenlang2015], Gassner [bibcite key=gassner2011disp]

2.15.5 section references

2.15.6 Figures, caption

See Fig. 2.2 for an image from the web embedded in this documentation.

See Fig. 2.3 for embedding a local file.

Fig. 2.2: This is an example caption.

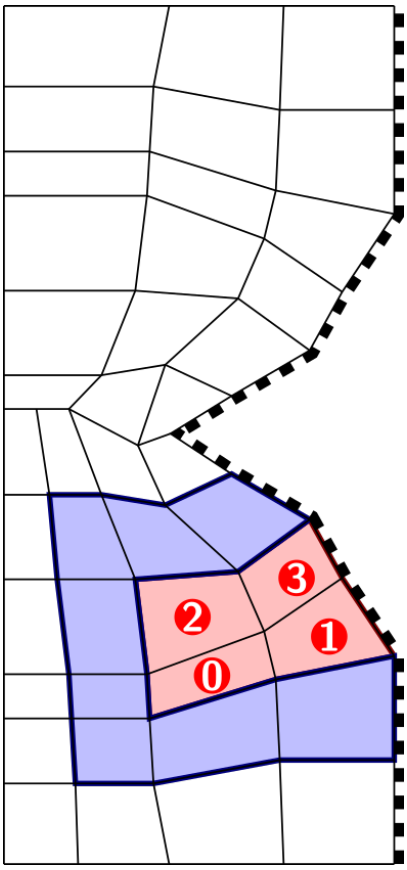

Fig. 2.3: This is an example caption.

2.15.7 tables

2.15.8 unnumbered section headings

just add

```
{-}
```

after the heading

2.15.9 Code blocks for various languages

```
int a = 32;
int a = 32;
```

This guide is organized to guide the first implementation steps as well as provide a complete overview of the simulation code's features from a developer's point of view.

- The first Chapter *GitLab Workflow* shall give an overview over the development workflow within the Gitlab environment, and the necessary steps to create a release, deploy updates to the Collaborative Numerics Group and GitHub.
- The second Chapter *Style Guide* describes the rules and guidelines regarding code development such as how the header of functions and subroutines look like.
- Chapter *Documentation* describes how to build the html/pdf files locally before committing changes to the repository and pointers on writing documentation.
- Chapter *Code Extension* describes how to extend standardized code blocks, e.g. add a new variable to the output.
- Chapter *Useful Functions* contains a summary of useful functions and subroutines that might be re-used in the future.
- Chapter *MPI Implementation* describes how PICLas subroutines and functions are parallelized.
- Chapter *Regression Testing* summarizes the PICLas' continuous integration through regression testing.
- Chapter *Unit tests* shows which unit tests are used to test individual key components of the source code.
- Chapter *Compiler Options* gives an overview of compiler options that are used in PICLas and their purpose.
- Chapter *Developer Tools* gives an overview over the tools and scripts for developers.
- Chapter *Performance Analysis* describes different tools that can be utilized for measuring the computational performance
- Chapter *Building the AppImage Executable* described how an AppImage executable of HOPR is created.
- Chapter *Markdown Examples* gives a short overview of how to include code, equations, figures, tables etc. in the user and developer guides in Markdown.

CHAPTER

THREE

REFERENCES

BIBLIOGRAPHY

[1] F Hindenlang, T Bolemann, and C-D. Munz. Mesh Curving Techniques for High Order Discontinuous Galerkin Simulations. In *IDIHOM: Industrialization of High-Order Methods-A Top-Down Approach*, pages 133–152. Springer, 2015.

[2] Matthias Sonntag. *Shape derivatives and shock capturing for the Navier-Stokes equations in discontinuous Galerkin methods*. PhD thesis, University of Stuttgart, 2017.

[3] David A Kopriva, Stephen L Woodruff, and M Yousuff Hussaini. Computation of electromagnetic scattering with a non-conforming discontinuous spectral element method. *International journal for numerical methods in engineering*, 53(1):105–122, 2001.

