Pulse shape simulation for germanium detectors - SolidStateDetectors.jl

Felix Hagemann

PIRE GEMADARC Summer School
June 22nd, 2022

PIRE
GEMADARC

MAX-PLANCK-INSTITUT
FÜR PHYSIK

Example Detector Simulation Setup

13:00

https://indico.cern.ch/event/1142628/timetable/#20220622

Digital processing of HPGe signals

13:30 - 15:00

14:00

15:00

HPGe detector pulse shape simulation *Felix Hagemann*

BEGe.yaml
SSD_LiveDemonstration.ipynb

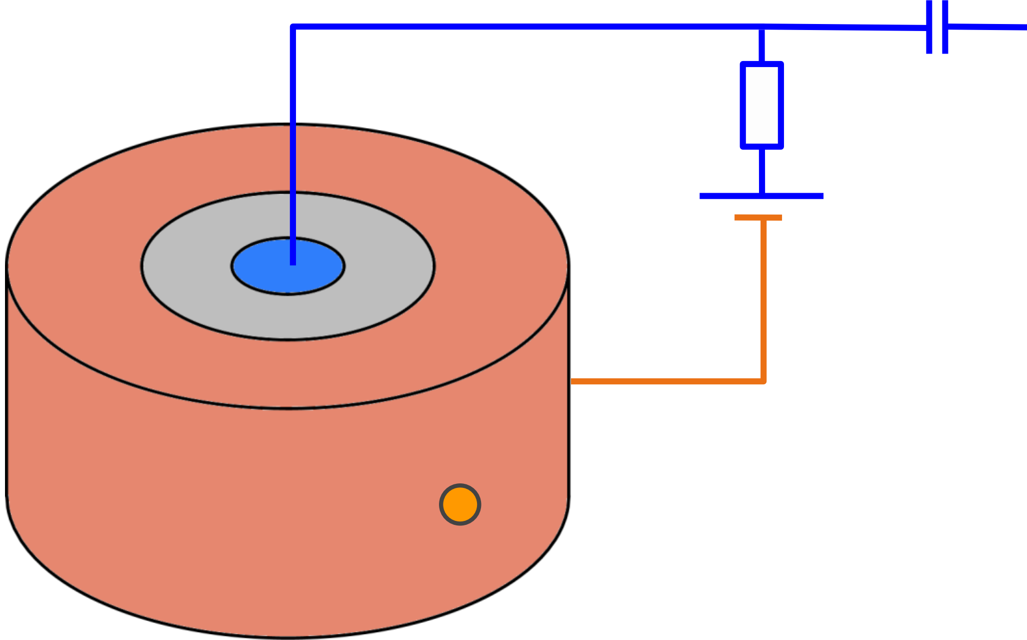
16:00

15:30 - 17:00

17:00

MAX-PLANCK-INSTITUT
FÜR PHYSIK

Example Detector Simulation Setup



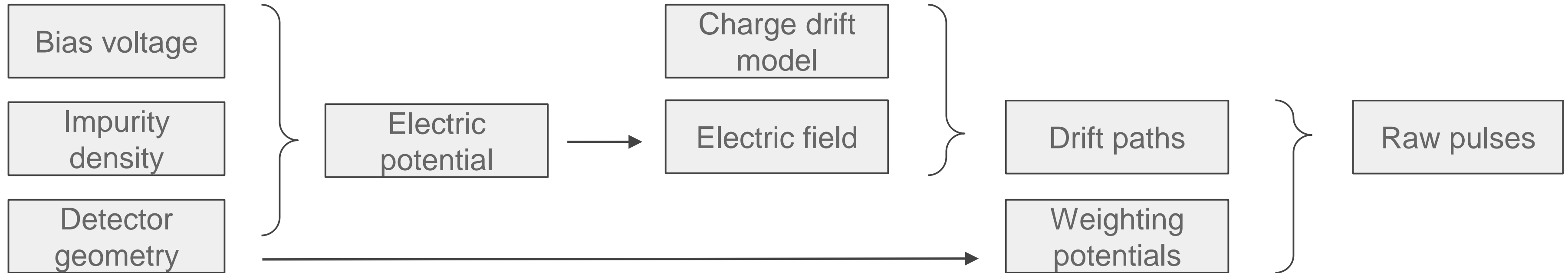
MAX-PLANCK-INSTITUT
FÜR PHYSIK

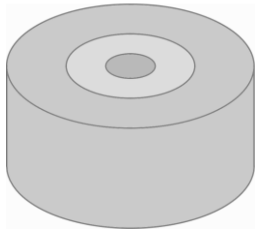
SolidStateDetectors.jl

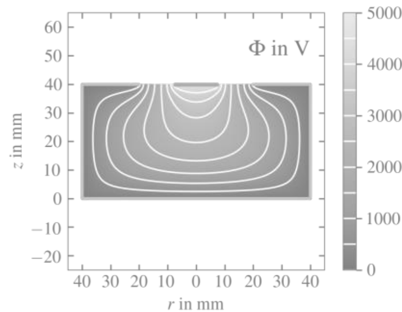
- Open-source simulation software package, written in julia
- 3D calculation of electric potentials and electric fields
- Can simulate arbitrary geometries, e.g. segmented detectors
- Documentation: https://juliaphysics.github.io/SolidStateDetectors.jl/stable/
- Fast field calculation: SIMD on CPU, also supports GPU calculation
- Calculation of capacitance matrix
- Simulation of fields in undepleted detectors ⇒ C-V curves
- Experimental features: diffusion and self-repulsion of charge clouds

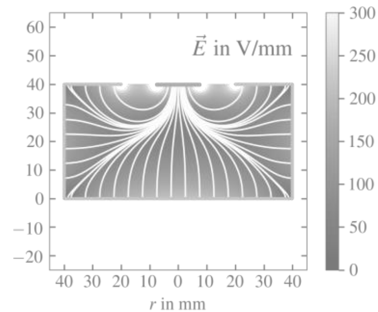
MAX-PLANCK-INSTITUT FÜR PHYSIK

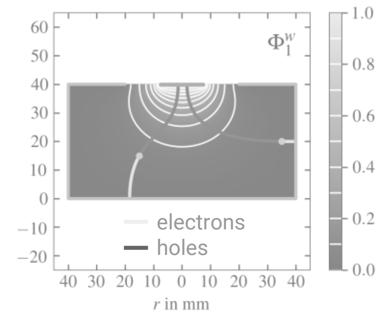
Pulse Shape Simulation Chain

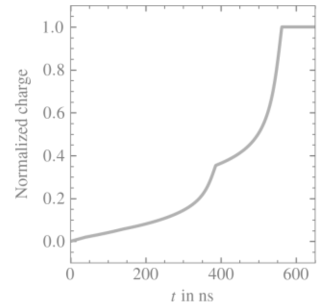












MAX-PLANCK-INSTITUT
FÜR PHYSIK

Electric Potential Calculation

1. Maxwell equation:

$$\nabla \cdot \mathbf{D}(\mathbf{r}) = \rho(\mathbf{r})$$

$$\mathbf{D}(\mathbf{r}) = \epsilon_0 \epsilon_r(\mathbf{r}) \mathbf{E}(\mathbf{r})$$

$$\mathbf{E}(\mathbf{r}) = -\nabla \Phi(\mathbf{r})$$

$$\nabla \cdot (\epsilon_r(\mathbf{r}) \cdot \nabla \Phi(\mathbf{r})) = -\frac{\rho(\mathbf{r})}{\epsilon_0}$$

Electric potential

Required input:

- charge density $\rho(\mathbf{r})$,
- dielectric distribution $\epsilon_r(\mathbf{r})$,
- boundary conditions for $\Phi(\mathbf{r})$

Impurity density

Bias voltage

Detector geometry

SSD solves this numerically

- Successive Over-Relaxation (SOR) algorithm
- Red-Black division of the grid → parallelization (CPU vectorization, GPU support)
- Adaptive grid

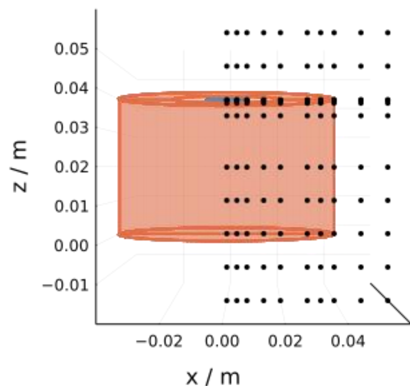
MAX-PLANCK-INSTITUT FÜR PHYSIK

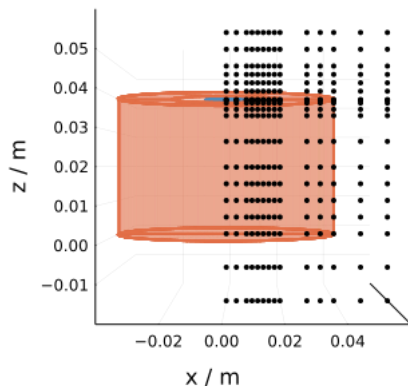
Electric Potential Calculation

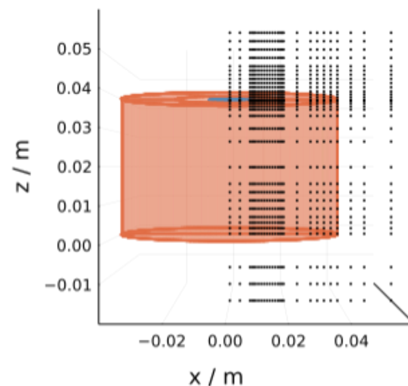
Electric potential

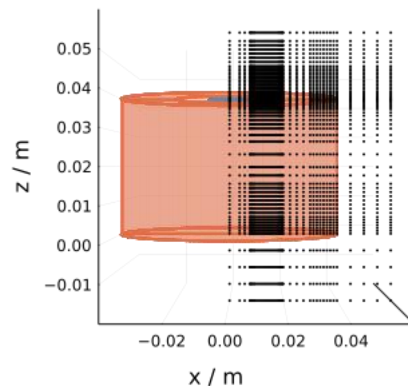
Numerical approach: Divide your world (detector + surroundings) into small parts and calculate for each part (grid point) its potential

Adaptive grid: Start with a coarse grid (10 x 10 x 10 points) and become finer (eg. 200 x 200 x 200 points)









MAX-PLANCK-INSTITUT FÜR PHYSIK

Electric Potential Calculation

How to calculate the potential of a single grid point?

$$\nabla \cdot (\epsilon_r(\mathbf{r}) \cdot \nabla \Phi(\mathbf{r})) = -\frac{\rho(\mathbf{r})}{\epsilon_0}$$

Integral form:

$$\iiint_V \nabla \cdot (\epsilon_r(\mathbf{r}) \cdot \nabla \Phi(\mathbf{r})) \; dV = \iiint_V -\frac{\rho(\mathbf{r})}{\epsilon_0} \, dV$$

