Implementing delayed weighting fields in GARFIELD++

J. Hasenbichler, W. Riegler, H. Schindler, A. Wang

RD50 Workshop, 13 June 2019

What's GARFIELD++?

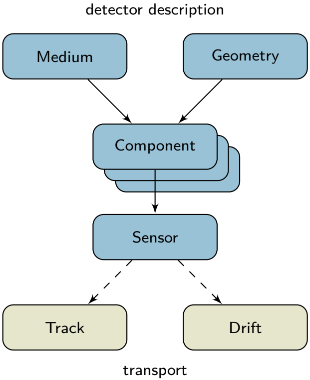
- GARFIELD++ is a C++ toolkit for the detailed simulation of particle detectors that are based on ionization measurement in gases or semiconductors.
- It inherits many concepts and techniques from the Fortran program GARFIELD (`https://cern.ch/garfield`), which is widely used for simulating gas-based detectors.
- Development of the C++ version of GARFIELD started $\sim$ 2011.
- Main differences with respect to the Fortran version include
  - focus on microscopic electron transport in gases,
  - user interface,
  - option to simulate silicon detectors.
- For more details, see `https://cern.ch/garfieldpp`.
- The source code is available on `https://gitlab.cern.ch/garfield/garfieldpp`.
- Pre-compiled libraries are available on cvmfs.

material properties

- gas → Magboltz
- silicon

field calculation

- analytic
- field maps
- neBEM



primary ionization

- Heed
- SRIM

charge transport

- microscopic
- MC integration
- RKF integration

Microscopic simulation of electron avalanches in a GEM (left) and around a wire (right).

Primary ionization

- Ionization by fast charged particles can be simulated using the program HEED (interfaced to GARFIELD++), based on the photoabsorption ionization (PAI) model.
  - I. B. Smirnov, Nucl. Instr. Meth. A **554** (2005), 474 – 493 (link).
- HEED simulates not only the deposited energy, but also atomic relaxation and δ electron transport. As a result, one obtains the position of all "conduction" electrons/holes.
- For simulating the ionization by ions, one can import results calculated using SRIM.
  - `http://garfieldpp.web.cern.ch/garfieldpp/examples/srim/`
- It is also possible to interface GEANT4 and GARFIELD++.
  - D. Pfeiffer *et al.*, Nucl. Instr. Meth. A **935** (2019), 121–134 (link).
  - `https://garfieldpp.web.cern.ch/garfieldpp/examples/geant4-interface/`

Electric fields

- For simple structures, can use parameterizations provided by the user.
- For more complex devices, one typically imports field maps calculated using TCAD:
  - either by probing the electric field/potential in SVisual on a regular grid and exporting the values to a text file, which can then be read by GARFIELD++, or
  - by importing directly the mesh (`.grd` file) and solution (`.dat` file).
- Can import maps of mobility, lifetimes and other parameters at the same time.

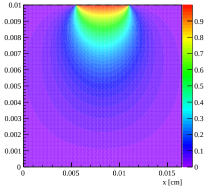
Signals in a sensor with zero conductivity

- Given the coordinates $\mathbf{x}_j$ of each point along a simulated drift line, the induced current is calculated using the usual Shockley-Ramo formalism,

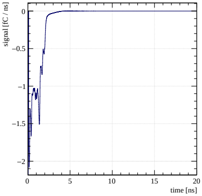
$$i\left(t_j\right) = -q\mathbf{E}_w\left(\mathbf{x}_j\right) \cdot \mathbf{v}_j,$$

  where $\mathbf{E}_w$ is the static weighting field.
- For calculating $\mathbf{E}_w$, the same approaches as for the (drift) electric field can be followed.
  - Analytic expressions for strip and pixel weighting fields are pre-implemented.
- The front-end response can be modelled by convoluting $i\left(t\right)$ with a transfer function.

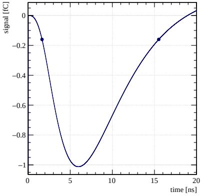
Example: Signal in a 100 μm thick *n*-on-*p* sensor with 55 μm pixel pitch.



Weighting potential.



Induced current from a charged particle track.



Front-end output (after convolution).