[4] Tan Bui-Thanh and Omar Ghattas. Analysis of an hp-nonconforming discontinuous Galerkin spectral element method for wave propagation. *SIAM Journal on Numerical Analysis*, 50(3):1801–1826, 2012.

[5] S.M. Copplestone, P. Ortwein, and C.-D. Munz. Complex-Frequency Shifted PMLs for Maxwell's Equations with Hyperbolic Divergence Cleaning and Their Application in Particle-in-Cell Codes. *IEEE Transactions on Plasma Science*, 2017. doi:10.1109/TPS.2016.2637061.

[6] M H Carpenter and C A Kennedy. Fourth-order 2N-storage Runge-Kutta Schemes. *NASA Technical Memorandum*, 109112:1–26, 1994.

[7] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Steven Benson, Jed Brown, Peter Brune, Kris Buschelman, Emil M. Constantinescu, Lisandro Dalcin, Alp Dener, Victor Eijkhout, Jacob Faibussowitsch, William D. Gropp, Václav Hapla, Tobin Isaac, Pierre Jolivet, Dmitry Karpeev, Dinesh Kaushik, Matthew G. Knepley, Fande Kong, Scott Kruger, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Lawrence Mitchell, Todd Munson, Jose E. Roman, Karl Rupp, Patrick Sanan, Jason Sarich, Barry F. Smith, Stefano Zampini, Hong Zhang, Hong Zhang, and Junchao Zhang. PETSc Web page. https://petsc.org/, 2022. URL: https://petsc.org/.

[8] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Steven Benson, Jed Brown, Peter Brune, Kris Buschelman, Emil Constantinescu, Lisandro Dalcin, Alp Dener, Victor Eijkhout, Jacob Faibussowitsch, William D. Gropp, Václav Hapla, Tobin Isaac, Pierre Jolivet, Dmitry Karpeev, Dinesh Kaushik, Matthew G. Knepley, Fande Kong, Scott Kruger, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Lawrence Mitchell, Todd Munson, Jose E. Roman, Karl Rupp, Patrick Sanan, Jason Sarich, Barry F. Smith, Stefano Zampini, Hong Zhang, Hong Zhang, and Junchao Zhang. PETSc/TAO users manual. Technical Report ANL-21/39 - Revision 3.18, Argonne National Laboratory, 2022.

[9] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, 163–202. Birkhäuser Press, 1997.

[10] S. M. Copplestone. *Particle-Based Numerical Methods for the Simulation of Electromagnetic Plasma Interactions*. PhD thesis, University of Stuttgart, 2019.

[11] Jose F. Padilla and Iain D. Boyd. Assessment of Gas-Surface Interaction Models for Computation of Rarefied Hypersonic Flow. *Journal of Thermophysics and Heat Transfer*, 23(June):96–105, 2009. doi:10.2514/1.36375.

[12] Min Lei, Xiaobin Wu, Wei Zhang, Xiaoping Li, and Xuedong Chen. The implementation of subsonic boundary conditions for the direct simulation Monte Carlo method in dsmcFoam. *Computers and Fluids*, 156:209–219, 2017. URL: http://dx.doi.org/10.1016/j.compfluid.2017.07.010, doi:10.1016/j.compfluid.2017.07.010.

[13] Dmitry Levko and Laxminarayan L Raja. Breakdown of atmospheric pressure microgaps at high excitation frequencies. *Journal of Applied Physics*, 117(17):173303, 2015.

[14] Andreas Pflug, Michael Siemers, Thomas Melzig, Lothar Schaefer, and Günter Bräuer. Simulation of linear magnetron discharges in 2d and 3d. *Surface and Coatings Technology*, 260:411–416, 2014.

[15] Diederik Depla, Stijn Mahieu, and Roger De Gryse. Magnetron sputter deposition: linking discharge voltage with target properties. *Thin Solid Films*, 517(9):2825–2839, 2009.