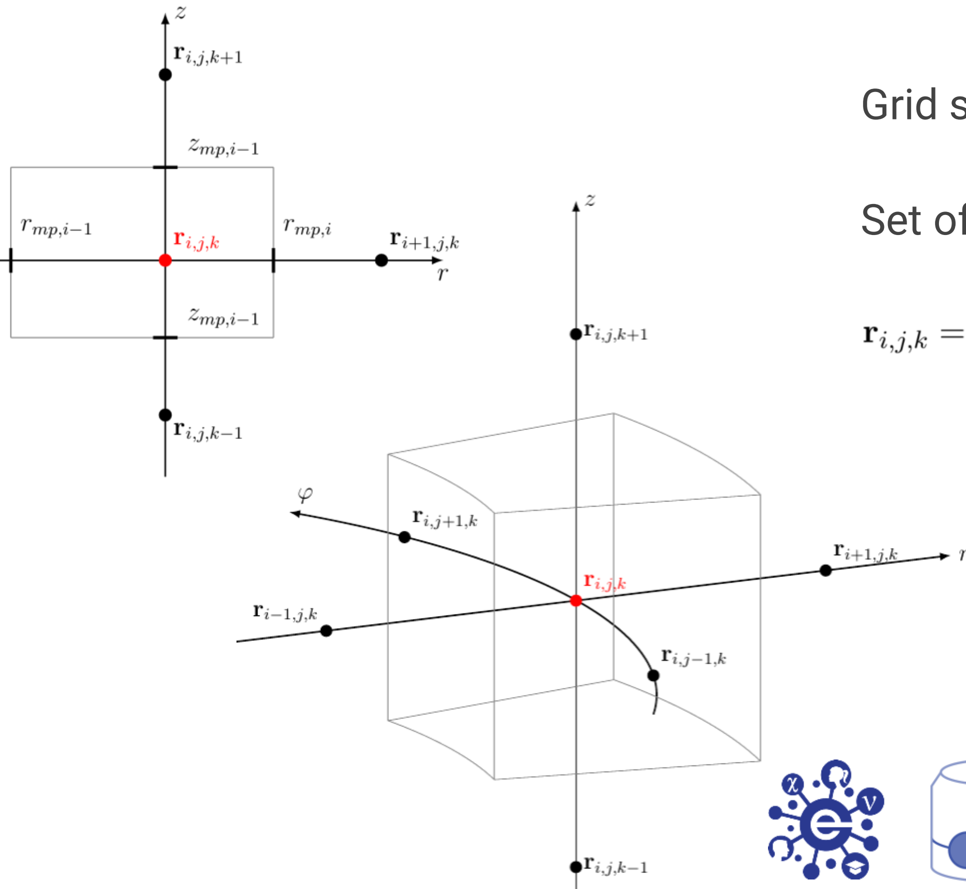
Divergence theorem:

$$\oiint_S (\epsilon_r(\mathbf{r}) \cdot \nabla \Phi(\mathbf{r})) \cdot d\mathbf{S} = -\iiint_V \frac{\rho(\mathbf{r})}{\epsilon_0} \, dV$$

MAX-PLANCK-INSTITUT FÜR PHYSIK

Electric Potential Calculation

Electric potential



Grid size: $N_r \times N_\varphi \times N_z$ = N grid points

Set of grid points:

$$\mathbf{r}_{i,j,k} = \begin{pmatrix} r_i \\ \varphi_j \\ z_k \end{pmatrix} \qquad i \in 1,\ldots,N_r;\; j \in 1,\ldots,N_\varphi;\; k \in 1,\ldots,N_z$$

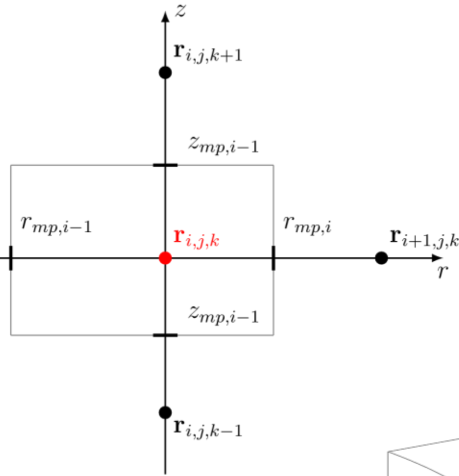
Mid points :
(points between actual grid points)

$$\begin{aligned} r_{mp,i} &= r_i + 0.5 \cdot (r_{i+1} - r_i) \\ \varphi_{mp,j} &= \varphi_j + 0.5 \cdot (\varphi_{j+1} - \varphi_j) \\ z_{mp,k} &= z_k + 0.5 \cdot (z_{k+1} - z_k) \end{aligned}$$

MAX-PLANCK-INSTITUT FÜR PHYSIK

Solid State Detectors

Electric Potential Calculation

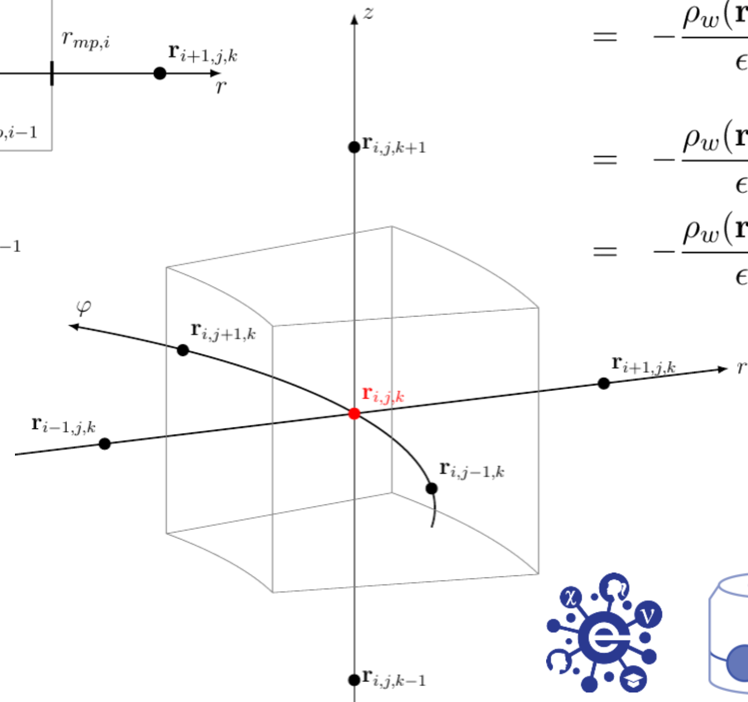


$$-\iiint_V \frac{\rho(\mathbf{r})}{\epsilon_0}\,dV = -\int_{r_{mp,i-1}}^{r_{mp,i}}\int_{\varphi_{mp,j-1}}^{\varphi_{mp,j}}\int_{z_{mp,k-1}}^{z_{mp,k}} r\cdot\frac{\rho(\mathbf{r})}{\epsilon_0}\,dz\,d\varphi\,dr$$

$$= -\frac{\rho_w(\mathbf{r}_{i,j,k})}{\epsilon_0}\int_{r_{mp,i-1}}^{r_{mp,i}}\int_{\varphi_{mp,j-1}}^{\varphi_{mp,j}}\int_{z_{mp,k-1}}^{z_{mp,k}} r\,dz\,d\varphi\,dr$$

$$= -\frac{\rho_w(\mathbf{r}_{i,j,k})}{\epsilon_0}\cdot\frac{1}{2}(r^2_{mp,i}-r^2_{mp,i-1})(\varphi_{mp,j}-\varphi_{mp,j-1})(z_{mp,k}-z_{mp,k-1})$$

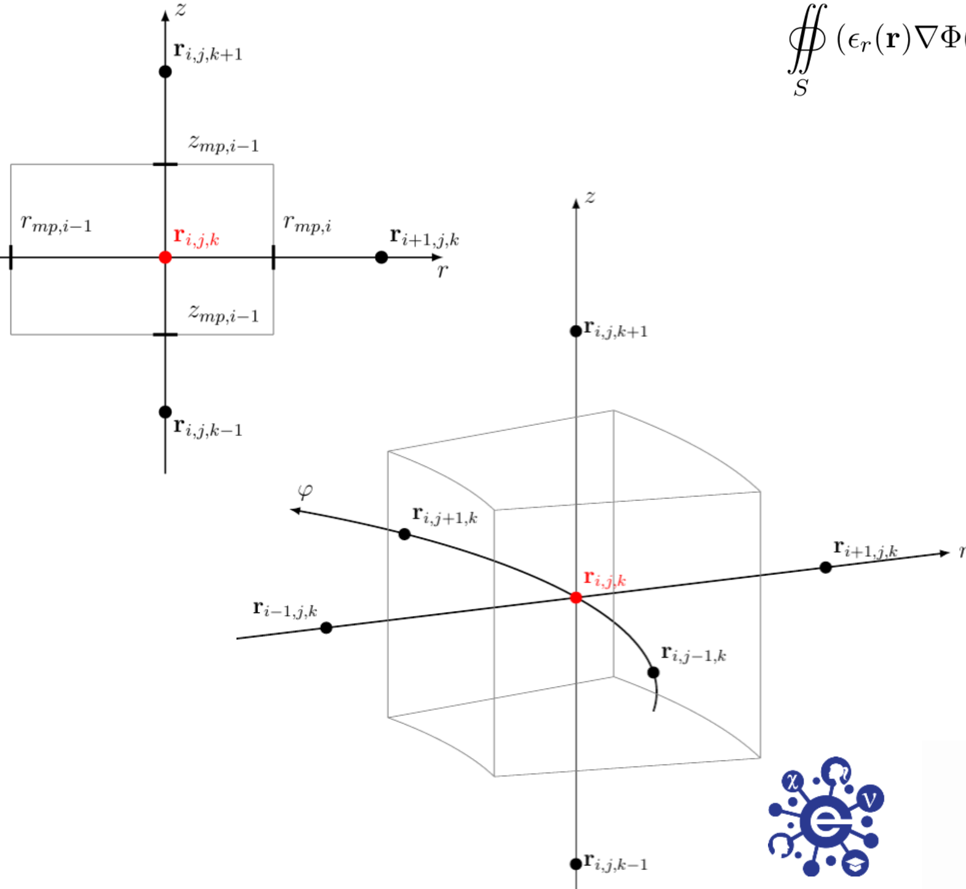
$$= -\frac{\rho_w(\mathbf{r}_{i,j,k})}{\epsilon_0}\cdot V_{i,j,k} = Q^{eff}_{i,j,k}$$



