ViennaWD

User Manual
v3.0.0

Institute for Microelectronics
Technische Universität Wien
Gußhausstraße 27-29 / E360
A-1040 Vienna, Austria

Principal Investigator:
Mihail Nedjalkov

Developers:
Paul Ellinghaus
Josef Weinbub
Matthias Glanz

Advisors:
Ivan Dimov
Dragica Vasileska
Siegfried Selberherr

Former Contributers:
Marek Pobjecky
Philipp Schwaha

Institute for Microelectronics, TU Wien
Gußhausstraße 27-29 / E360
A-1040 Vienna, Austria/Europe

Phone	+43-1-58801-36001
FAX	+43-1-58801-36099
Web	`http://www.iue.tuwien.ac.at/`

Contents

Chapter 1

Overview

ViennaWD is a suite of Monte Carlo simulation tools intended for the simulation of semiconductor devices in the classical domain and the investigation of quantum phenomena using the Wigner formalism. The suite encompasses the following simulation tools:

- Wigner Ensemble Monte Carlo (WEMC)
- Classical Ensemble Monte Carlo (CEMC)
- Phonon Decoherence (PD)

The purpose, use and installation of each tool is discussed self-consistently in the following chapters.

Chapter 2

Building Information

ViennaWD ships a central CMake [1] script, allowing to configure and build all simulators via one script. In the following, we will discuss general installation aspects, however, for simulator-specific aspects please consult the respective building-related sections.

2.1 Build Configuration

A default build configuration, which setup all simulators, is generated by:

```
$> cd ViennaWD/
$> mkdir build
$> cd build
$> cmake ..
```

Watch for errors or warnings reported by CMake during the configuration process.

CMake will automatically try to discover the required dependencies on your system.

Use CMake's GUI application (`cmake-gui`) to conveniently select build features.

2.2 Configuration Options

ViennaWD's build system allows to select the optimization level as well as the simulation tools to be generated by using the following CMake [1] options:

- -D CMAKE_BUILD_TYPE=Debug|Release|RelWithDebInfo (default=Release)
- -D BUILD_WEMC=OFF (default=ON)
- -D BUILD_CEMC=ON (default=OFF)
- -D BUILD_PD=ON (default=OFF)

2.3 Building

To build the simulators issue the following

```
$> make
```

Use `-jN`, where `N` refers to the number of parallel jobs/available CPU cores, to speed-up the build process.

Chapter 3

Wigner Ensemble Monte Carlo

3.1 Introduction

Wigner Ensemble Monte Carlo (WEMC) is a C-based simulator, with MPI parallelisation, for quantum transport simulations in one- and two-dimensional structures.

The Wigner formalism expresses quantum mechanics in terms of functions and variables in the phase space. Such a formulation allows a more intuitive insight, versus wave functions and operators, and allows the utilization of some classical notions, e.g. Boltzmann scattering. The Wigner function is known as a quasi-distribution function since, despite negative values occurring, it can be used in the same way as a true distribution function to calculate physical averages of interest, e.g. density or current. The Wigner equation describes the temporal evolution of the Wigner function, f_w. The choice of a finite coherence length L [2] yields the semi-discrete form of the Wigner equation, the one-dimensional version of which is defined as

$$\frac{\partial f_w}{\partial t} + \frac{\hbar q \Delta k}{m^*} \frac{\partial f_w}{\partial x} = \sum_{q=-K}^{K} V_w\left(x, q-q', t\right) f_w\left(x, q', t\right), \tag{3.1}$$

where q is an index which represents the quantized momentum, i.e. $p = \hbar\left(q\Delta k\right)$. The resolution Δk is determined by the coherence length, $\Delta k = \frac{\pi}{L}$. The effective mass is represented by m^* in the above equation. The Wigner potential V_w is defined as

$$\begin{aligned} V_w\left(x, q\right) &\equiv \frac{1}{i\hbar L}\int_{-\frac{L}{2}}^{\frac{L}{2}} ds\, e^{-i2q\Delta k\cdot s}\delta V\left(s; x\right); \\ \delta V\left(s; x\right) &\equiv V\left(x+s\right) - V\left(x-s\right). \end{aligned} \tag{3.2}$$

The WEMC simulator solves (3.1), using the signed-particle Monte Carlo method [3] for the one- and two-dimensional case. The evolution of an initial condition, e.g. a minimum uncertainty wave packet (default), can be investigated in the presence of a static potential profile. The minimum uncertainty wave packet is defined as

$$\psi\left(x, t_0\right) = \left(2\pi\sigma^2\right)^{-\frac{1}{4}} e^{-\frac{\left(x-x_0\right)^2}{4\sigma^2}} e^{ik_0 z}, \tag{3.3}$$

where σ, x_0 and k_0 represent the standard deviation (in space), the mean position and the mean momentum (i.t.o. k), respectively. The initial condition and the potential profile can be specified using input files (discussed in Section 3.3).

Boltzmann-type scattering mechanisms can be selectively activated. Currently, models for acoustic, Coulomb, optical (intravalley), XX/LL/XL intervalley and surface roughness are implemented for the silicon material system.

The simulator offers parallel execution in a distributed-memory environment, using MPI. The parallelization approach is based on a spatial domain decomposition where each MPI process is assigned a part of the domain and treats only particles present in its subdomain. Further details can be found in [4][5].

3.2 Building Information

3.2.1 Dependencies

The following list represents the required packages to build and execute the WEMC simulator. Note that a serial version and a parallel MPI version of the simulator can be compiled, requiring an appropriate MPI library.

- C-Compiler (e.g. GCC [6] Version ≥ 4.4)
- CMake [1] Version ≥ 2.6
- GSL [7] Version ≥ 1.16
- FFTW [8] Version $\geq 3.3.3$
- Lua [9] Version 5.1
- MPI (e.g. OpenMPI [10] Version ≥ 1.3) - optional -
- gnuplot [11] Version ≥ 4.2 - optional -
- Python [12] Version ≥ 3.0 - optional -

All external dependencies are usually shipped with the popular Linux distributions. For instance, on Linux Mint check for the following packages to install the aforementioned packages: cmake, libgsl0-dev, libfftw3-dev, liblua5.1-0-dev, openmpi-bin

3.2.2 Configuration Options

Several optional configuration targets are provided, such as:

- -D USE_MPI=OFF (default=ON)
- -D DEBUG_OUTPUT=OFF (default=ON)
- -D FILE_OUTPUT=OFF (default=ON)
- -D BUILD_TESTS=OFF (default=ON)

To build only the serial version of the simulator the following configuration can be used, for instance:

```
$> cmake -D USE_MPI=OFF ..
```

To build the parallel MPI version of the simulator in addition to the serial version, use the following configuration:

```
$> cmake -D USE_MPI=ON ..
```

3.3 Simulations

3.3.1 Simulator Control

The simulations are set up via text-based configuration file, which allows the user to specify various parameters. Furthermore, there exists the option to specify a potential profile and/or an initial state and/or a working directory where output files of the simulator are saved. The details of these input files/arguments are discussed in the following.

3.3.1.1 Configuration

The simulation parameters that can be specified by the user are fed to the simulator using a text-based Lua [9] input file. The parameters pertain to a specification of the simulation domain, computational aspects and physical models (amongst other things). The configuration file `examples/full_config.lua` explains all user-specifiable parameters along with their default values. Only certain parameters must be specified; optional parameters revert to default values if not specified.

The simulation domain is defined by specifying the minimum and maximum values along each axis, i.e. the boundaries of a rectangle. A one-dimensional domain must be specified using the y parameters; the x-values must be set to zero. Two kinds of boundary conditions can be specified, namely absorbing or reflecting, for each of the four boundaries (two in 1D case).

Other examples of configuration files are shipped with WEMC and are located under

```
wigner_ensemble_monte_carlo/examples/
```

The configuration file is passed to the simulator as the first argument. For the serial simulator this can be done by

```
$> cd ViennaWD/build/wigner_ensemble_monte_carlo/
$> ./wemcsim ../../wigner_ensemble_monte_carlo/examples/configuration.lua
```

or, when using the MPI-enabled simulator, by

```
$> cd ViennaWD/build/wigner_ensemble_monte_carlo/
$> mpirun -np 4 ./wemcsim_mpi \
   ../../wigner_ensemble_monte_carlo/examples/configuration.lua
```

3.3.1.2 Potential profile

The default potential profile is a square potential barrier of which the parameters can be specified in the Lua configuration file; if no parameters are specified in the configuration file a constant (zero) potential is assumed. The potential profile is hard-coded in the file `src/potential_profile.c`. This file contains several parameterized analytical formulas in functions, which can be extended and combined to construct a potential profile. Otherwise, the simulator also offers the possibility to specify a potential profile using a CSV file.