[16] Hui Liu, Boying Wu, Daren Yu, Yong Cao, and Ping Duan. Particle-in-cell simulation of a hall thruster. *Journal of Physics D: Applied Physics*, 43(16):165202, 2010.

[17] AI Morozov and VV Savel'ev. Sructure of steady-state debye layers in a low-density plasma near a dielectric surface. *Plasma Physics Reports*, 30(4):299–306, 2004.

[18] J Theis, G Werner, T Jenkins, and J Cary. Computing the paschen curve for argon with speed-limited particle-in-cell simulation. *Phys. Plasmas*, 2021.

[19] A. V. Phelps and Z. Lj Petrovic. Cold-cathode discharges and breakdown in argon: surface and gas phase production of secondary electrons. *Plasma Sources Science Technology*, 8(3):R21–R44, August 1999. doi:10.1088/0963-0252/8/3/201.

[20] Ming Zeng, Hui Liu, Lei Qiao, Fufeng Wang, Hongyan Huang, and Daren Yu. Experimental investigation of dielectric wall material effects on low-power hemp thruster. *AIP Advances*, 10(8):085317, 2020.

[21] A Dunaevsky, Y Raitses, and NJ Fisch. Secondary electron emission from dielectric materials of a hall thruster with segmented electrodes. *Physics of Plasmas*, 10(6):2574–2577, 2003.

[22] Sylvain Coulombe and Jean-Luc Meunier. Thermo-field emission: a comparative study. *Journal of Physics D: Applied Physics*, 30:776–780, 3 1997. URL: https://iopscience.iop.org/article/10.1088/0022-3727/30/5/009, doi:10.1088/0022-3727/30/5/009.

[23] Ernest Y. Wu and Baozhen Li. The schottky emission effect: a critical examination of a century-old model. *Journal of Applied Physics*, 132:025105, 7 2022. URL: https://aip.scitation.org/doi/10.1063/5.0087909, doi:10.1063/5.0087909.

[24] Erin Farbar and Iain D. Boyd. Subsonic flow boundary conditions for the direct simulation Monte Carlo method. *Computers and Fluids*, 102:99–110, 2014. URL: http://dx.doi.org/10.1016/j.compfluid.2014.06.025, doi:10.1016/j.compfluid.2014.06.025.

[25] Martin W. Tysanner and Alejandro L. Garcia. Measurement bias of fluid velocity in molecular simulations. *Journal of Computational Physics*, 196(1):173–183, 2004. doi:10.1016/j.jcp.2003.10.021.

[26] Alejandro L. Garcia and Wolfgang Wagner. Generation of the Maxwellian inflow distribution. *Journal of Computational Physics*, 217(2):693–708, 2006. doi:10.1016/j.jcp.2006.01.025.

[27] G B Jacobs and J S Hesthaven. High-order Nodal Discontinuous \G\alerkin Particle-in-cell Method on Unstructured Grids. *J. Comput. Phys.*, 214(1):96–121, may 2006. URL: http://dx.doi.org/10.1016/j.jcp.2005.09.008, doi:10.1016/j.jcp.2005.09.008.

[28] A Stock, J Neudorfer, M Riedlinger, G Pirrung, G Gassner, R Schneider, S Roller, and C.-D. Munz. Three-Dimensional Numerical Simulation of a 30-GHz Gyrotron Resonator With an Explicit High-Order Discontinuous-\G\alerkin-Based Parallel Particle-In-Cell Method. *Plasma Science, IEEE Transactions on*, 40(7):1860–1870, 2012.

[29] Konstantin Hinsberger. *Development and Implementation of the Calculation of Magnetic Fields Within PICLas*. Bachelor Thesis, University of Stuttgart, 2017.

[30] Branko Ruscic, Reinhardt E. Pinzon, Melita L. Morton, Gregor Von Laszevski, Sandra J. Bittner, Sandeep G. Nijsure, Kaizar A. Amin, Michael Minkoff, and Albert F. Wagner. Introduction to active thermochemical tables: Several "Key" enthalpies of formation revisited. *Journal of Physical Chemistry A*, 108(45):9979–9997, 2004. doi:10.1021/jp047912y.