MAX-PLANCK-INSTITUT FÜR PHYSIK

Electric Potential Calculation

Electric potential



$$\oiint_S (\epsilon_r(\mathbf{r})\nabla\Phi(\mathbf{r}))\cdot d\mathbf{S} = \iint_{r^+} + \iint_{r^-} + \iint_{\varphi^+} + \iint_{\varphi^-} + \iint_{z^+} + \iint_{z^-}$$

$$\iint_{r^+} = \int_{z_{mp,k-1}}^{z_{mp,k}} \int_{\varphi_{mp,j-1}}^{\varphi_{mp,j}} -\epsilon_r(\mathbf{r})\left(\nabla\Phi(\mathbf{r})\right) r_{mp,i}\,\mathbf{e}_r\,d\varphi\,dz$$

$$\iint_{r^-} = \int_{z_{mp,k-1}}^{z_{mp,k}} \int_{\varphi_{mp,j-1}}^{\varphi_{mp,j}} +\epsilon_r(\mathbf{r})\left(\nabla\Phi(\mathbf{r})\right) r_{mp,i+1}\,\mathbf{e}_r\,d\varphi\,dz$$

$$\iint_{\varphi^+} = \int_{r_{mp,i-1}}^{r_{mp,i}} \int_{z_{mp,k-1}}^{z_{mp,k}} -\epsilon_r(\mathbf{r})\left(\nabla\Phi(\mathbf{r})\right)\mathbf{e}_\varphi\,dz\,dr$$

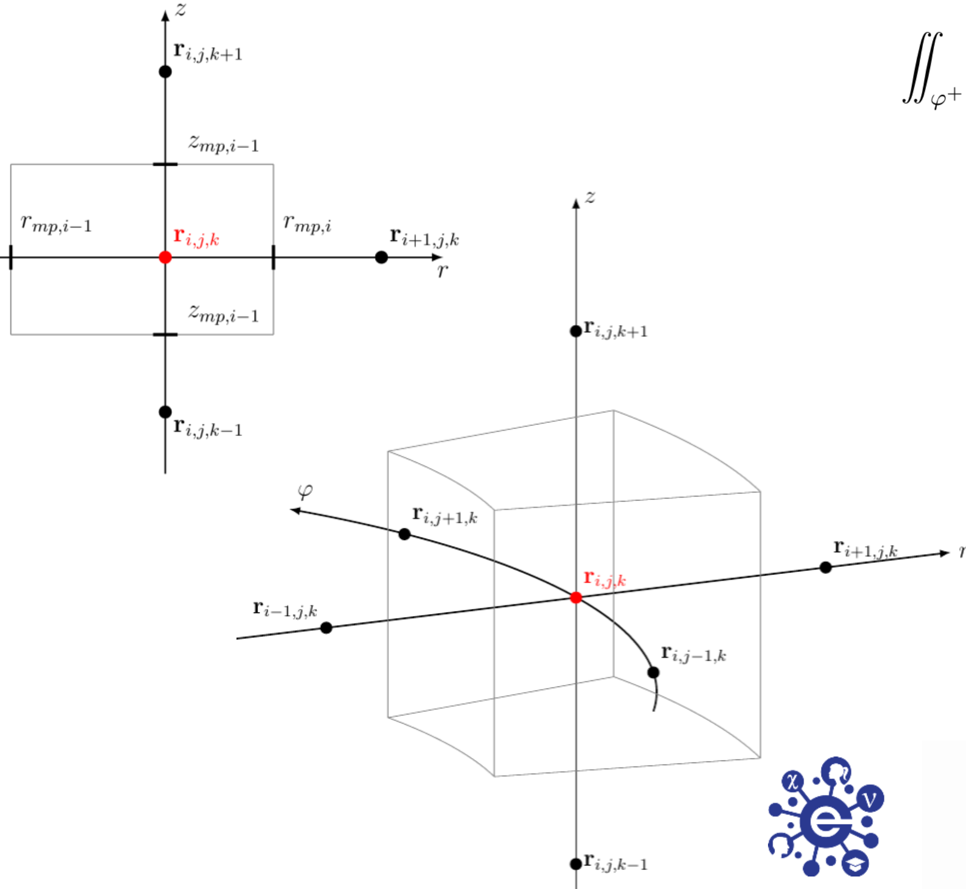
$$\iint_{\varphi^-} = \int_{r_{mp,i-1}}^{r_{mp,i}} \int_{z_{mp,k-1}}^{z_{mp,k}} +\epsilon_r(\mathbf{r})\left(\nabla\Phi(\mathbf{r})\right)\mathbf{e}_\varphi\,dz\,dr$$

$$\iint_{z^+} = \int_{r_{mp,i-1}}^{r_{mp,i}} \int_{\varphi_{mp,j-1}}^{\varphi_{mp,j}} -\epsilon_r(\mathbf{r})\left(\nabla\Phi(\mathbf{r})\right) r\,\mathbf{e}_z\,d\varphi\,dr$$

$$\iint_{z^-} = \int_{r_{mp,i-1}}^{r_{mp,i}} \int_{\varphi_{mp,j-1}}^{\varphi_{mp,j}} +\epsilon_r(\mathbf{r})\left(\nabla\Phi(\mathbf{r})\right) r\,\mathbf{e}_z\,d\varphi\,dr\;.$$

Electric Potential Calculation

Electric potential



$$
\begin{aligned}
\iint_{\varphi^+} &= \int_{r_{mp,i-1}}^{r_{mp,i}} \int_{z_{mp,k-1}}^{z_{mp,k}} +\epsilon_r(\mathbf{r})\,(\nabla\Phi(\mathbf{r}))\;\mathbf{e}_\varphi\,dz\,dr \\
&= \int_{r_{mp,i-1}}^{r_{mp,i}} \int_{z_{mp,k-1}}^{z_{mp,k}} \epsilon_r(\mathbf{r})\frac{1}{r_i}\frac{\partial}{\partial\varphi}\Phi(\mathbf{r})\,dz\,dr \\
&= \int_{r_{mp,i-1}}^{r_{mp,i}} \int_{z_{mp,k-1}}^{z_{mp,k}} \epsilon_r(\mathbf{r})\frac{\Phi_{i,j+1,k}-\Phi_{i,j,k}}{r_i\cdot(\varphi_{j+1}-\varphi_j)}\,dz\,dr \\
&= \frac{\Phi_{i,j+1,k}-\Phi_{i,j,k}}{r_i\cdot(\varphi_{j+1}-\varphi_j)}\cdot\int_{r_{mp,i-1}}^{r_{mp,i}} \int_{z_{mp,k-1}}^{z_{mp,k}} \epsilon_r(\mathbf{r})\,dz\,dr \\
&= \frac{\Phi_{i,j+1,k}-\Phi_{i,j,k}}{r_i\cdot(\varphi_{j+1}-\varphi_j)}\cdot\epsilon_{i,j,k}^{w,\varphi^+}\cdot\int_{r_{mp,i-1}}^{r_{mp,i}} \int_{z_{mp,k-1}}^{z_{mp,k}} dz\,dr \\
&= \frac{\Phi_{i,j+1,k}-\Phi_{i,j,k}}{r_i\cdot(\varphi_{j+1}-\varphi_j)}\cdot\epsilon_{i,j,k}^{w,\varphi^+}\cdot(r_{mp,i}-r_{mp,i-1})(z_{mp,k}-z_{mp,k-1}) \\
&= \frac{\Phi_{i,j+1,k}-\Phi_{i,j,k}}{r_i\cdot(\varphi_{j+1}-\varphi_j)}\cdot\epsilon_{i,j,k}^{w,\varphi^+}\cdot A_{i,j,k}^{\varphi^+}
\end{aligned}
$$

Electric Potential Calculation

Electric potential

$$\oiint_S (\epsilon_r(\mathbf{r}) \nabla \Phi(\mathbf{r})) \cdot d\mathbf{S} = + \iint_{r^+} + \iint_{r^-} + \iint_{\varphi^+} + \iint_{\varphi^-} + \iint_{z^+} + \iint_{z^-}$$

$$\begin{aligned} = &+ \frac{\Phi_{i+1,j,k} - \Phi_{i,j,k}}{r_{i+1} - r_i} \cdot \epsilon^{w,r^+}_{i,j,k} \cdot A^{r^+}_{i,j,k} \\ &- \frac{\Phi_{i,j,k} - \Phi_{i-1,j,k}}{r_i - r_{i-1}} \cdot \epsilon^{w,r^-}_{i,j,k} \cdot A^{r^-}_{i,j,k} \\ &+ \frac{\Phi_{i,j+1,k} - \Phi_{i,j,k}}{r_i \cdot (\varphi_{j+1} - \varphi_j)} \cdot \epsilon^{w,\varphi^+}_{i,j,k} \cdot A^{\varphi^+}_{i,j,k} \\ &- \frac{\Phi_{i,j,k} - \Phi_{i,j-1,k}}{r_i \cdot (\varphi_j - \varphi_{j-1})} \cdot \epsilon^{w,\varphi^-}_{i,j,k} \cdot A^{\varphi^-}_{i,j,k} \\ &+ \frac{\Phi_{i,j,k+1} - \Phi_{i,j,k}}{z_{k+1} - z_k} \cdot \epsilon^{w,z^+}_{i,j,k} \cdot A^{z^+}_{i,j,k} \\ &- \frac{\Phi_{i,j,k} - \Phi_{i,j,k-1}}{z_k - z_{k-1}} \cdot \epsilon^{w,z^-}_{i,j,k} \cdot A^{z^z}_{i,j,k} \end{aligned}$$