The filename of the potential profile must contain the string "potential" to be correctly identified as a potential input for the simulator, e.g. `potential_profile_example.csv`. Currently, the simulator only supports a uniform rectangular mesh. The user must ensure that the grid specified in the potential profile matches the dimensions and mesh size specified in the Lua input file. The profile is saved in a simple CSV format: the x-position [m], y-position [m] and potential [eV] are specified, using a space as a delimiter between them; each x-value is associated with a block of y-values. The file for a one-dimensional profile should contain an x-value of zero in the first column. For an example, refer to the files `examples/potential_*_example*.csv`.

The execution of the simulator with the specification of a potential profile in a CSV file is done as follows:

```
$> cd ViennaWD/build/wigner_ensemble_monte_carlo/
$> ./wemcsim ../../wigner_ensemble_monte_carlo/examples/configuration.lua \
   /dir_with_potential_files/potential_example.csv
```

3.3.1.3 Initial condition

The default initial condition, which can be specified in the Lua configuration file, is a single minimum uncertainty wave packet, as defined in (3.3) (and its 2D analogue). If a different initial condition is desired, the file `src/particle_initialization.c` can be modified. Alternatively, the initial condition can be specified by an input file. The initial condition can be specified using either a Wigner function or a particle ensemble.

The Wigner function is specified in a file by defining its value for each cell in the phase space. The values of the Wigner function can be negative, however, the sum of all values should still add up to 1. The absolute value of the Wigner function in each cell is multiplied by the value of `signed_total`, specified in the configuration file, and the resulting number of particles are generated in the cell with a sign corresponding to that of the Wigner function. The size of the phase space cells specified in the file must be compatible with the dimensions, mesh size and coherence length specified in the configuration file. The filename of the Wigner function must be named as `*wigner*.csv`, where the string "wigner" is used to correctly identify the file as an input for the initial condition in the form of a Wigner function.

The initial condition can also be specified through an input file defining the ensemble of particles with the properties of each particle. This method of specifying the initial condition is intended to start the simulator from a state simulated before and saved to file using the function `saveParticleStack()` in the the file `save_funcs.c`. In principle, the particle ensemble can also be generated otherwise (with all the required properties for each particle) and imported by the simulator. The simulator reads the input file using the function `readParticleStack()` in the file `particle_initialization.c`. The filename of the particle ensemble must be named as `*particle*.csv`, where the string "particle" is used to correctly identify the file as an input for the initial condition in the form of a particle ensemble.

The total number of initialized particles should not exceed the global maximum specified in the configuration file. A total of at most half the maximum value is advisable to accommodate particle generation using reasonable time steps.

The execution of the simulator with the specification of an initial condition in the form of a Wigner function in a CSV file is done as follows:

```
$> cd ViennaWD/build/wigner_ensemble_monte_carlo/
$> ./wemcsim ../../wigner_ensemble_monte_carlo/examples/configuration.lua \
   /dir_with_initial_conditions/wigner_IC_example.csv
```

Similarly, the MPI-enabled version can be called; an example where both a potential profile and an initial condition in the form of a particle ensemble is specified is done as follows:

```
$> cd ViennaWD/build/wigner_ensemble_monte_carlo/
$> mpirun -np 4 ./wemcsim_mpi \
   ../../wigner_ensemble_monte_carlo/examples/configuration.lua \
   /dir_with_initial_conditions/particle_ensemble_IC_example.csv \
   /dir_with_potentials/potential_example.csv
```

3.3.1.4 Working directory

The directory where files, containing the physical quantities specified in the Lua file, are saved can be specified by an additional argument:

```
$> mkdir ~/work_dir
$> cd ViennaWD/build/wigner_ensemble_monte_carlo/
$> mpirun -np 4 ./wemcsim_mpi \
   ../../wigner_ensemble_monte_carlo/examples/configuration.lua ~/work_dir
```

If no directory is specified, the default directory used is `/tmp/`.

3.3.2 Output

The WEMC simulator can calculate various physical quantities and distributions, which are saved in the working directory with a time stamp, using the file names and functions specified in Table 3.1. The associated functions can be found in `src/save_funcs.c` and are summoned at the specified save interval in the function `saveData()`. In the case of a parallel execution with MPI, the quantities associated to the subdomain of each process are saved and must be merged by a post-processing script (see Section 3.3.3).

The quantities are represented in the spatial and/or k domain(s) as histograms and follow a similar template: the range of the bins (i.e. minimum (inclusive) and maximum (exclusive) value) is followed by the probability of finding a particle within the stated range of the bin(s). Negative values may arise for 'probabilities' of (partial) Wigner distributions or in physical quantities due to numerical approximations/noise made when solving the Wigner equation.

The particle stack and Wigner function result in very large file size(s) and should not be saved regularly to avoid excessive file I/O.

Table 3.1: Simulator output

Data	File name	Function
Spatial distribution	density_t*.csv	`saveDensity()`
Momentum distribution	Kdistrib_t*.csv	`saveKdistrib()`
Particle stack	particleStack_t*.csv	`saveParticleStack()`
Wigner function	wignerFunciton_t*.csv	`saveWignerFunction()`
Potential profile	potential_profile_t*.csv	`savePotential()`
Particle generation rate	gamma.csv	`saveGamma()`
Energy distribution	energy_t*.csv	`saveEnergyDistrib()`
Current	current.csv	`saveCurrent()`
Density distribution f(x,kx)	x_kx_t*.csv	`saveXDensK()`
Density distribution f(y,ky)	y_ky_t*.csv	`saveYDensK()`

3.3.3 Post-processing

After the simulator has completed its simulation the generated data needs to be merged (in the case of parallel execution) and (optionally) plots of the data can be generated. The quantities to be saved/merged and (optionally) plotted are specified by flags, e.g. save_density, in the Lua input file. There are three possible values for each flag, explained in Table 3.2.

Table 3.2: Save flag options

Flag value	Meaning
0	no output data generated by the simulator
1	calculate and save quantity; merged by post_processing.py
2	calculate and save quantity; merged and plotted by post_processing.py

The Python script `tools/post_processing.py` performs the merging and plotting tasks by reading the simulation setup file, `SimulationSetup.txt`. This file is generated by the simulator during execution in `work_dir` and contains the values of the flags in the original input file and other information, e.g. the number of MPI processes, which are acquired at run-time. After the simulator (`wemcsim` or `wemcsim_mpi`) is finished the script `tools/post_processing.py` must be called manually; the working directory `work_dir`, which contains the generated output data, must be specified as an argument:

```
$> cd ViennaWD/wigner_ensemble_monte_carlo/tools/
$> python post_processing.py /work_dir/
```

In case of a parallel execution with `wemcsim_mpi`, the script merges the data files (prepended by a 'p' e.g. `p*_density_t*.csv`) generated by the MPI processes into one file, which represents the global domain, and then deletes the files associated with the sub-domains. The script saves the merged CSV files in the working directory (referred to as `work_dir`).

After merging the data files, plots are generated for the quantities specified with a flag value of 2 and can be found in the folder `work_dir/results`. The post-processing script prepares the data files (if needed) and calls the gnuplot scripts in `/plot_scripts`. The plots are saved as images in the GIF file format, which requires the gif terminal to be available in gnuplot. The terminals installed for gnuplot can be checked as follows:

```
$> gnuplot
$> set terminal
$> Available terminal types:
$>           canvas  HTML Canvas object
$>              cgm  Computer Graphics Metafile
$>         epscairo  eps terminal based on cairo
```

If `gif` is not listed, the package `libgd2-dev` should be installed. This library also provides the jpeg and png functionality of gnuplot. Alternatively, the file format for the generated plots can be changed manually in the gnuplot scripts in `/plot_scripts/` from gif to another terminal type.

All merging and plotting can still be done be performed individually by manually calling the various plot and merge scripts, which `post_processing.py` does automatically.

3.4 License

Chapter 4

Classic Ensemble Monte Carlo

4.1 Introduction

The Classic Ensemble Monte Carlo (CEMC) tool is designed for simulating two-dimensional MOSFET devices by means of solving the Boltzmann transport equation using the Ensemble Monte Carlo algorithm. It is based on Fortran code developed by Prof. Dragica Vasileska from Arizona State University. The original Fortran code has been translated to C and optimized.