[31] Branko Ruscic, Reinhardt E. Pinzon, Gregor Von Laszewski, Deepti Kodeboyina, Alexander Burcat, David Leahy, David Montoy, and Albert F. Wagner. Active Thermochemical Tables: thermochemistry for the 21st century. *Journal of Physics: Conference Series*, 16:561–570, 2005. doi:10.1088/1742-6596/16/1/078.

[32] Marcel Pfeiffer, Asim Mirza, and Stefanos Fasoulas. A grid-independent particle pairing strategy for DSMC. *Journal of Computational Physics*, 246:28–36, 2013. URL: http://linkinghub.elsevier.com/retrieve/pii/S0021999113001964, doi:10.1016/j.jcp.2013.03.018.

[33] A. A. Shevyrin, Ye A. Bondar, and M. S. Ivanov. Analysis of repeated collisions in the DSMC method. *AIP Conference Proceedings*, 762:565–570, 2005. doi:10.1063/1.1941596.

[34] Hassan Akhlaghi, Ehsan Roohi, and Stefan Stefanov. On the consequences of successively repeated collisions in no-time-counter collision scheme in DSMC. *Computers and Fluids*, 161:23–32, 2018. URL: https://doi.org/10.1016/j.compfluid.2017.11.005, doi:10.1016/j.compfluid.2017.11.005.

[35] Krishnan Swaminathan-Gopalan and Kelly A. Stephani. Recommended direct simulation Monte Carlo collision model parameters for modeling ionized air transport processes. *Physics of Fluids*, 28(2):027101, feb 2016. URL: http://dx.doi.org/10.1063/1.4939719http://scitation.aip.org/content/aip/journal/pof2/28/2/10.1063/1.4939719, doi:10.1063/1.4939719.

[36] Brian L. Haas, David B. Hash, Graeme A. Bird, Forrest E. III Lumpkin, and H. A. Hassan. Rates of thermal relaxation in direct simulation Monte Carlo methods. *Physics of Fluids*, 6(6):2191, 1994. URL: http://scitation.aip.org/content/aip/journal/pof2/6/6/10.1063/1.868221, doi:10.1063/1.868221.

[37] Forrest E. Lumpkin, Brian L. Haas, and Iain D. Boyd. Resolution of differences between collision number definitions in particle and continuum simulations. *Physics of Fluids A: Fluid Dynamics*, 3(9):2282–2284, 1991. URL: https://doi.org/10.1063/1.857964, arXiv:https://doi.org/10.1063/1.857964, doi:10.1063/1.857964.

[38] Iain D. Boyd. Analysis of rotational nonequilibrium in standing shock waves of nitrogen. *AIAA Journal*, 28(11):1997–1999, 1990. URL: https://doi.org/10.2514/3.10511, arXiv:https://doi.org/10.2514/3.10511, doi:10.2514/3.10511.

[39] I. D. Boyd. Rotational and vibrational nonequilibrium effects in rarefied hypersonic flow. *Journal of Thermophysics and Heat Transfer*, 4:478–484, October 1990.

[40] Paolo Valentini, Chonglin Zhang, and Thomas E. Schwartzentruber. Molecular dynamics simulation of rotational relaxation in nitrogen: implications for rotational collision number models. *Physics of Fluids*, 24(10):106101, 2012. URL: https://doi.org/10.1063/1.4757119, arXiv:https://doi.org/10.1063/1.4757119, doi:10.1063/1.4757119.

[41] Roger C. Millikan and Donald R. White. Systematics of vibrational relaxation. *The Journal of Chemical Physics*, 39(12):3209–3213, 1963. URL: https://doi.org/10.1063/1.1734182, arXiv:https://doi.org/10.1063/1.1734182, doi:10.1063/1.1734182.