MAX-PLANCK-INSTITUT
FÜR PHYSIK

Electric Potential Calculation

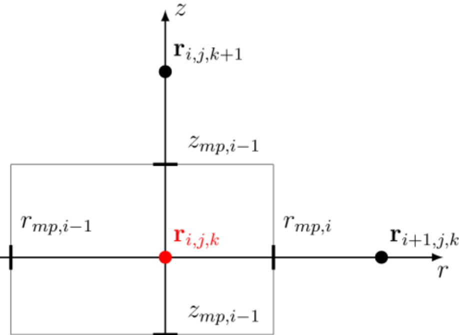
Electric potential

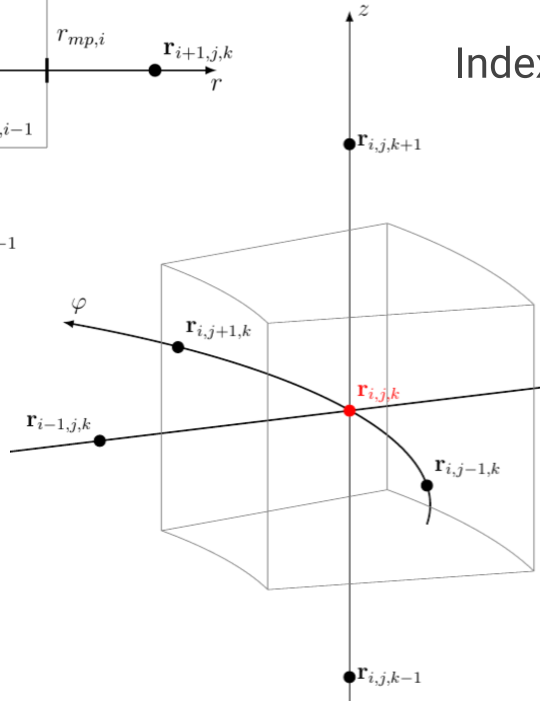
$$
\begin{aligned}
Q^{eff}_{i,j,k} = \; & + \frac{\Phi_{i+1,j,k} - \Phi_{i,j,k}}{r_{i+1} - r_i} \cdot \epsilon^{w,r^+}_{i,j,k} \cdot A^{r^+}_{i,j,k} \\
& - \frac{\Phi_{i,j,k} - \Phi_{i-1,j,k}}{r_i - r_{i-1}} \cdot \epsilon^{w,r^-}_{i,j,k} \cdot A^{r^-}_{i,j,k} \\
& + \frac{\Phi_{i,j+1,k} - \Phi_{i,j,k}}{r_i \cdot (\varphi_{j+1} - \varphi_j)} \cdot \epsilon^{w,\varphi^+}_{i,j,k} \cdot A^{\varphi^+}_{i,j,k} \\
& - \frac{\Phi_{i,j,k} - \Phi_{i,j-1,k}}{r_i \cdot (\varphi_j - \varphi_{j-1})} \cdot \epsilon^{w,\varphi^-}_{i,j,k} \cdot A^{\varphi^-}_{i,j,k} \\
& + \frac{\Phi_{i,j,k+1} - \Phi_{i,j,k}}{z_{k+1} - z_k} \cdot \epsilon^{w,z^+}_{i,j,k} \cdot A^{z^+}_{i,j,k} \\
& - \frac{\Phi_{i,j,k} - \Phi_{i,j,k-1}}{z_k - z_{k-1}} \cdot \epsilon^{w,z^-}_{i,j,k} \cdot A^{z^z}_{i,j,k}
\end{aligned}
$$

MAX-PLANCK-INSTITUT
FÜR PHYSIK

Electric Potential Calculation

Electric potential





$$\Phi_{i,j,k} = a^0_{i,j,k}\,[\,Q^{eff}_{i,j,k} + a^{r^+}_{i,j,k}\cdot\Phi_{i+1,j,k} + a^{r^-}_{i,j,k}\cdot\Phi_{i-1,j,k} + a^{\varphi^+}_{i,j,k}\cdot\Phi_{i,j+1,k} + a^{\varphi^-}_{i,j,k}\cdot\Phi_{i,j-1,k} + a^{z^+}_{i,j,k}\cdot\Phi_{i,j,k+1} + a^{z^-}_{i,j,k}\cdot\Phi_{i,j,k-1}\,]$$

Index change

$$\begin{pmatrix}\Phi_1\\ \Phi_2\\ \vdots\\ \Phi_N\end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,N}\\ a_{2,1} & \ddots & \ddots & \vdots\\ \vdots & \ddots & \ddots & \vdots\\ a_{N,1} & \cdots & \cdots & a_{N,N}\end{pmatrix}^{N\times N} \cdot \begin{pmatrix}\Phi_1\\ \Phi_2\\ \vdots\\ \Phi_N\end{pmatrix} + \begin{pmatrix} a^0_1\,Q^{eff}_1\\ a^0_2\,Q^{eff}_2\\ \vdots\\ a^0_N\,Q^{eff}_N\end{pmatrix}$$

$$\mathbf{\Phi} = \mathbf{A}\cdot\mathbf{\Phi} + \mathbf{Q}$$

System of N linear equations

MAX-PLANCK-INSTITUT FÜR PHYSIK

Electric Potential Calculation

$$\mathbf{\Phi} = \mathbf{A} \cdot \mathbf{\Phi} + \mathbf{Q}$$

System of N linear equations

$$\mathbf{\Phi}^{k+1} = \mathbf{A} \cdot \mathbf{\Phi}^{k} + \mathbf{Q}$$

Gauss-Seidel method

Set initial state: $\mathbf{\Phi}^{0}$

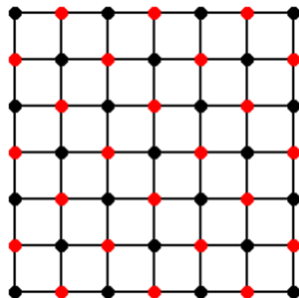
Solve this equation several times, until an equilibrium is reached (until it “converges”, i.e. the potential does not change any more).

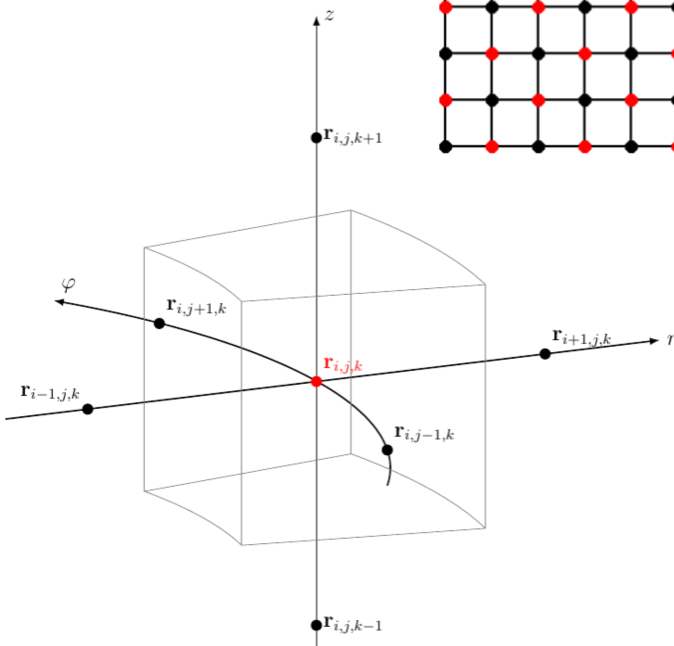
The Successive Over-Relaxation (SOR) method is based on the Gauss-Seidel method, but normally converge much faster to its equilibrium.

MAX-PLANCK-INSTITUT FÜR PHYSIK

Red-Black Algorithm

Electric potential





$$\mathbf{\Phi}^{k+1} = \mathbf{A} \cdot \mathbf{\Phi}^{k} + \mathbf{Q}$$ N equations

Red-Black algorithm (Even / Odd)

$$\mathbf{\Phi}_R^{k+1} = \mathbf{A}_R \cdot \mathbf{\Phi}_B^{k} + \mathbf{Q}_R \quad | \; N/2 \text{ equations}$$

$$\mathbf{\Phi}_B^{k+1} = \mathbf{A}_B \cdot \mathbf{\Phi}_R^{k} + \mathbf{Q}_B \quad | \; N/2 \text{ equations}$$

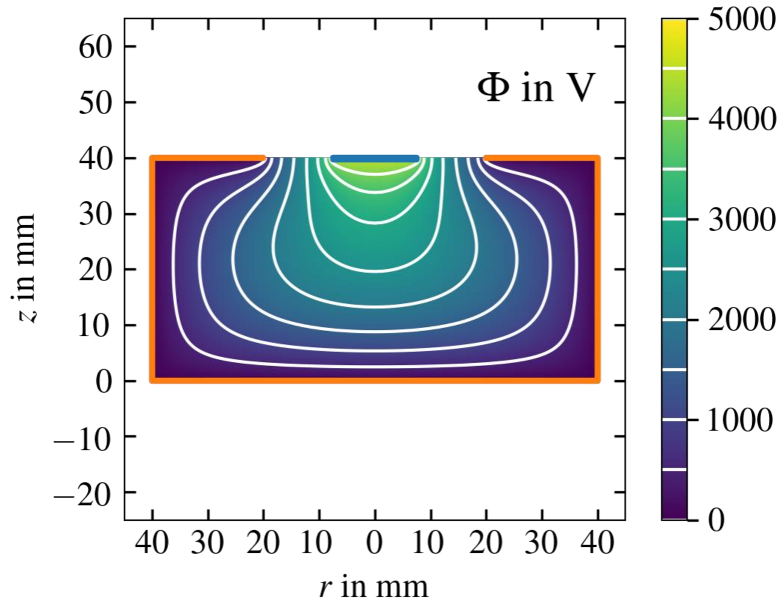
Red (black) points do not depend on values of other red (black) points, so they can be updated simultaneously

MAX-PLANCK-INSTITUT FÜR PHYSIK

Electric Potential Calculation

Electric potential

julia > calculate_electric_potential!(sim)



MAX-PLANCK-INSTITUT
FÜR PHYSIK

Electric Potential Calculation

julia > calculate_electric_potential!(sim)