Currently the implemented scattering mechanism include Coulomb scattering (Brooks-Herring approach), acoustic phonon scattering and intervalley scattering (zero-order f and g absorption and emission processes). The implemented 2D Monte Carlo Poisson solver solves the 2D Poisson-equation using the successive over-relaxation (SOR) method.

4.2 Building Information

4.2.1 Dependencies

In the following the dependencies of the CEMC simulator are presented.

- C-Compiler (e.g. GCC [6] Version ≥ 4.4)
- Math library (Unix Standard)

4.3 Examples

Three examples are shipped with this package.

The first example can be found in the directory

```
examples/example01
```

and contains input files with default values as described in Section 4.4, Table 4.1 and Table 4.2 (MOSFET device in on-state).

The second example is located in the directory

```
examples/example02
```

and contains the same device geometry with zero drain voltage (`inputVd = 0.0`) representing a MOSFET device in off-state.

The first two examples can be executed by the following.

```
$> cd ViennaWD/build/classic_ensemble_monte_carlo/
$> ./emc ../../classic_ensemble_monte_carlo/examples/example0x/device_dimensions.in
      ../../classic_ensemble_monte_carlo/examples/example0x/simulation_parameters.in
```

The third example is located in the directory

`examples/example03`

and contains an example of how to use the simulator kernel in an external C++ program. The example consists of a `main.cpp` C++ source code file where a minimalist code required to run the simulator is given. The following data structures

- `const_t constants;`
- `geometry_t *geometry;`
- `scatpar_t *scatpar;`
- `phys_quant_t *phys_quantities;`
- `output_t *outputdata;`

must be declared and allocated as shown. Then the user needs to initialize the simulation parameters shown in table 4.1 as well as the device dimension parameters shown in table 4.2. Also the following initialization functions

- `oooMatParInitialization(&constants);`
- `oooScatParInitialization(scatpar);`
- `oooDeviceStructureInitialization(constants, geometry, scatpar);`

must be called as shown to initialize the data structures.
The simulator itself is started by calling:
`outputdata = EMC(constants, geometry, scatpar, phys_quantities);`
This function has a return value of type `output_t *` and returns a pointer to the output data which can be processed in the next step. The output data structure is described in section 4.5. Finally the allocated memory needs to be freed in order to avoid memory leaks before exiting the program.

4.4 Simulation Control

The simulations can be controlled by two input files:

- `simulation_parameters.in`
- `device_dimensions.in`,

which are read in the main program as well as constants defined in `emc.h` and silicon material parameters implemented in `mat_par_initialization.c` located in the `src` folder. This section describes all relevant parameters.

4.4.1 Input file format

Simulation parameters contained in `simulation_parameters.in` are described in table 4.1.

Variable	**Definition**	**Default**
`scatpar->dt`	differential time step in $[s]$	$1.5e^{-16}$
`scatpar->totalTime`	total time to be simulated in $[s]$	$10e^{-12}$
`scatpar->transientTime`	duration of the initial transient in $[s]$	$3.0e^{-12}$
`scatpar->averTime`	averaging time in $[s]$	$2.0e^{-13}$
`phys_quantities->inputVs`	input source voltage in $[V]$	0.0
`phys_quantities->inputVd`	input drain voltage in $[V]$	1.0
`phys_quantities->inputVg`	input gate voltage in $[V]$	1.0
`phys_quantities->inputVsub`	input substrate voltage in $[V]$	0.0
`constants.omega`	relaxation factor for the SOR Poisson solver, should be in the range $1 \leq \omega \leq 2$	1.8
`constants.tolerance`	convergence tolerance for the Poisson solver	$1e^{-4}$

Table 4.1: The input variables in `simulaton_parameters.in` are described.

The device parameters contained in `device_dimensions.in` are described in table 4.2.

Variable	**Definition**	**Default**
`geometry->lengthSD`	length of the source and drain contacts in $[m]$	$50e^{-9}$
`geometry->depthSD`	depth of the source and drain contacts in $[m]$	$30e^{-9}$
`geometry->lengthG`	length of the gate in $[m]$	$25e^{-9}$
`geometry->depthB`	bulk depth in $[m]$	$70e^{-9}$
`geometry->deviceWidth`	width of the device in $[m]$	$1e^{-6}$
`geometry->oxideThickness`	thickness of the SiO_2 layer in $[m]$	$1.2e^{-9}$
`geometry->meshSize`	mesh size in $[m]$	$1e^{-9}$
`scatpar->dopingCon[0]`	doping concentration for region 1 in $[m^{-3}]$	$5e^{25}$
`scatpar->dopingCon[1]`	doping concentration for region 2 in $[m^{-3}]$	$5e^{25}$
`scatpar->dopingCon[2]`	doping concentration for region 3 in $[m^{-3}]$	$-5e^{24}$
`scatpar->dopingCon[3]`	doping concentration for region 4 in $[m^{-3}]$	$-5e^{23}$

Table 4.2: The input variables in `device_dimensions.in` are described.

4.4.2 `emc.h` header file description

The `emc.h` header file contains constant definitions as well as data structure and function declarations. Relevant macros are depicted in Table 4.3.

Constant	Definition	Default
`#define MAXNX`	maximum mesh x coordinate	200
`#define MAXNY`	maximum mesh y coordinate	200
`#define DOPREG`	maximum number of doping regions	4
`#define MAXSC`	maximum number of scattering mechanisms	10
`#define NLEV`	number of energy levels in the scattering table	1000
`#define MAXEN`	maximum number of electrons	1000000

Table 4.3: The constant definitions in `emc.h` are shown.

The user may need to modify the constants `MAXNX` and `MAXNY` if the default values exceed the ratios $\frac{device\ x-length}{mesh\ size}$ and $\frac{device\ y-length}{mesh\ size}$.

`MAXEN` may also be modified by the user in case the simulator reports the error *Actual number of electrons exceeds* `MAXEN`.

4.4.3 Material & Scattering parameters

The material parameters for Silicon are defined in `mat_par_initialization.c`. The table 4.4 shows material parameter variables that may be modified by the user if needed.

Variable	Definition	Default value
`TL`	lattice temperature in $[K]$	300.0
`am_l`	lateral effective mass	0.91
`am_t`	transversal effective mass	0.19
`constpar->eps_sc`	relative permittivity of Silicon	$11.8\,\varepsilon_0$
`constpar->eps_ox`	relative permittivity of SiO_2	$3.9\,\varepsilon_0$
`constpar->Ni`	intrinsic carrier density in Silicon	$1.45e^{16}$
`constpar->delta_Ec`	half band gap energy normalized by V_T	0.575/`constpar->vt`
`constpar->density`	Silicon density in $[kg/m^3]$	2329.0
`scatpar->vsound`	sound velocity in Silicon in $[m/s]$	9040.0
`constpar->af`	non-parabolicity factor	0.5

Table 4.4: User modifiable material parameters and their respective default values in `mat_par_initialization.c` are shown.

Scattering mechanisms are controlled by setting the variable switches listed in table 4.5.

By default, coulomb scattering, acoustic phonon scattering and intervalley zero-order scattering are enabled whereas intervalley first-order scattering and surface roughness are disabled.

Variable	**Mechanism**	**Value**
`scatpar->coulombscattering`	coulomb scattering	0 .. off, 1 .. on
`scatpar->acousticscattering`	acoustic phonon scattering	0 .. off, 1 .. on
`scatpar->intervalley0g`	intervalley phonon scattering zero order g-process	0 .. off, 1 .. on
`scatpar->intervalley0f`	intervalley phonon scattering zero order f-process	0 .. off, 1 .. on
`scatpar->intervalley1g`	intervalley phonon scattering first order g-process	0 .. off, 1 .. on
`scatpar->intervalley1f`	intervalley phonon scattering first order f-process	0 .. off, 1 .. on
`scatpar->surfaceroughness`	surface roughness	0 .. off, 1 .. on

Table 4.5: Scattering mechanism selection variables `mat_par_initialization.c` are shown.

Parameters for the scattering processes can be modified by setting the corresponding variables listed in table 4.6.