[42] Takashi Abe. Inelastic collision model for vibrational–translational and vibrational–vibrational energy transfer in the direct simulation monte carlo method. *Physics of Fluids*, 6(9):3175–3179, 1994. URL: https://doi.org/10.1063/1.868094, arXiv:https://doi.org/10.1063/1.868094, doi:10.1063/1.868094.

[43] Erin D. Farbar. *Kinetic Simulation of Rarefied and Weakly Ionized Hypersonic Flow Fields.* PhD thesis, University of Michigan, Horace H. Rackham School of Graduate Studies, 2010. URL: http://hdl.handle.net/2027.42/78779.

[44] Iain D. Boyd. Analysis of vibration-dissociation-recombination processes behind strong shock waves of nitrogen. *Physics of Fluids A: Fluid Dynamics*, 4(1):178–185, 1992. URL: https://doi.org/10.1063/1.858495, arXiv:https://doi.org/10.1063/1.858495, doi:10.1063/1.858495.

[45] Derek S. Liechty and Mark Lewis. Electronic Energy Level Transition and Ionization Following the Quantum-Kinetic Chemistry Model. *Journal of Spacecraft and Rockets*, 48(2):283–290, mar 2011. URL: http://arc.aiaa.org/doi/abs/10.2514/1.48826, doi:10.2514/1.48826.

[46] Marcel Pfeiffer. Extending the particle ellipsoidal statistical Bhatnagar-Gross-Krook method to diatomic molecules including quantized vibrational energies. *Physics of Fluids*, 30(11):116103, 2018. URL: http://aip.scitation.org/doi/10.1063/1.5054961, doi:10.1063/1.5054961.

[47] Jonathan M. Burt and Eswar Josyula. DSMC modeling of nonequilibrium electronic excitation and emission for hypersonic sensor applications. *45th AIAA Thermophysics Conference*, pages 1–16, 2015. doi:10.2514/6.2015-2511.

[48] A. Kramida, Yu. Ralchenko, and J. Reader. NIST Atomic Spectra Database (version 5.4). 2016. URL: http://physics.nist.gov/asd.

[49] K. P. Huber and G. Herzberg. IV. Constants of Diatomic Molecules. In *Molecular Spectra and Molecular Structure*. Van Nostrand Reinhold Company, Boston, Massachusetts, 1979. URL: http://link.springer.com/10.1007/978-1-4757-0961-2, doi:10.1007/978-1-4757-0961-2.

[50] G. Herzberg. III. Electronic spectra and electronic structure of polyatomic molecules. In *Molecular Spectra and Molecular Structure*. D. Van Nostrand Company, Princeton, New Jersey, 1966.

[51] Graeme A. Bird. *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*. Oxford Engineering Science, 2nd edition, 1994. ISBN 978-0198561958.

[52] Graeme A. Bird. The Q-K model for gas-phase chemical reaction rates. *Physics of Fluids*, 2011. URL: http://scitation.aip.org/content/aip/journal/pof2/23/10/10.1063/1.3650424, doi:10.1063/1.3650424.

[53] Antonio Fernández-Ramos, Benjamin A. Ellingson, Rubén Meana-Pañeda, Jorge M. C. Marques, and Donald G. Truhlar. Symmetry numbers and chemical reaction rates. *Theoretical Chemistry Accounts*, 118(4):813–826, jul 2007. URL: http://link.springer.com/10.1007/s00214-007-0328-0, doi:10.1007/s00214-007-0328-0.

[54] C.K. Birdsall. Particle-in-cell charged-particle simulations, plus Monte Carlo collisions with neutral atoms, PIC-MCC. *IEEE Transactions on Plasma Science*, 19(2):65–85, apr 1991. URL: http://ieeexplore.ieee.org/document/106800/, doi:10.1109/27.106800.

[55] V. Vahedi and M. Surendra. A Monte Carlo collision model for the particle-in-cell method: applications to argon and oxygen discharges. *Computer Physics Communications*, 87(1-2):179–198, 1995. doi:10.1016/0010-4655(94)00171-W.