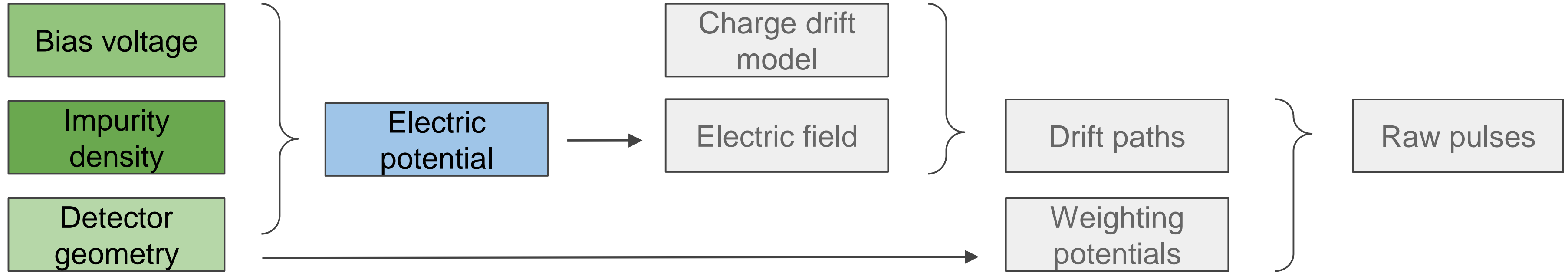
Keywords

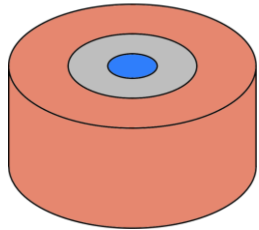
- `convergence_limit::Real`: `convergence_limit` times the bias voltage sets the convergence limit of the relaxation. The convergence value is the absolute maximum difference of the potential between two iterations of all grid points. Default of `convergence_limit` is `1e-7` (times bias voltage).
- `refinement_limits`: Defines the maximum relative (to applied bias voltage) allowed differences of the potential value of neighbored grid points in each dimension for each refinement.
  - `rl::Real` -> One refinement with `rl` equal in all 3 dimensions.
  - `rl::Tuple{<:Real,<:Real,<:Real}` -> One refinement with `rl` set individual for each dimension.
  - `rl::Vector{<:Real}` -> `length(l)` refinements with `rl[i]` being the limit for the i-th refinement.
  - `rl::Vector{<:Real,<:Real,<:Real}}` -> `length(rl)` refinements with `rl[i]` being the limits for the `i`-th refinement.
- `min_tick_distance::Tuple{<:Quantity, <:Quantity, <:Quantity}`: Tuple of the minimum allowed distance between two grid ticks for each dimension. It prevents the refinement to make the grid too fine. Default is `1e-5` for linear axes and `1e-5 / (0.25 * r_max)` for the polar axis in case of a cylindrical `grid`.
- `max_tick_distance::Tuple{<:Quantity, <:Quantity, <:Quantity}`: Tuple of the maximum allowed distance between two grid ticks for each dimension used in the initialization of the grid. Default is 1/4 of size of the world of the respective dimension.
- `max_distance_ratio::Real`: Maximum allowed ratio between the two distances in any dimension to the two neighbouring grid points. If the ratio is too large, additional ticks are generated such that the new ratios are smaller than `max_distance_ratio`. Default is `5`.
- `grid::Grid`: Initial grid used to start the simulation. Default is `Grid(sim)`.
- `depletion_handling::Bool`: Enables the handling of undepleted regions. Default is `false`.
- `use_nthreads::Union{Int, Vector{Int}}`: If `<:Int`, `use_nthreads` defines the maximum number of threads to be used in the computation. Fewer threads might be used depending on the current grid size due to threading overhead. Default is `Base.Threads.nthreads()`. If `<:Vector{Int}`, `use_nthreads[i]` defines the number of threads used for each grid (refinement) stage of the field simulation. The environment variable `JULIA_NUM_THREADS` must be set appropriately before the Julia session was started (e.g. `export JULIA_NUM_THREADS=8` in case of bash).
- `sor_consts::Union{<:Real, NTuple{2, <:Real}}`: Two element tuple in case of cylindrical coordinates. First element contains the SOR constant for `r` = 0. Second contains the constant at the outer most grid point in `r`. A linear scaling is applied in between. First element should be smaller than the second one and both should be `∈ [1.0, 2.0]`. Default is `[1.4, 1.85]`. In case of Cartesian coordinates, only one value is taken.
- `max_n_iterations::Int`: Set the maximum number of iterations which are performed after each grid refinement. Default is `10000`. If set to `-1` there will be no limit.
- `not_only_paint_contacts::Bool = true`: Whether to only use the painting algorithm of the surfaces of `Contact` without checking if points are actually inside them. Setting it to `false` should improve the performance but the points inside of `Contact` are not fixed anymore.
- `paint_contacts::Bool = true`: Enable or disable the painting of the surfaces of the `Contact` onto the `grid`.
- `verbose::Bool=true`: Boolean whether info output is produced or not.

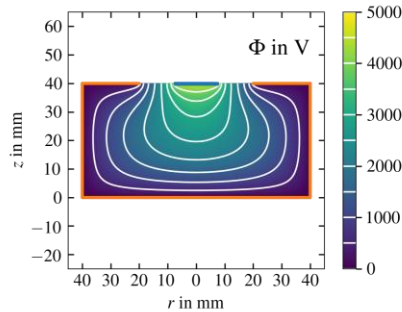
Documentation on GitHub
https://juliaphysics.github.io/SolidStateDetectors.jl/stable/

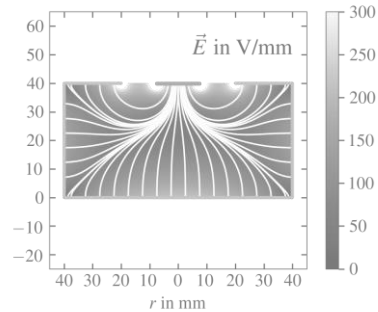
MAX-PLANCK-INSTITUT FÜR PHYSIK

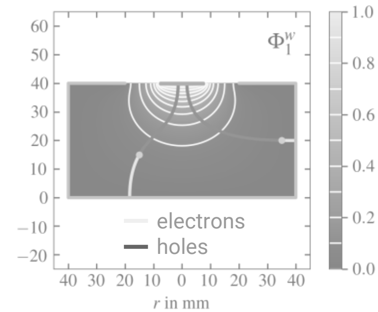
Pulse Shape Simulation Chain

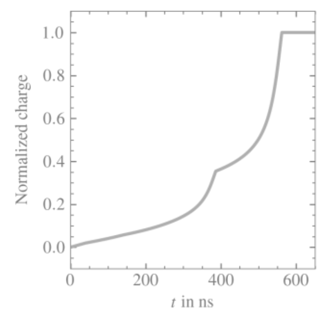












MAX-PLANCK-INSTITUT FÜR PHYSIK

Electric Field Calculation

Electric potential

$$\Phi_{i,j,k}$$

$$\mathbf{E}(\mathbf{r}) = -\nabla\Phi(\mathbf{r})$$

$$\mathbf{E}_{i,j,k} = \left(E_r^{i,j,k}, E_\varphi^{i,j,k}, E_z^{i,j,k}\right)$$

Mean of finite difference:

$$E_r^{i,j,k} = \frac{1}{2}\left(\frac{\Phi_{i+1,j,k}-\Phi_{i,j,k}}{r_{i+1}-r_i} + \frac{\Phi_{i,j,k}-\Phi_{i-1,j,k}}{r_i - r_{i-1}}\right)$$

$$E_\varphi^{i,j,k} = \frac{1}{2}\left(\frac{\Phi_{i,j+1,k}-\Phi_{i,j,k}}{\varphi_{j+1}-\varphi_j} + \frac{\Phi_{i,j,k}-\Phi_{i,j-1,k}}{\varphi_j - \varphi_{j-1}}\right)$$

$$E_z^{i,j,k} = \frac{1}{2}\left(\frac{\Phi_{i,j,k+1}-\Phi_{i,j,k}}{z_{k+1}-z_k} + \frac{\Phi_{i,j,k}-\Phi_{i,j,k-1}}{z_k - z_{k-1}}\right)$$

Electric field

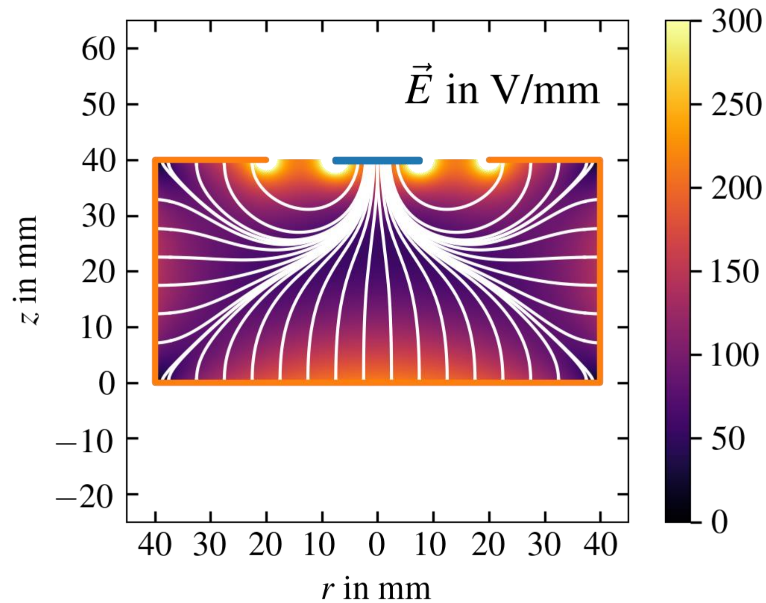
$$\mathbf{E}(\mathbf{r})$$

Electric field at any point **r** (through linear interpolation)

MAX-PLANCK-INSTITUT FÜR PHYSIK

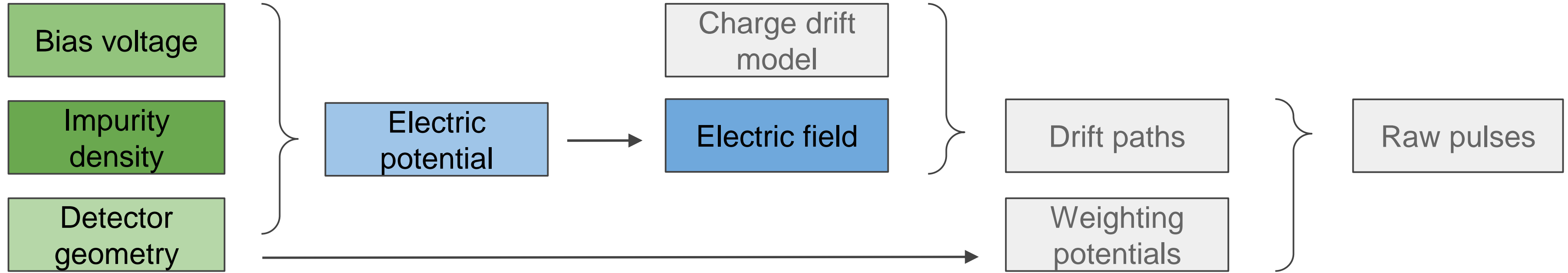
Electric Field Calculation

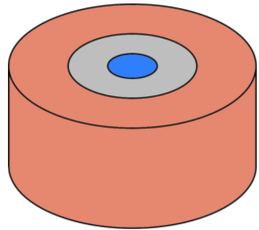
```
julia > calculate_electric_field!(sim)
```