Variable	**Mechanism**	**Value**
`scatpar->sigma`	acoustic deformation potential in $[eV]$	6.55
`scatpar->defpot0g`	zero-order g-process deformation potential in $[eV/m]$	$5.23e^{10}$
`scatpar->defpot0f`	zero-order f-process deformation potential in $[eV/m]$	$5.23e^{10}$
`scatpar->defpot1g`	1st order g-process deformation potential in $[eV/m]$	0.0
`scatpar->defpot1f`	1st order f-process deformation potential in $[eV/m]$	0.0
`scatpar->phonon0g`	zero-order g-process phonon energy in $[eV]$	$63.0e^{-3}$
`scatpar->phonon0f`	zero-order f-process phonon energy in $[eV]$	$59.0e^{-3}$
`scatpar->phonon1g`	1st order g-process phonon energy in $[eV]$	$27.8e^{-3}$
`scatpar->phonon1f`	1st order f-process phonon energy in $[eV]$	$29.0e^{-3}$

Table 4.6: Scattering parameters with their default values in `mat_par_initialization.c` are shown.

4.5 Simulation output

4.5.1 Output data structure

All output quantities are saved in the data structure `output_t` defined in `emc.h`. This data structure can be accessed in the `main(...)` function directly or it can be written into files by calling `oooWriteOutput(...)` which generates most of the output files described in the next section. The structure itself is shown in the code snipped below.

```
typedef struct
{
  int totaltime;
  int x_max, y_max;
  int iterTotal;
  double x_axis[MAXNX],
         y_axis[MAXNY];

  currents_t *current_cumulative;
  curr_from_charge_t *current_from_charge;

  /* 2D quantities */
  double potential[MAXNX][MAXNY];
  double electrondensity[MAXNX][MAXNY];

  double fieldXY_x[MAXNX][MAXNY],
         fieldXY_y[MAXNX][MAXNY];

  double currentdensityX[MAXNX][MAXNY],
         currentdensityY[MAXNX][MAXNY],
         currentdensity[MAXNX][MAXNY];

  /* quantities along x direction with y = 0 */
  double fieldX[MAXNX], fieldX2[MAXNX],
         fieldY[MAXNX], fieldY2[MAXNY];
  double density[MAXNX];
  double sheetdensity[MAXNX];

  double velocityX[MAXNX],
         velocityY[MAXNX];
  double energy[MAXNX];
} output_t;
```

It contains 1D and 2D data arrays which can be accessed by `for`-loops going from 0 to `x_max` and `y_max` respectively where `x_max` and `y_max` are the maximum mesh dimensions.
`int totaltime` contains the total simulation runtime in seconds.
`int iterTotal` contains the total number of iterations used to access the pointers `*current_cumulative` and `*current_from_charge` holding source and drain current data. Those pointers point to an array of size iterTotal of items of the type

```
typedef struct
{
  double Is_cumul, Id_cumul;
  double Is_momentary, Id_momentary;
} currents_t;
```

and

```
typedef struct
{
  double Time;
  double sourceCurr, drainCurr;
  double sourceFactor, drainFactor;
} curr_from_charge_t;
```

respectively.
The type `currents_t` contain cumulative source (`Is_cumul`) and drain (`Id_cumul`) currents as well as momentary source (`Is_momentary`) and drain (`Id_momentary`) currents for each iteration.
The type `curr_from_charge_t` contains the simulation time (`Time`) at each iteration, the source (`sourceCurr`) and drain (`drainCurr`) currents calculated from charge and the source and drain factors which represent the no. of particles leaving the given contact - no. of eliminated particles + no. of created particles.

4.5.2 Output file format

The simulator generates the following output files:

```
cur_from_charge_SD.csv
current_cumulative.csv
current_densityX.csv
current_densityY.csv
current_density.csv
electron_density.csv
fields_density_x.csv
fieldXY.csv
potential.csv
rateAcoustic.csv
rateCoulomb.csv
rateIntervalleyAbf.csv
rateIntervalleyAbg.csv
rateIntervalleyEmf.csv
rateIntervalleyEmg.csv
sheet_density_x.csv
total_simulation_time.csv
v_e_aver.csv
x_axis.csv
y_axis.csv
```

Tables 4.7 and 4.8 show the file format for the output files, which is based on the comma-separated values (CSV) format. Visualization aspects are discussed in Appendix A.

FILE: `cur_from_charge_SD.csv`				
column 1	column 2	column 3	column 4	column 5
time $[ps]$	source current $[A]$	drain current $[A]$	source factor	drain factor

FILE: `currentcum.csv`				
column 1	column 2	column 3		
time $[ps]$	cumulative source current $[mA]$	cumulative drain current $[mA]$	momentary source current $[mA]$	momentary drain current $[mA]$

FILE: `current_densityX.csv`		
column 1	column 2	column 3
x-coordinate	y-coordinate	current density in x-direction $[mA/m^3]$

FILE: `current_densityY.csv`		
column 1	column 2	column 3
x-coordinate	y-coordinate	current density in y-direction $[mA/m^3]$

FILE: `current_density.csv`		
column 1	column 2	column 3
x-coordinate	y-coordinate	current density $[mA/m^3]$

FILE: `electron_density.csv`		
column 1	column 2	column 3
x-coordinate	y-coordinate	electron density $[1/m^3]$

FILE: `fields_density_x.csv`		
column 1	column 2	column 3
electric field x-component $[V/m]$	electric field y-component $[V/m]$	averaged sheet density $[1/m^2]$

FILE: `fieldXY.csv`			
column 1	column 2	column 3	column 4
x-coordinate	y-coordinate	electric field x-component in $[V/m]$	electric field y-component in $[V/m]$

FILE: `potential.csv`		
column 1	column 2	column 3
x-coordinate	y-coordinate	potential $[V]$

Table 4.7: File format for output files 1-8 is shown.

FILE: **`rateAcoustic.csv`**	
column 1	column 2
energy $[eV]$	acoustic scattering rate $[1/s]$

FILE: **`rateCoulomb.csv`**	
column 1	column 2
energy $[eV]$	coulomb scattering rate $[1/s]$

FILE: **`rateIntervalleyAbf.csv`**	
column 1	column 2
energy $[eV]$	intervalley f-process absorption scattering rate $[1/s]$

FILE: **`rateIntervalleyAbg.csv`**	
column 1	column 2
energy $[eV]$	intervalley g-process absorption scattering rate $[1/s]$

FILE: **`rateIntervalleyEmf.csv`**	
column 1	column 2
energy $[eV]$	intervalley f-process emission scattering rate $[1/s]$

FILE: **`rateIntervalleyEmg.csv`**	
column 1	column 2
energy $[eV]$	intervalley g-process emission scattering rate $[1/s]$

FILE: **`sheet_density_x.csv`**
column 1
sheet density in $[1/m^2]$

FILE: **`total_simulation_time.csv`**
column 1
total simulation runtime in $[s]$

FILE: **`v_e_aver.csv`**		
column 1	column 2	column 3
mean x-velocity $[m/s]$	mean y-velocity $[m/s]$	mean particle energy $[eV]$

FILE: **`x_axis.csv`**
column 1
x-axis mesh point coordinates in $[m]$

FILE: **`y_axis.csv`**
column 1
y-axis mesh point coordinates in $[m]$

Table 4.8: File format for output files 9-17 is shown.

4.6 License

Chapter 5

Phonon Decoherence

5.1 Introduction

The Phonon Decoherence (PD) tool comprises methods, approaches and algorithms for particle simulation of quantum decoherence in the phase space. The theoretical foundation for the implemented algorithms is the Wigner description from the field of quantum mechanics.

The evolution of a system in the Wigner formalism is remarkably similar to the classical phase space description resulting in Boltzmann's equation. The distinctive difference is that where the Boltzmann description is restricted to using distribution functions, the quantum case also admits negative values – hence using quasi distribution functions. This extension to negative values poses algorithmic difficulties in the general case.

Quantum computational and communication ideas rely on the fundamental physical notions of superposition, entanglement, and interference and thus to a coherent evolution.

Decoherence, which destroys the unitary evolution of the coherent state is the major show-stopper of an effective practical realization of the above ideas. The system interacts with the environment so that system and environment states entangle into a common, usually macroscopic state. The system state is obtained after a trace on the additional variables, which rules out certain correlations. The theory of decoherence addresses the manner in which some quantum systems become classical due to such entanglement with the environment. The latter in effect monitors certain observables in the system, destroying coherence between the states corresponding to their eigenvalues. Only preferred survive consecutive 'measurements' by the environment. The rest of the states, which actually comprise a major part of the Hilbert space are eliminated. Many of the features of 'classical' systems are actually induced in quantum systems by their environment.