[56] Leanne C. Pitchford, Luis L. Alves, Klaus Bartschat, Stephen F. Biagi, Marie Claude Bordage, Igor Bray, Chris E. Brion, Michael J. Brunger, Laurence Campbell, Alise Chachereau, Bhaskar Chaudhury, Loucas G. Christophorou, Emile Carbone, Nikolay A. Dyatko, Christian M. Franck, Dmitry V. Fursa, Reetesh K. Gangwar, Vasco Guerra, Pascal Haefliger, Gerjan J.M. Hagelaar, Andreas Hoesl, Yukikazu Itikawa, Igor V. Kochetov, Robert P. McEachran, W. Lowell Morgan, Anatoly P. Napartovich, Vincent Puech, Mohamed Rabie, Lalita Sharma, Rajesh Srivastava, Allan D. Stauffer, Jonathan Tennyson, Jaime de Urquijo, Jan van Dijk, Larry A. Viehland, Mark C. Zammit, Oleg Zatsarinny, and Sergey Pancheshnyi. LXCat: an Open-Access, Web-Based Platform for Data Needed for Modeling Low Temperature Plasmas. *Plasma Processes and Polymers*, 14(1-2):1–17, 2017. doi:10.1002/ppap.201600098.

[57] M. Hossein Gorji and Patrick Jenny. An efficient particle Fokker–Planck algorithm for rarefied gas flows. *Journal of Computational Physics*, 262:325–343, 2014. URL: http://linkinghub.elsevier.com/retrieve/pii/S0021999113008541, doi:10.1016/j.jcp.2013.12.046.

[58] Marcel Pfeiffer and M. Hossein Gorji. Adaptive particle–cell algorithm for Fokker–Planck based rarefied gas flow simulations. *Computer Physics Communications*, 213:1–8, 2017. URL: http://linkinghub.elsevier.com/retrieve/pii/S0010465516303575, doi:10.1016/j.cpc.2016.11.003.

[59] Eunji Jun, Marcel Pfeiffer, Luc Mieussens, and M. Hossein Gorji. Comparative Study Between Cubic and Ellipsoidal Fokker–Planck Kinetic Models. *AIAA Journal*, pages 1–10, mar 2019. URL: https://arc.aiaa.org/doi/10.2514/1.J057935, doi:10.2514/1.J057935.

[60] M. Pfeiffer, A Mirza, and P Nizenkov. Evaluation of particle-based continuum methods for a coupling with the direct simulation Monte Carlo method based on a nozzle expansion. *Physics of Fluids*, 31(7):073601, 2019. URL: http://dx.doi.org/10.1063/1.5098085http://aip.scitation.org/doi/10.1063/1.5098085, doi:10.1063/1.5098085.

[61] Marcel Pfeiffer. Particle-based fluid dynamics: Comparison of different Bhatnagar-Gross-Krook models and the direct simulation Monte Carlo method for hypersonic flows. *Physics of Fluids*, 30(10):106106, 2018. URL: http://aip.scitation.org/doi/10.1063/1.5042016, doi:10.1063/1.5042016.

[62] Marcel Pfeiffer and Paul Nizenkov. Coupled ellipsoidal statistical BGK-DSMC simulations of a nozzle expansion. *AIP Conference Proceedings*, 2132:070019, 2019. URL: http://aip.scitation.org/doi/abs/10.1063/1.5119573, doi:10.1063/1.5119573.

[63] Marcel Pfeiffer, Paul Nizenkov, and Stefanos Fasoulas. Extension of particle-based BGK models to polyatomic species in hypersonic flow around a flat-faced cylinder. *AIP Conference Proceedings*, 2132:100001, 2019. URL: http://aip.scitation.org/doi/abs/10.1063/1.5119596, doi:10.1063/1.5119596.