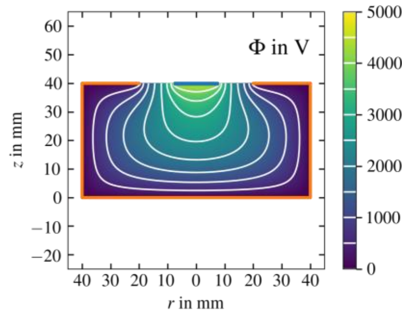


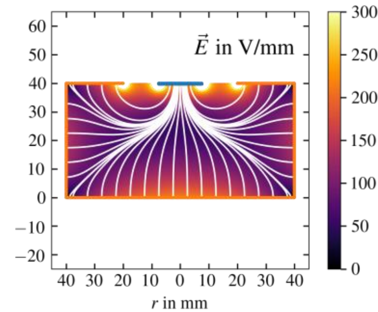
MAX-PLANCK-INSTITUT
FÜR PHYSIK

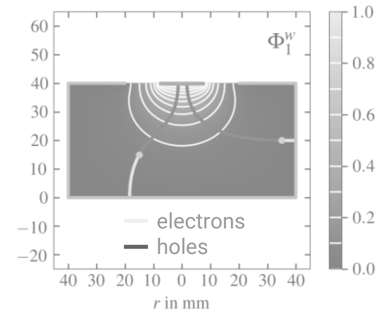
Pulse Shape Simulation Chain

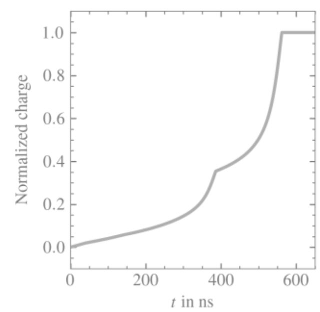






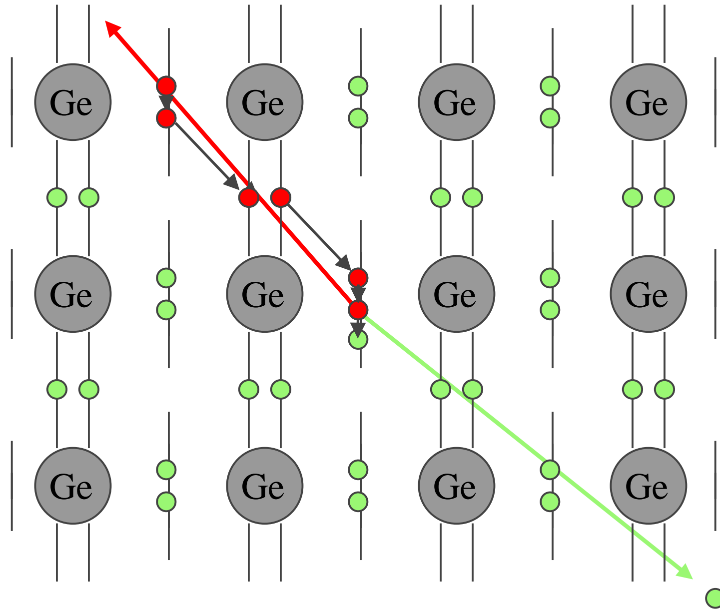


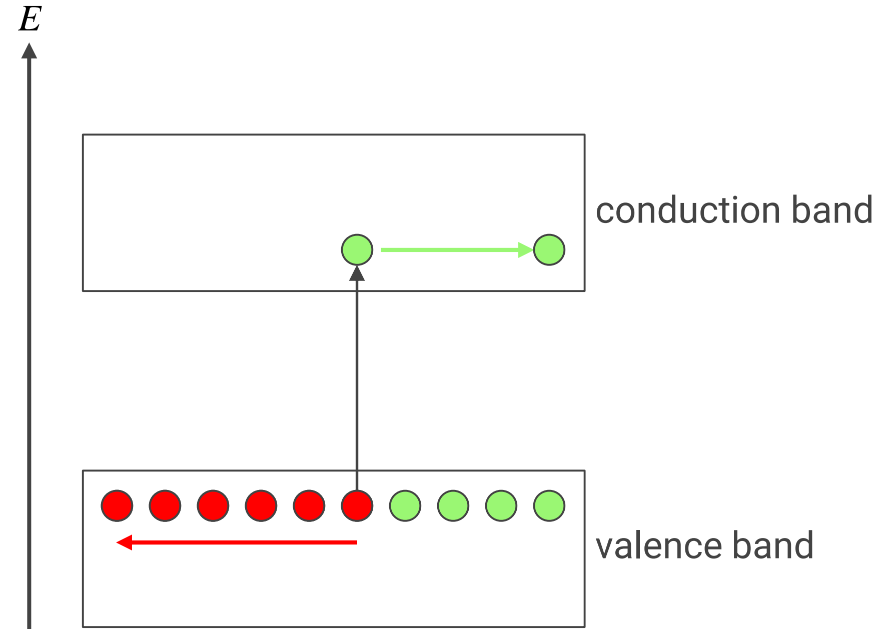




MAX-PLANCK-INSTITUT FÜR PHYSIK

Charge Drift Models





one electron-hole pair per 2.96eV

MAX-PLANCK-INSTITUT FÜR PHYSIK

Charge Drift Models

Charge drift model

Charge carriers in germanium move in the presence of an electric field $\mathbf{E}(\mathbf{r})$

Drift velocity of electrons and holes: $\mathbf{v}_{e,h}(\mathbf{r}) = \mu_{e,h}\,\mathbf{E}(\mathbf{r})$

$\mu_{e,h}$ is the mobility tensor:
- saturates for high electric field strengths
- anisotropic in germanium
- temperature dependent

There are models for $\mu_{e,h}$:

L. Mihailescu *et al.*, Nucl. Instr. and Meth. A **447** (2000) 350, doi: 10.1016/S0168-9002(99)01286-3
B. Bruyneel *et al.*, Nucl. Instr. and Meth. A **569** (2006) 764, doi: 10.1016/j.nima.2006.08.130

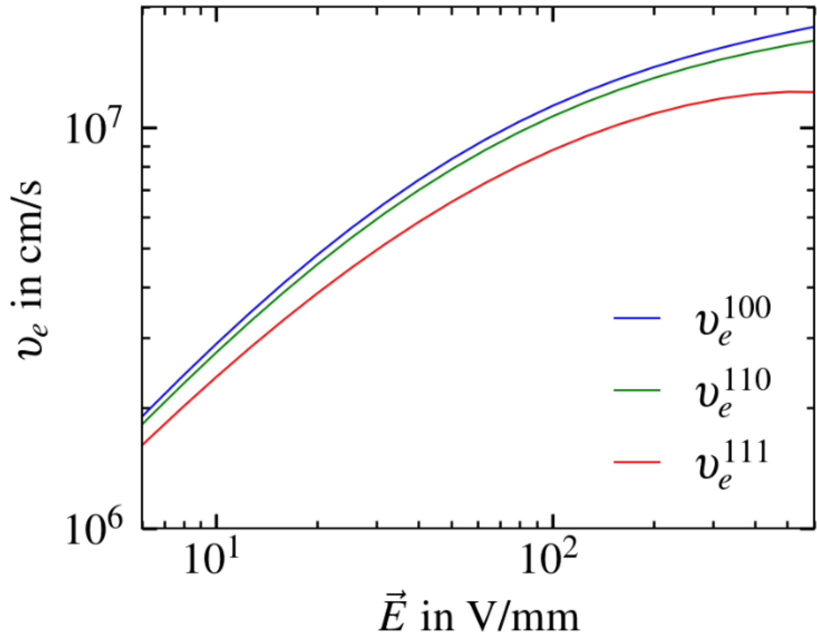
but usually parameters of the models have to be fitted to each individual detector

MAX-PLANCK-INSTITUT
FÜR PHYSIK

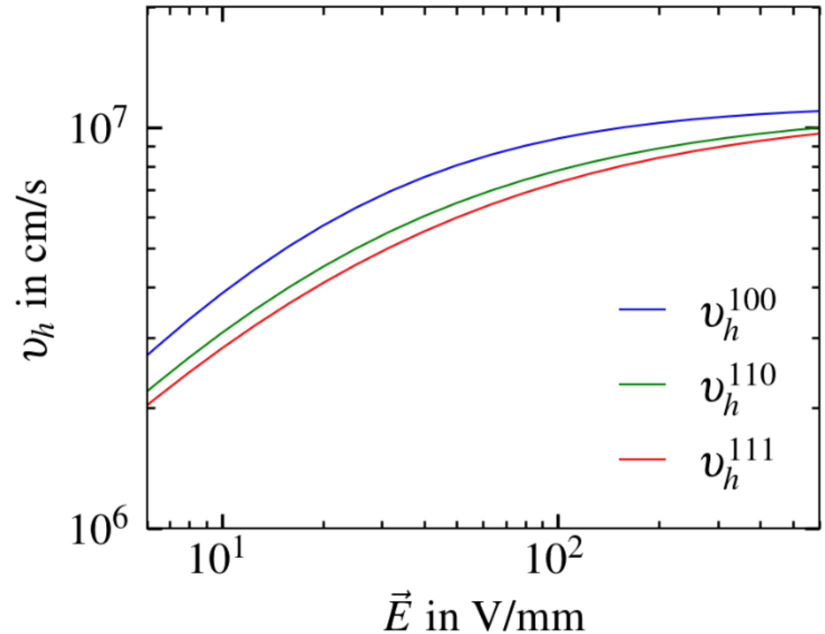
Charge Drift Models

Charge drift model

Electron drift in germanium



Hole drift in germanium



SSD offers a predefined model
doi: [10.1016/j.nima.2006.08.130](https://doi.org/10.1016/j.nima.2006.08.130)

Charge drift model

Charge Drift Models

julia > cdm = ADLChargeDriftModel( )
julia > sim.detector = SolidStateDetector(sim.detector, cdm)

Custom Charge Drift Model

The user can implement and use his own drift model.

The first step is to define a `struct` for the model which is a subtype of `SolidStateDetectors.AbstractChargeDriftModel`:

```
using SolidStateDetectors
using SolidStateDetectors: SSDFloat, AbstractChargeDriftModel
using StaticArrays

struct CustomChargeDriftModel{T <: SSDFloat} <: AbstractChargeDriftModel{T}
    # optional fields to parameterize the model
end
```

The second step is to define two methods (`getVe` for electrons and `getVh` for holes), which perform the transformation of an electric field vector, `fv::SVector{3,T}`, into a velocity vector. Note, that the vectors are in cartesian coordinates, independent of the coordinate system (cartesian or cylindrical) of the simulation.

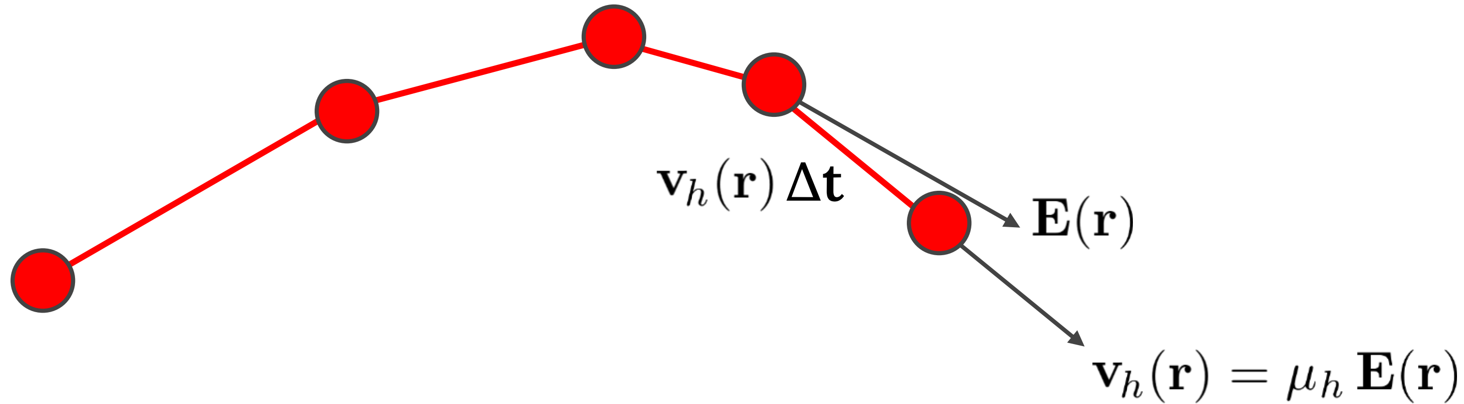
```
function SolidStateDetectors.getVe(fv::SVector{3, T}, cdm::CustomChargeDriftModel)::SVector{3, T
    # arbitrary transformation of fv
    return -fv
end

function SolidStateDetectors.getVh(fv::SVector{3, T}, cdm::CustomChargeDriftModel)::SVector{3, T
    # arbitrary transformation of fv
    return fv
end
```