The PD tool provides tools for analysis of the evolution of an initially entangled electron state which evolves in presence of semiconductor lattice vibrations - phonons. The initial electron state is constructed by a superposition of two Gaussian wave packets and has a pronounced interference term comprised of alternating positive and negative values of the Wigner function. The simulations show how the phonons effectively destroy the interference term. The initial coherence in wave vector distribution is pushed towards the equilibrium distribution. Phonons hinder the natural spread of the density with time pushing towards a classical localization. The initially pure electron state evolves towards a state with an entirely different physical interpretation: it is a mixed state where the electron can be with given probability in one of the two Gaussian packets. The decoherence effect of the phonons causing transition from quantum to classical state is demonstrated by the purity of the state, which decreases from its initial value of 1, with a speed depending on the lattice temperature.

The PD tool is considered a free open source platform to provide the research community with the actual implementations of published theoretical work in this field [13]. Based on the WIENS source code repository [14], PD tool extends the functionality with respect to an increased degree of decoupling and provides potential users with access to the result data structures.

5.2 Building Information

5.2.1 Dependencies

In the following the dependencies of the PD simulator are presented.

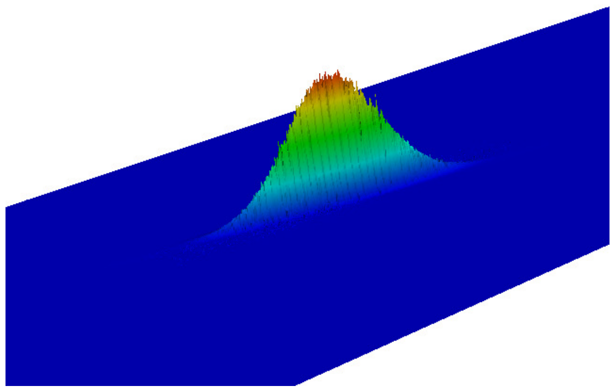
- C++11-Compiler (e.g. GCC [6] Version ≥ 4.6)
- Boost [15] Version ≥ 1.46
- Lua [9] Version 5.1

5.3 Examples

To execute the generated simulation executable, two input files have to be utilized. In the following, the provided example simulation setup is used to execute a simulation.

```
$> cd ViennaWD/build/phonon_decoherence/
$> ./pdsim ../../phonon_decoherence/examples/parameters.lua
              ../../phonon_decoherence/examples/config.lua
```

Fig. 5.1 depicts exemplary simulation results using the visualization approach introduced in Appendix A.



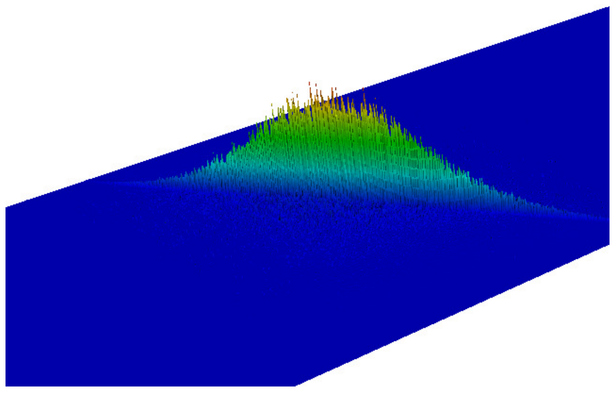


Figure 5.1: Phase space distribution functions computed by the PD simulator are shown. In the classical case it can be interpreted directly as probability to find a particle, the indefinite nature in the quantum case, prohibits this direct interpretation, but still provides correct derived quantities such as density and momentum distribution. The evolution is shown at $100fs$ (**top**) and at $1ps$ (**bottom**).

5.4 Simulation Control

The simulations are controlled via the input configuration and parameter XML files. In the following the individual variables are explained in detail.

Variable	**Definition**	**Unit**	**Default**
`TL`	Lattice Temperature	K	300
`effmass`	Effective Mass Factor		0.067
`alpha`	Non-Parabolicity Factor	$1/eV$	0.61
`rho`	Density	kg/m^3	$5.36e3$
`optical_phonon_energy`	Optical Phonon Energy	eV	0.0343
`polar_optical_phonon_energy`	Polar Optical Phonon Energy	eV	0.036
`optical_permitivity`	Optical Permitivity		10.92
`static_permitivity`	Static Permitivity		12.9
`free_flight_coeff`	Free Flight Coefficient	$1/Hz$	$1./2.6e13$

Table 5.1: The parameter variables are shown.

Variable	**Definition**	**Unit**	**Default**
`x_count`	Number of simulation domain points in x-direction		400
`y_count`	Number of simulation domain points in y-direction		3000
`timestep`	The time difference between two time steps	s	100.0e-15
`max_iterations`	Number of simulated time steps		10
`max_particle_count`	Number of generated particles for each point in the phase space		30

Table 5.2: The configuration variables are shown.

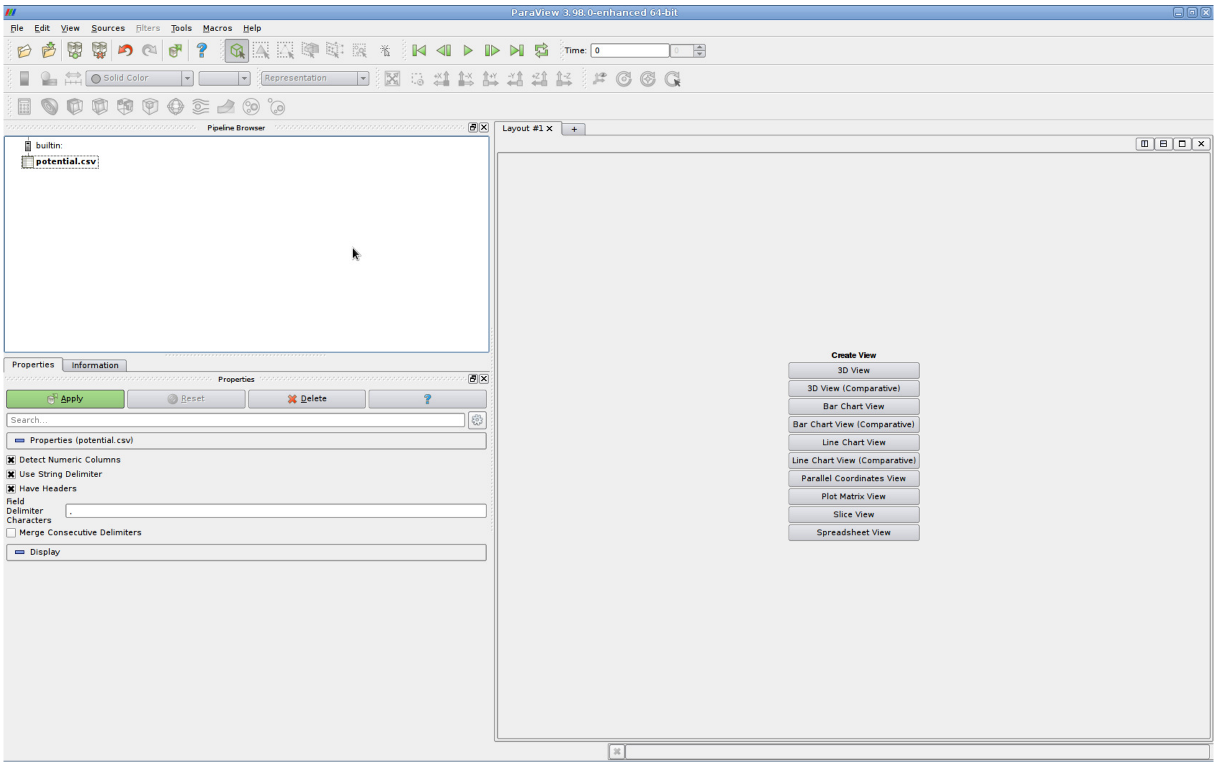
5.5 License

Appendix A

Visualization

The majority of the generated outputfiles of the provided simulation tools utilize the CSV format. This CSV format is supported by various available tools, such as the free open source tools ParaView [16], or Gnuplot [11] and the commercial MATLAB [17] suite.