[64] Marcel Pfeiffer, Asim Mirza, and Paul Nizenkov. Multi-species modeling in the particle-based ellipsoidal statistical Bhatnagar–Gross–Krook method for monatomic gas species. *Physics of Fluids*, 33:036106, 2021. doi:10.1063/5.0037915.

[65] Cyril Galitzine and Iain D. Boyd. An adaptive procedure for the numerical parameters of a particle simulation. *Journal of Computational Physics*, 281:449–472, 2015. URL: http://dx.doi.org/10.1016/j.jcp.2014.10.044, doi:10.1016/j.jcp.2014.10.044.

[66] Julian Beyer, Marcel Pfeiffer, and Stefanos Fasoulas. Non-equilibrium radiation modeling in a gas kinetic simulation code. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 280:108083, 2022. URL: https://www.sciencedirect.com/science/article/pii/S0022407322000206, doi:https://doi.org/10.1016/j.jqsrt.2022.108083.

[67] Julian Beyer, Paul Nizenkov, Stefanos Fasoulas, and Marcel Pfeiffer. Simulation of radiating non-equilibrium flows around a capsule entering titan's atmosphere. In *32nd International Symposium on Rarefied Gas Dynamics*. 2022.

[68] Marcel Pfeiffer, Julian Beyer, Jérémie Vaubaillon, Pavol Matlovič, Juraj Tóth, Stefanos Fasoulas, and Stefan Löhle. Numerical simulation of an iron meteoroid entering into earth's atmosphere using dsmc and a radiation solver with comparison to ground testing data. *Icarus*, 407:115768, 2024. URL: https://www.sciencedirect.com/science/article/pii/S0019103523003457, doi:https://doi.org/10.1016/j.icarus.2023.115768.

[69] Peter J Schmid, Knud Erik Meyer, and Oliver Pust. Dynamic mode decomposition and proper orthogonal decomposition of flow in a lid-driven cylindrical cavity. In *8th International Symposium on Particle Image Velocimetry*, 25–28. 2009.

[70] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. CRC Press, 1988.

[71] Gustaaf Jacobs, Jan Hesthaven, and Giovanni Lapenta. Simulations of the weibel instability with a high-order discontinous galerkin particle-in-cell solver. In *44th AIAA Aerospace Sciences Meeting and Exhibit*, 1171. 2006.

[72] J. Allegre, D. Bisch, and J. C. Lengrand. Experimental rarefied heat transfer at hypersonic conditions over 70-degree blunted cone. *Journal of Spacecraft and Rockets*, 34(6):724–728, 1997. doi:10.2514/2.3302.

[73] James Moss, Virendra Dogra, Joseph Price, and David Hash. Comparison of dsmc and experimental results for hypersonic external flows. *30th Thermophysics Conference*, 1995. doi:10.2514/6.1995-2028.

[74] Paul Nizenkov, Peter Noeding, Martin Konopka, and Stefanos Fasoulas. Verification and validation of a parallel 3D direct simulation Monte Carlo solver for atmospheric entry applications. *CEAS Space Journal*, 9(1):127–137, 2017. doi:10.1007/s12567-016-0133-5.

[75] Julien Mathiaud, Luc Mieussens, and Marcel Pfeiffer. An es-bgk model for diatomic gases with correct relaxation rates for internal energies. *arXiv preprint arXiv:2202.10906*, 2022.

[76] Jens Niegemann, Richard Diehl, and Kurt Busch. Efficient low-storage Runge–Kutta schemes with optimized stability regions. *Journal of Computational Physics*, 231(2):364–372, 2012.

[77] Wladimir Reschke, Bartomeu Massuti-Ballester, Marcel Pfeiffer, Georg Herdrich, and Stefanos Fasoulas. Validation of DSMC and CFD based catalysis modelling using plasma wind tunnel flows. *AIP Conference Proceedings*, 2132:070020, 2019. URL: http://aip.scitation.org/doi/abs/10.1063/1.5119574, doi:10.1063/1.5119574.