Documentation on GitHub
https://juliaphysics.github.io/SolidStateDetectors.jl/stable/

MAX-PLANCK-INSTITUT FÜR PHYSIK

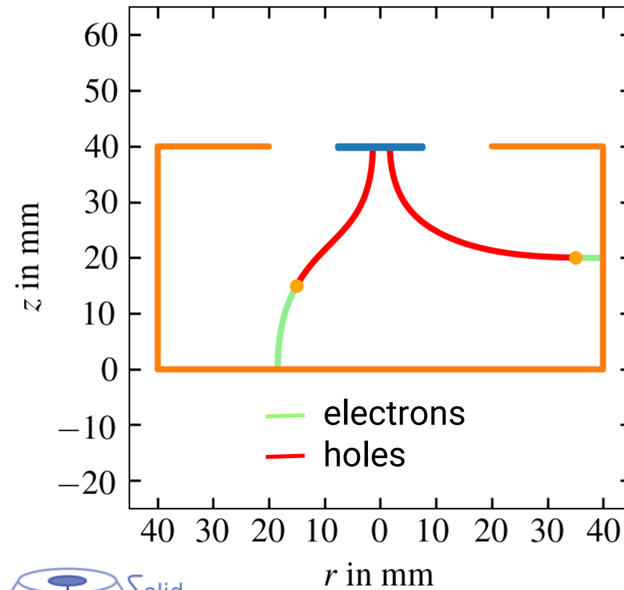
Charge Drift Simulation

Drift velocity for electrons and holes: $\mathbf{v}_{e,h}(\mathbf{r}) = \mu_{e,h}\,\mathbf{E}(\mathbf{r})$



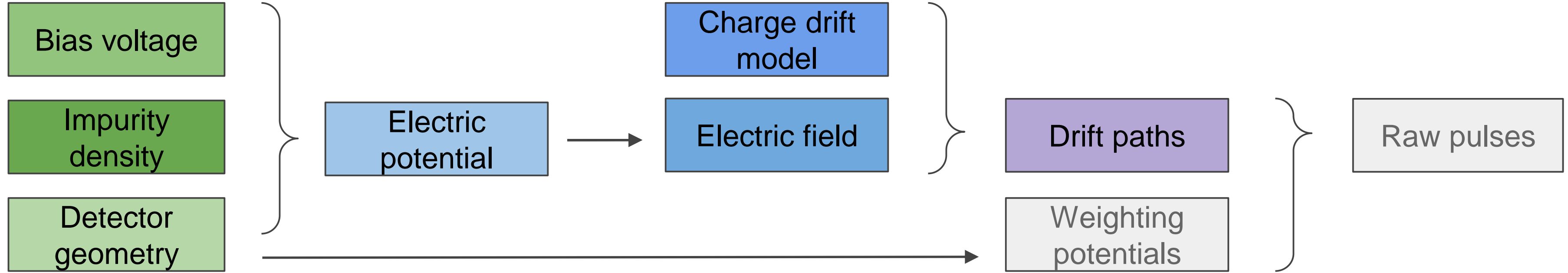
Charge Drift Simulation

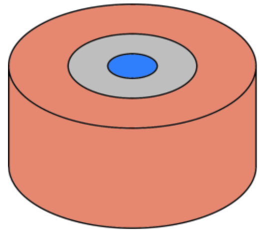
```
julia > locations = [CartesianPoint(0.035,0,0.02), CartesianPoint(-0.015,0,0.015)]
julia > energies = [1000u"keV", 300u"keV"]
julia > evt = Event(locations, energies)
julia > drift_charges!(evt, sim)
```

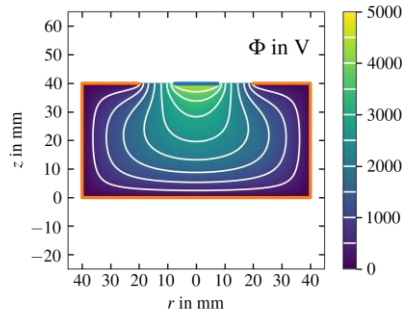


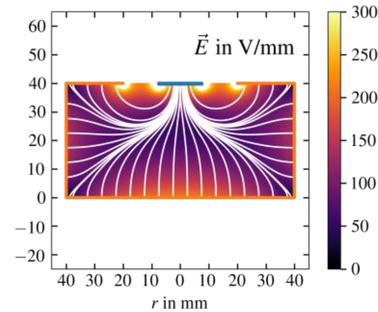
MAX-PLANCK-INSTITUT
FÜR PHYSIK

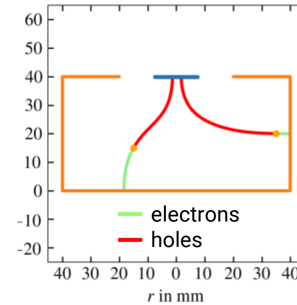
Pulse Shape Simulation Chain

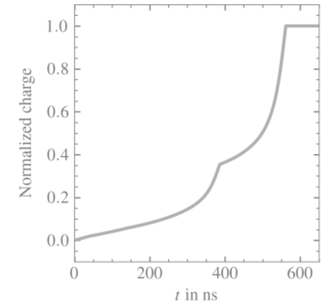












MAX-PLANCK-INSTITUT FÜR PHYSIK

Weighting Potential Calculation

Weighting potentials

$\Phi_i^w(\mathbf{r})$ is the so-called weighting potential for electrode i.

It describes how much charge is induced on the electrode depending on the position **r** of the charge carrier in the crystal.

$$\nabla \cdot (\epsilon_r(\mathbf{r}) \cdot \nabla \Phi_i^w(\mathbf{r})) = 0$$

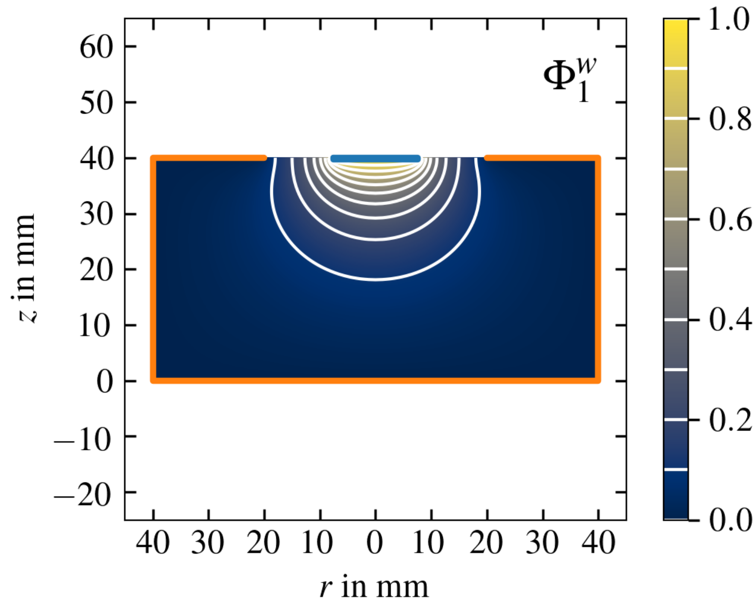
Same algorithm as for the electric potential but:

- Charge density is set to 0
- The potential values at all contacts are set to 0, but only the potential value of contact i is set to 1.

MAX-PLANCK-INSTITUT FÜR PHYSIK

Weighting Potential Calculation


```
julia > calculate_weighting_potential!(sim, 1)
```

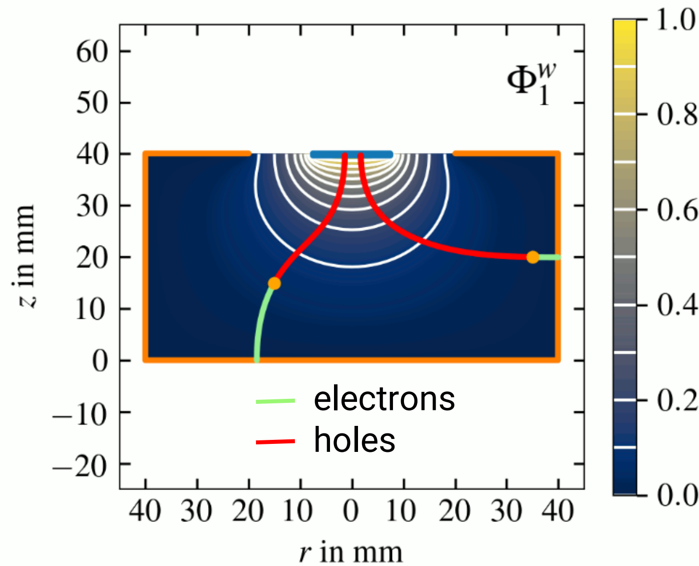


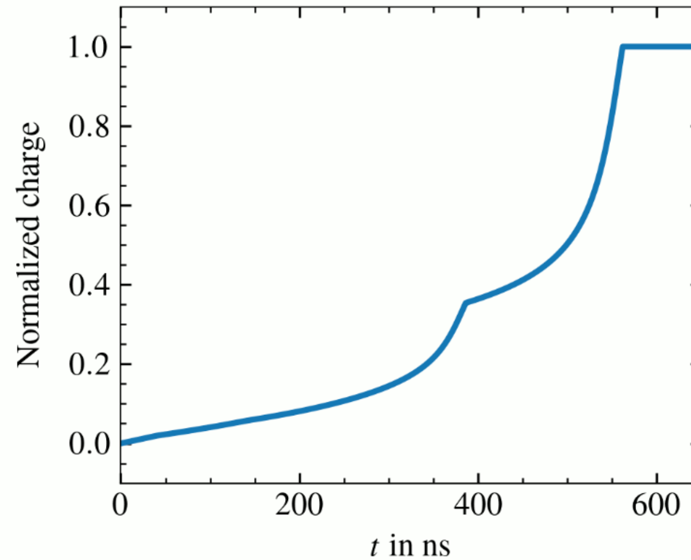
MAX-PLANCK-INSTITUT FÜR PHYSIK

Raw pulses

Signal Generation

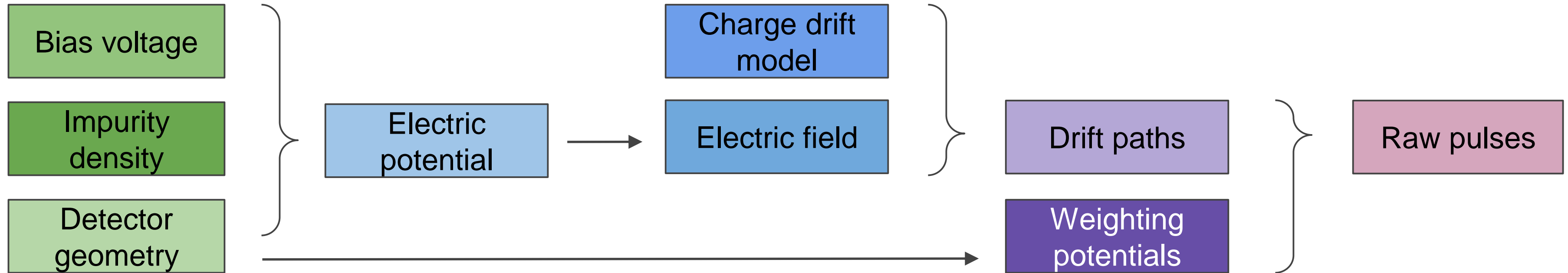
Shockley-Ramo Theorem $Q_i^{ind}(\mathbf{r}_e(t), \mathbf{r}_h(t)) = q \cdot [\Phi_i^w(\mathbf{r}_e(t)) - \Phi_i^w(\mathbf{r}_h(t))]$