In the following an exemplary visualization approach based on ParaView (Version 3.98.0) is depicted. The first step of processing a CSV file is to load it via the `Open File` dialog. As the the CSV output files generated by a ViennaWD simulator offer the `*.csv` file extension, ParaView applies the corresponding file-processer automatically. Fig. 5.2-5.8 show the required steps. Several Paraview state files, containing predefined filter sequences, are included with the WEMC simulator under `wigner_ensemble_monte_carlo/plot_scripts`

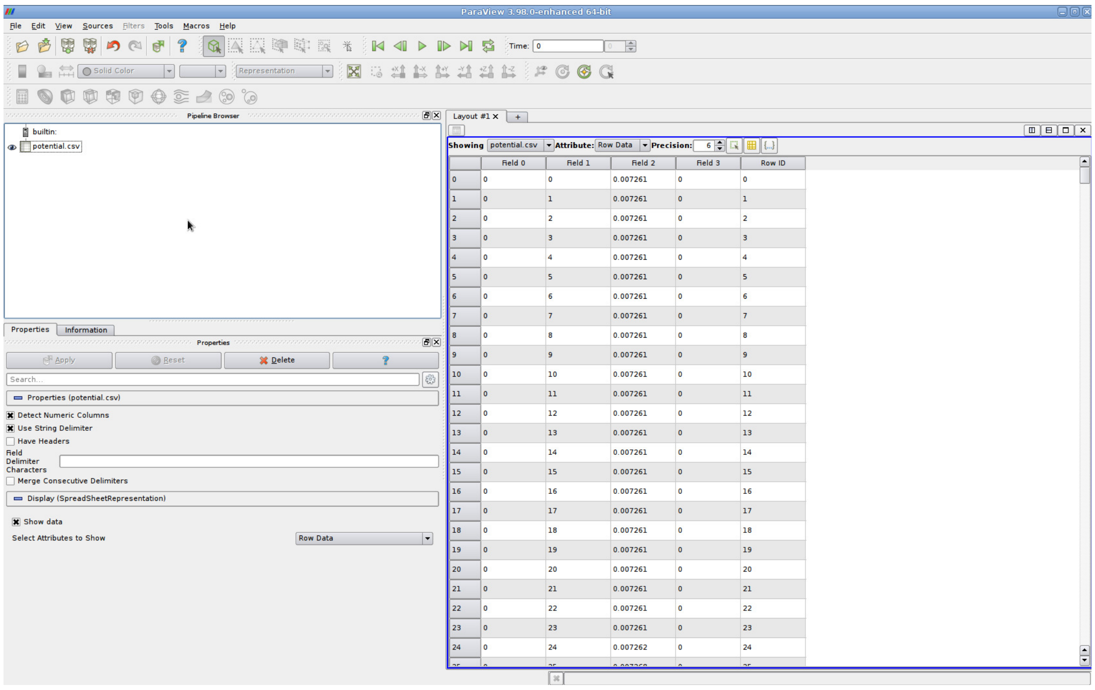


Properties (potential.csv)

[✓] Detect Numeric Columns
[✓] Use String Delimiter
[✓] Have Headers
Field Delimiter Characters: ,
[ ] Merge Consecutive Delimiters

Display

Figure 5.2: A CSV file has been loaded into ParaView.



Properties (potential.csv)

- [✓] Detect Numeric Columns
- [✓] Use String Delimiter
- [ ] Have Headers
- Field Delimiter Characters:
- [ ] Merge Consecutive Delimiters

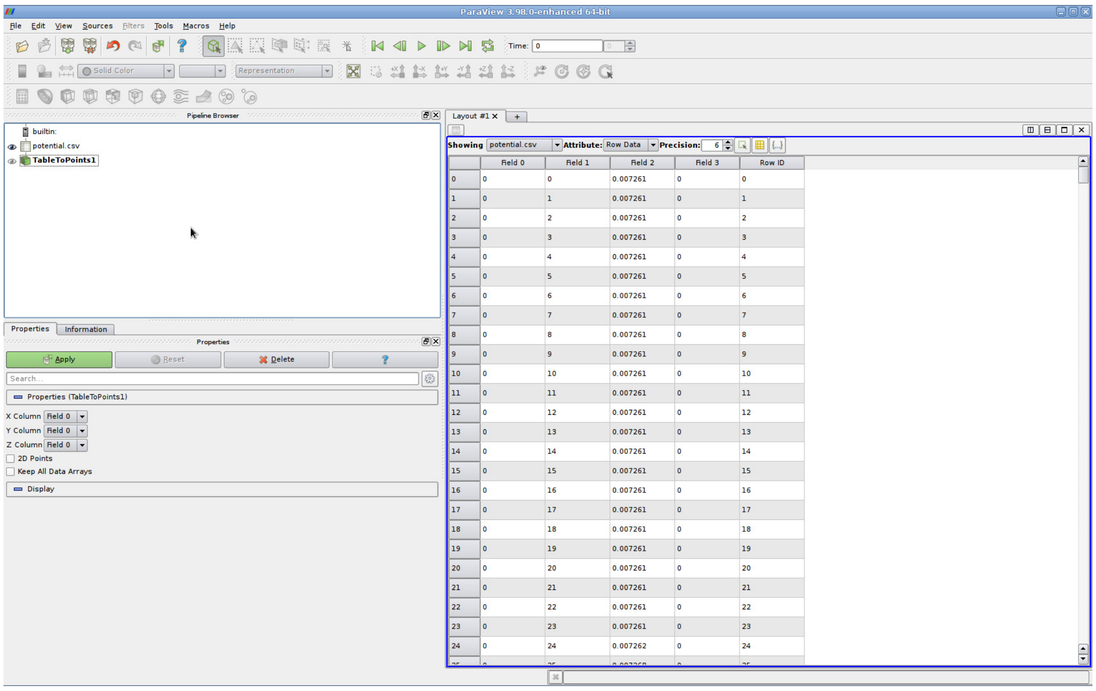
Display (SpreadSheetRepresentation)

- [✓] Show data
- Select Attributes to Show: Row Data

Showing potential.csv Attribute: Row Data Precision: 6

	Field 0	Field 1	Field 2	Field 3	Row ID
0	0	0	0.007261	0	0
1	0	1	0.007261	0	1
2	0	2	0.007261	0	2
3	0	3	0.007261	0	3
4	0	4	0.007261	0	4
5	0	5	0.007261	0	5
6	0	6	0.007261	0	6
7	0	7	0.007261	0	7
8	0	8	0.007261	0	8
9	0	9	0.007261	0	9
10	0	10	0.007261	0	10
11	0	11	0.007261	0	11
12	0	12	0.007261	0	12
13	0	13	0.007261	0	13
14	0	14	0.007261	0	14
15	0	15	0.007261	0	15
16	0	16	0.007261	0	16
17	0	17	0.007261	0	17
18	0	18	0.007261	0	18
19	0	19	0.007261	0	19
20	0	20	0.007261	0	20
21	0	21	0.007261	0	21
22	0	22	0.007261	0	22
23	0	23	0.007261	0	23
24	0	24	0.007262	0	24

Figure 5.3: Headers are deactivated, as well as the comma has been replaced with a whitespace in the delimiter field. After applying the changes, the table on the right is populated.



Properties (TableToPoints1)

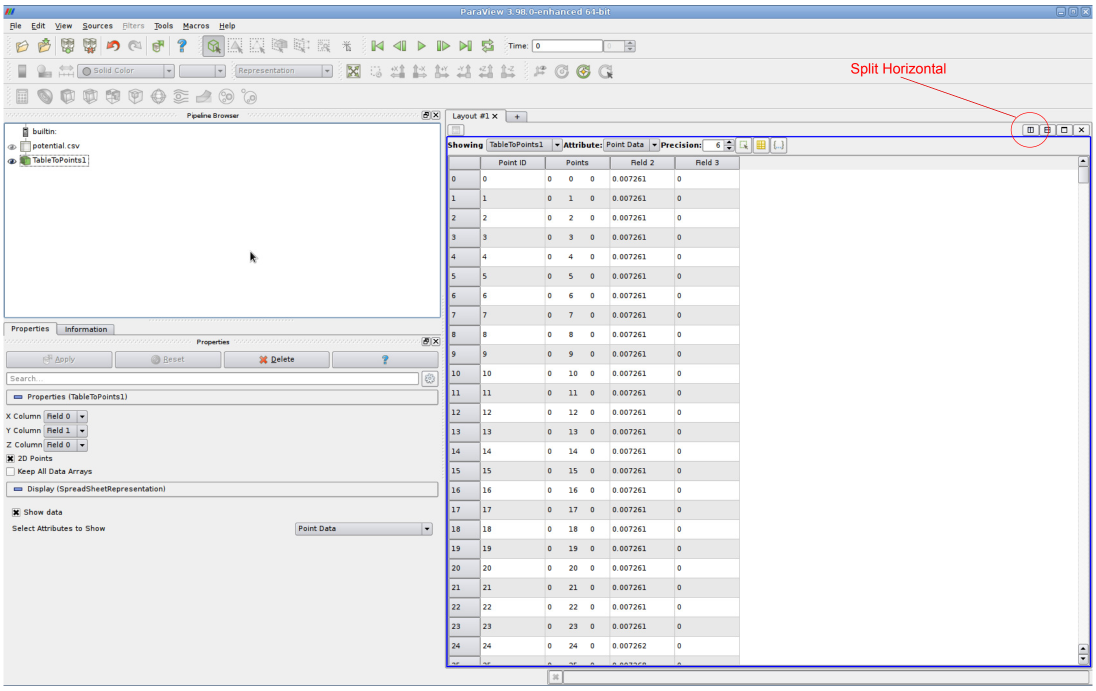
- X Column: Field 0
- Y Column: Field 0
- Z Column: Field 0
- [ ] 2D Points
- [ ] Keep All Data Arrays