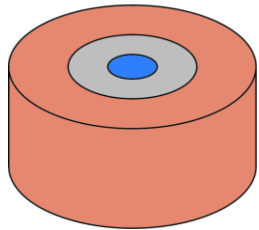


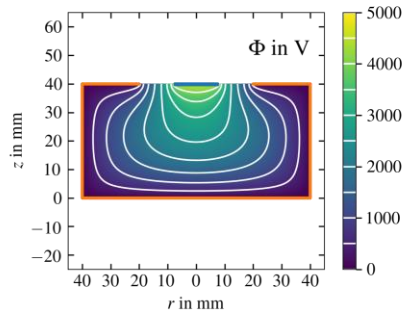


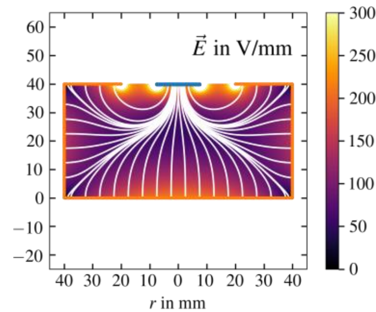
MAX-PLANCK-INSTITUT FÜR PHYSIK

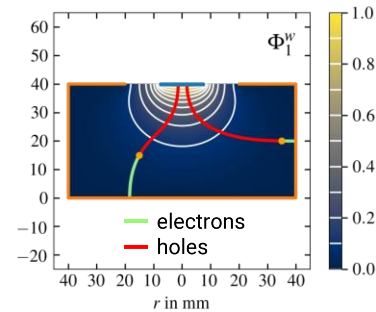
Pulse Shape Simulation Chain

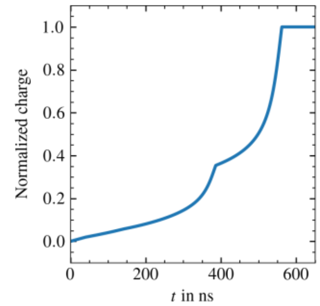












MAX-PLANCK-INSTITUT FÜR PHYSIK

Pulse Shape Simulation Chain

```
julia > using SolidStateDetectors, Unitful

julia > sim = Simulation{Float64}(“BEGe.yaml”)
julia > calculate_electric_potential!(sim)
julia > calculate_electric_field!(sim)
julia > for i in 1:2
            calculate_weighting_potential!(sim, i)
        end

julia > locations = [CartesianPoint(0.035,0,0.02)]
julia > energies = [1000u”keV”]
julia > evt = Event(locations, energies)
julia > simulate!(evt, sim)
```

MAX-PLANCK-INSTITUT
FÜR PHYSIK

Configuration File

```yaml
name: Point-contact detector
units:
  length: mm
  angle: deg
  potential: V
  temperature: K
grid:
  coordinates: cylindrical
  axes:
    r:
      to: 60
      boundaries: inf
    phi:
      from: 0
      to: 0
      boundaries:
        left: periodic
        right: periodic
    z:
      from: -20
      to: 60
      boundaries:
        left: inf
        right: inf
medium: vacuum
```

```yaml
detectors:
- semiconductor:
    material: HPGe
    impurity_density:
      name: constant
      value: -1e10cm^-3
    charge_drift_model:
      include: ADLChargeDriftModel/drift_velocity_config.yaml
    geometry:
      translate:
        tube:
          r: 40
          h: 40
        z: 20
  contacts:
  - name: Core
    material: HPGe
    id: 1
    potential: -4500
    geometry:
      tube:
        r: 7.5
        h: 0.3
        origin:
          z: 39.85
```

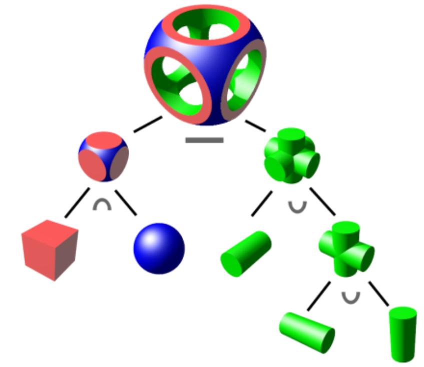
```yaml
- name: Mantle
  material: HPGe
  id: 2
  potential: 0
  geometry:
    union:
    - tube:
        r:
          from: 0
          to: 40
        h: 0
    - tube:
        r:
          from: 40
          to: 40
        h: 40
        origin:
          z: 20
    - tube:
        r:
          from: 20
          to: 40
        h: 0
        origin:
          z: 40
```

Impurity density

Charge drift model

Bias voltage

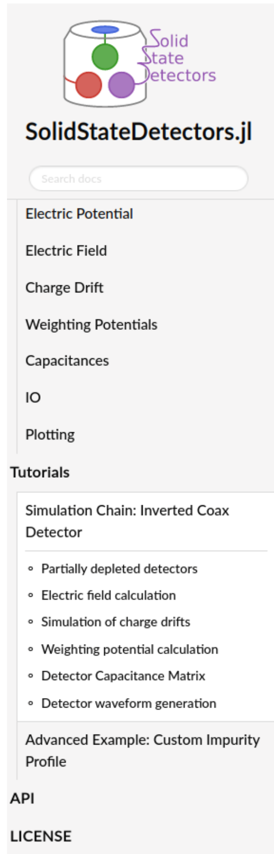
Detector geometry



Constructive Solid Geometry
Documentation on GitHub

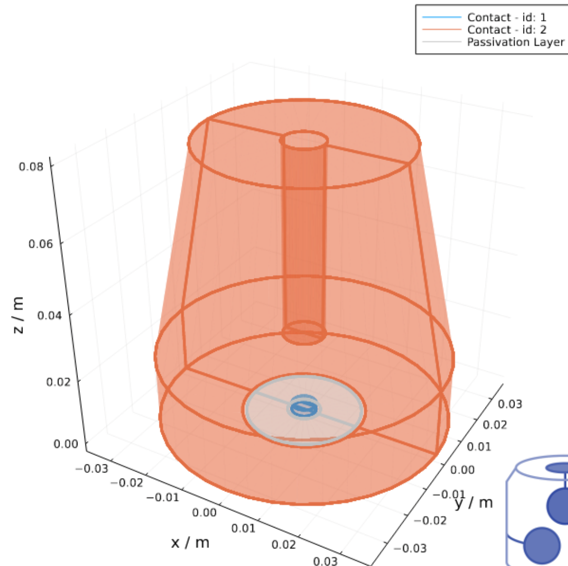
MAX-PLANCK-INSTITUT FÜR PHYSIK

Documentation on GitHub



Simulation Chain: Inverted Coax Detector

```
using Plots
using SolidStateDetectors
using Unitful

T = Float32
sim = Simulation{T}(SSD_examples[:InvertedCoax])

plot(sim.detector, size = (700, 700))
```



Find the latest version of the documentation on GitHub: https://juliaphysics.github.io/SolidStateDetectors.jl/stable/

MAX-PLANCK-INSTITUT FÜR PHYSIK

Pulse Shape Simulation Chain

```
julia > using SolidStateDetectors, Unitful

julia > sim = Simulation{Float64}(“BEGe.yaml”)
julia > calculate_electric_potential!(sim)
julia > calculate_electric_field!(sim)
julia > for i in 1:2
            calculate_weighting_potential!(sim, i)
        end

julia > locations = [CartesianPoint(0.035,0,0.02)]
julia > energies = [1000u”keV”]
julia > evt = Event(locations, energies)
julia > simulate!(evt, sim)
```

MAX-PLANCK-INSTITUT
FÜR PHYSIK

Undepleted Detectors

```
julia > using SolidStateDetectors, Unitful

julia > sim = Simulation{Float64}("BEGe.yaml")
julia > sim.detector = SolidStateDetector(sim, contact_potential = 500, contact_id = 1)
julia > calculate_electric_potential!(sim, depletion_handling = true)
julia > calculate_electric_field!(sim)
julia > for i in 1:2
            calculate_weighting_potential!(sim, i, depletion_handling = true)
        end

julia > calculate_mutual_capacitance(sim, (1, 2))
```

MAX-PLANCK-INSTITUT
FÜR PHYSIK

GPU Support

```
julia > using SolidStateDetectors, Unitful
julia > using CUDAKernels, CUDA

julia > sim = Simulation{Float64}("BEGe.yaml")
julia > calculate_electric_potential!(sim, device_array_type = CuArray)
julia > calculate_electric_field!(sim)
julia > for i in 1:2
            calculate_weighting_potential!(sim, i, device_array_type = CuArray)
        end

julia > locations = [CartesianPoint(0.035,0,0.02)]
julia > energies = [1000u"keV"]
julia > evt = Event(locations, energies)
julia > simulate!(evt, sim)
```

MAX-PLANCK-INSTITUT
FÜR PHYSIK

Simulating Group Effects

```
julia > using SolidStateDetectors, Unitful

julia > sim = Simulation{Float64}("BEGe.yaml")
julia > calculate_electric_potential!(sim)
julia > calculate_electric_field!(sim)
julia > for i in 1:2
            calculate_weighting_potential!(sim, i)
        end

julia > locations = [CartesianPoint(0.035,0,0.02)]
julia > energies = [1000u"keV"]
julia > evt = Event(NBodyChargeCloud(locations, energies, 100))
julia > simulate!(evt, sim, diffusion = true, self_repulsion = true)
```

MAX-PLANCK-INSTITUT
FÜR PHYSIK