Display

Showing potential.csv Attribute: Row Data Precision: 6

	Field 0	Field 1	Field 2	Field 3	Row ID
0	0	0	0.007261	0	0
1	0	1	0.007261	0	1
2	0	2	0.007261	0	2
3	0	3	0.007261	0	3
4	0	4	0.007261	0	4
5	0	5	0.007261	0	5
6	0	6	0.007261	0	6
7	0	7	0.007261	0	7
8	0	8	0.007261	0	8
9	0	9	0.007261	0	9
10	0	10	0.007261	0	10
11	0	11	0.007261	0	11
12	0	12	0.007261	0	12
13	0	13	0.007261	0	13
14	0	14	0.007261	0	14
15	0	15	0.007261	0	15
16	0	16	0.007261	0	16
17	0	17	0.007261	0	17
18	0	18	0.007261	0	18
19	0	19	0.007261	0	19
20	0	20	0.007261	0	20
21	0	21	0.007261	0	21
22	0	22	0.007261	0	22
23	0	23	0.007261	0	23
24	0	24	0.007262	0	24

Figure 5.4: The `TableToPoints` filter has been added loaded.

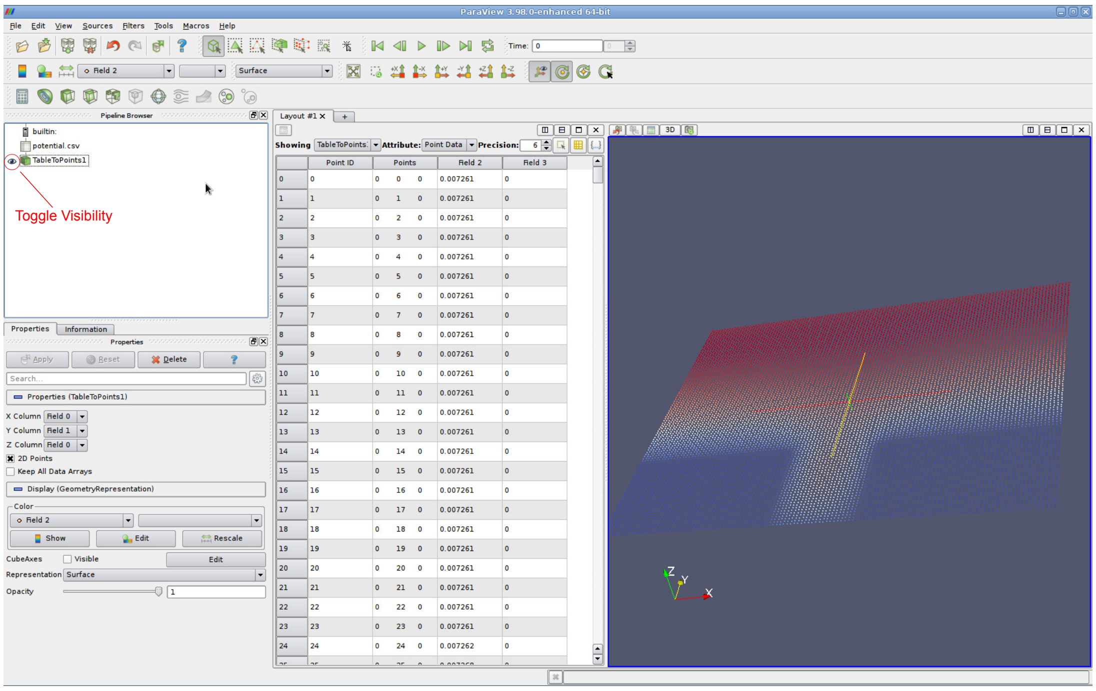


Properties (TableToPoints1)
X Column: Field 0
Y Column: Field 1
Z Column: Field 0
[✓ 2D Points]
Keep All Data Arrays
Display (SpreadSheetRepresentation)
[✓ Show data]
Select Attributes to Show: Point Data

Showing: TableToPoints1 Attribute: Point Data Precision: 6

	Point ID	Points			Field 2	Field 3
0	0	0	0	0	0.007261	0
1	1	0	1	0	0.007261	0
2	2	0	2	0	0.007261	0
3	3	0	3	0	0.007261	0
4	4	0	4	0	0.007261	0
5	5	0	5	0	0.007261	0
6	6	0	6	0	0.007261	0
7	7	0	7	0	0.007261	0
8	8	0	8	0	0.007261	0
9	9	0	9	0	0.007261	0
10	10	0	10	0	0.007261	0
11	11	0	11	0	0.007261	0
12	12	0	12	0	0.007261	0
13	13	0	13	0	0.007261	0
14	14	0	14	0	0.007261	0
15	15	0	15	0	0.007261	0
16	16	0	16	0	0.007261	0
17	17	0	17	0	0.007261	0
18	18	0	18	0	0.007261	0
19	19	0	19	0	0.007261	0
20	20	0	20	0	0.007261	0
21	21	0	21	0	0.007261	0
22	22	0	22	0	0.007261	0
23	23	0	23	0	0.007261	0
24	24	0	24	0	0.007262	0

Figure 5.5: The `TableToPoints` filter has been configured and applied. Note that the X and Y column has been set to field 0 and 1, respectively. Furthermore, the table is set to represent 2D data. Note that the Z column is therefore obsolete. A new 3D render view has to be created for visualizing the data in the subsequent steps. This can be achieved, by pushing the `Split Horizontal` button.

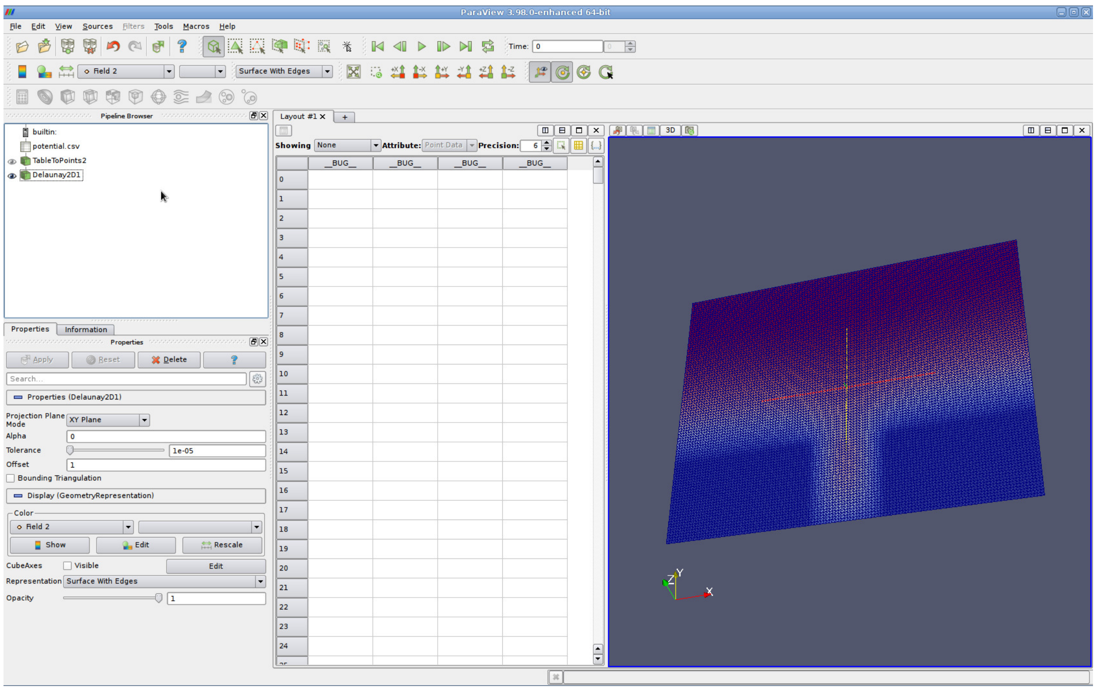


Properties (TableToPoints1)
X Column: Field 0
Y Column: Field 1
Z Column: Field 0
[✓ 2D Points]
Keep All Data Arrays
Display (GeometryRepresentation)
Color: Field 2
CubeAxes: Visible
Representation: Surface
Opacity: 1

Showing: TableToPoints1 Attribute: Point Data Precision: 6

	Point ID	Points			Field 2	Field 3
0	0	0	0	0	0.007261	0
1	1	0	1	0	0.007261	0
2	2	0	2	0	0.007261	0
3	3	0	3	0	0.007261	0
4	4	0	4	0	0.007261	0
5	5	0	5	0	0.007261	0
6	6	0	6	0	0.007261	0
7	7	0	7	0	0.007261	0
8	8	0	8	0	0.007261	0
9	9	0	9	0	0.007261	0
10	10	0	10	0	0.007261	0
11	11	0	11	0	0.007261	0
12	12	0	12	0	0.007261	0
13	13	0	13	0	0.007261	0
14	14	0	14	0	0.007261	0
15	15	0	15	0	0.007261	0
16	16	0	16	0	0.007261	0
17	17	0	17	0	0.007261	0
18	18	0	18	0	0.007261	0
19	19	0	19	0	0.007261	0
20	20	0	20	0	0.007261	0
21	21	0	21	0	0.007261	0
22	22	0	22	0	0.007261	0
23	23	0	23	0	0.007261	0
24	24	0	24	0	0.007262	0

Figure 5.6: The data can be actually visualized by activating the data via hitting the 'greyed out eye' next to the `TableToPoints` filter on the left. Coloring the output can be done by changeing from solid color to a data field, for instance, `Field 2`.

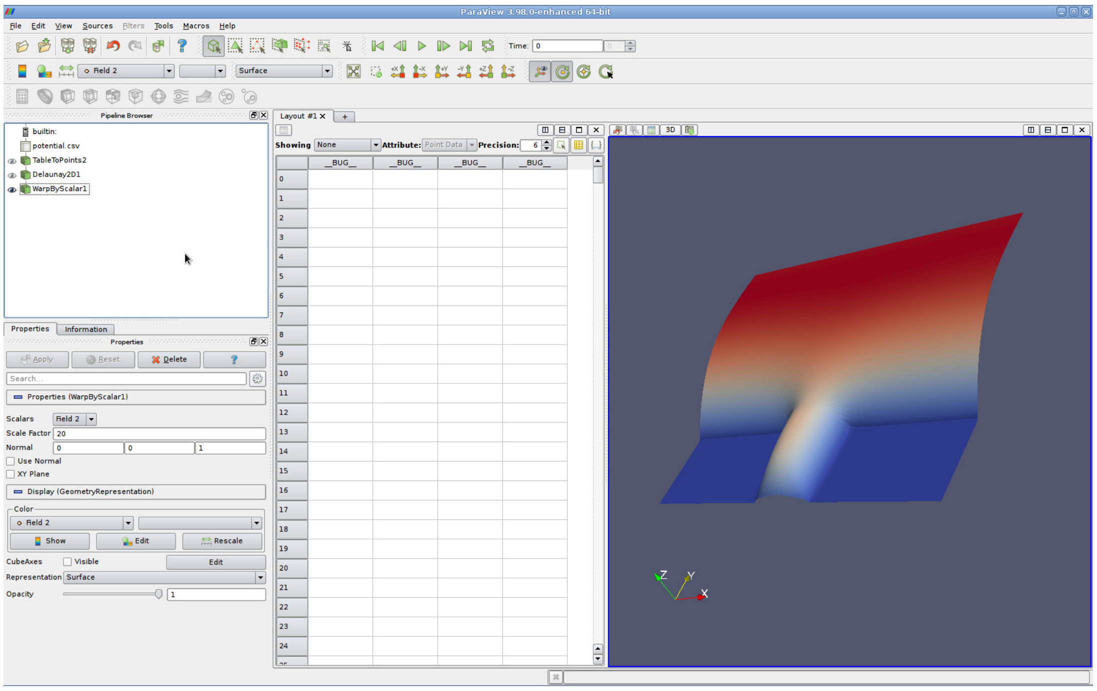


Pipeline Browser: builtin:, potential.csv, TableToPoints2, Delaunay2D1

Properties (Delaunay2D1)
Projection Plane Mode: XY Plane
Alpha: 0
Tolerance: 1e-05
Offset: 1
Bounding Triangulation
Display (GeometryRepresentation)
Color: Field 2
CubeAxes: Visible
Representation: Surface With Edges
Opacity: 1

Showing: None; Attribute: Point Data; Precision: 6

Figure 5.7: So far only points have been visualized, similar to particles. In case a mesh is required for visualization, the `Delaunay2D` filter can be used. No configurations are required, simply adding and applying produces a 2D mesh.



Pipeline Browser: builtin:, potential.csv, TableToPoints2, Delaunay2D1, WarpByScalar1

Properties (WarpByScalar1)
Scalars: Field 2
Scale Factor: 20
Normal: 0 0 1
Use Normal
XY Plane
Display (GeometryRepresentation)
Color: Field 2
CubeAxes: Visible
Representation: Surface
Opacity: 1

Showing: None; Attribute: Point Data; Precision: 6

Figure 5.8: For improving the visualization experience of 2D data, the quantity magnitude can be correlated with a height or Z-coordinate. For this purpose, the `WarpByScalar` filter can be used. By changeing the `Scale Factor` to a higher number, increased scaling is achieved.

Bibliography

[1] "CMake." URL: http://www.cmake.org/

[2] M. Nedjalkov and D. Vasileska, "Semi-Discrete 2D Wigner-Particle Approach," *Journal of Computational Electronics*, vol. 7, no. 3, pp. 222–225, 2008. URL: http://www.iue.tuwien.ac.at/pdf/ib_2008/JB2008_Nedjalkov_2.pdf. DOI: 10.1007/s10825-008-0197-3

[3] M. Nedjalkov, P. Schwaha, S. Selberherr, J. M. Sellier, and D. Vasileska, "Wigner Quasi-Particle Attributes – An Asymptotic Perspective," *Applied Physics Letters*, vol. 102, no. 16, pp. 163 113–1–163 113–4, 2013. URL: http://www.iue.tuwien.ac.at/pdf/ib_2012/JB2013_Nedjalkov_1.pdf. DOI: 10.1063/1.4802931

[4] P. Ellinghaus, J. Weinbub, M. Nedjalkov, S. Selberherr, and I. Dimov, "Distributed-Memory Parallelization of the Wigner Monte Carlo Method Using Spatial Domain Decomposition," *Journal of Computational Electronics*, vol. 14, no. 1, pp. 151–162, 2015. URL: http://www.iue.tuwien.ac.at/pdf/ib_2014/JB2015_Ellinghaus_1.pdf. DOI: 10.1007/s10825-014-0635-3

[5] J. Weinbub, P. Ellinghaus, and M. Nedjalkov, "Domain Decomposition Strategies for the Two-Dimensional Wigner Monte Carlo Method," *Journal of Computational Electronics*, vol. 14, no. 4, pp. 922–929, 2015. URL: http://www.iue.tuwien.ac.at/pdf/ib_2015/JB2015_weinbub_1.pdf. DOI: 10.1007/s10825-015-0730-0

[6] "GNU GCC." URL: https://gcc.gnu.org/

[7] "GSL." URL: http://www.gnu.org/software/gsl/

[8] "FFTW." URL: http://www.fftw.org/

[9] "Lua." URL: http://www.lua.org/

[10] "OpenMPI." URL: http://www.open-mpi.org/

[11] "Gnuplot." URL: http://www.gnuplot.info/

[12] "Python." URL: https://www.python.org/

[13] P. Schwaha, M. Nedjalkov, S. Selberherr, and I. Dimov, "Monte Carlo Investigations of Electron Decoherence due to Phonons," in *Abstracts of the IMACS Seminar on Monte Carlo Methods (MCM)*, 2011, p. 48.

[14] “WIENS.” URL: https://github.com/wiens/wiens/

[15] “The Boost C++ Libraries.” URL: http://www.boost.org/

[16] “Paraview.” URL: http://www.paraview.org/

[17] “MATLAB.” URL: http://www.mathworks.com/products/matlab/