Signals in a sensor with finite conductivity

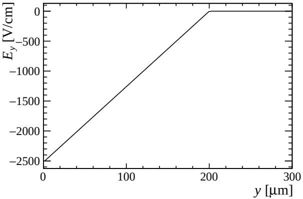
- The weighting field is split in a prompt weighting field and a delayed weighting field.
- The prompt contribution is calculated in the same way as in the static ($\sigma = 0$) case.
- The delayed weighting field for a drift path segment $\mathbf{x}(t'),\, t_j < t' < t_{j+1}$ is given by

$$i\left(t_j + t\right) = -q \int_0^t \mathrm{d}t' \mathbf{E}_w \left(\mathbf{x}\left(t'\right), t - t'\right) \cdot \mathbf{v}\left(t'\right).$$

- We assume that the velocity along a drift line step is constant.

Simple example

- As an illustration/proof of principle, consider an underdepleted planar pad sensor with a thickness of 300 µm and a depleted depth of 200 µm.
- We'll start with an analytic model of the electric field and the (static) weighting field.

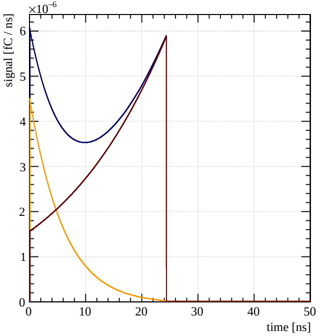


```
// Thickness of the sensor [cm].
constexpr double gap = 300.e-4;
// Depletion depth [cm].
constexpr double d = 200.e-4;

void efield(const double /*x*/, const double y, const double /*z*/, double& ex, double& ey, double& ez) {
  ex = ez = 0.;
  constexpr double v = -25.2;
  ey = y < d ? 2 * (v / d) * (1. - y / d) : 0.;
}
void wfield(const double /*x*/, const double /*y*/, const double /*z*/,
            double& wx, double& wy, double& wz, const std::string /*label*/) {
  wx = wz = 0.;
  wy = 1. / gap;
}

int main(int argc, char *argv[]) {
  Garfield::MediumSilicon si;
  Garfield::GeometrySimple geo;
  Garfield::SolidBox box(0, 0.5 * gap, 0, gap, 0.5 * gap, gap);
  geo.AddSolid(&box, &si);
  //...
  Garfield::ComponentUser cmp;
  cmp.SetGeometry(&geo);
  // Set the function to be called for calculating the drift field.
  cmp.SetElectricField(efield);
  // Set the function to be called for calculating the weighting field.
  cmp.SetWeightingField(wfield);

  //...
}
```

Simple example (continued)

- We first compute the prompt signal induced by a *e*-*h* pair created at a depth of 150 μm.

```
//...
int main(int argc, char *argv[]) {
  //...
  Garfield::Sensor sensor;
  // Set the object that calculates the drift field.
  sensor.AddComponent(&cmp);
  // Use 2000 time bins with a width of 25 ps.
  sensor.SetTimeWindow(0., 0.025, 2000);
  // Set the object that calculates the weighting field.
  sensor.AddElectrode(&cmp, "readout");

  Garfield::AvalancheMC drift;
  drift.SetSensor(&sensor);
  // Make 1 um steps.
  drift.SetDistanceSteps(1.e-4);
  // Switch off diffusion.
  drift.DisableDiffusion();
  drift.EnableSignalCalculation();
  // Simulate an electron-hole pair starting at y = 150 um.
  drift.DriftElectron(0, 150.e-4, 0, 0);
  drift.DriftHole(0, 150.e-4, 0, 0);
  //...
}
```

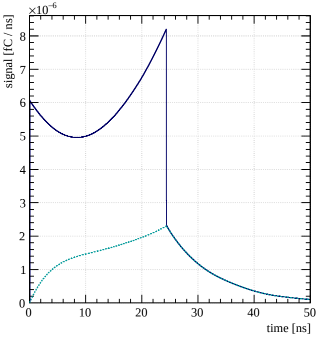


Total induced current as function of time and contributions from electron and hole.

Simple example (continued)

- As a next step, we include the delayed component of the signal.

```
//...
void dwfield(const double /*x*/, const double /*y*/,
             const double /*z*/, const double t,
             double& wx, double& wy, double& wz,
             const std::string& /*label*/) {
  // Time constant [ns].
  constexpr double tau = 7.9;
  wx = wz = 0.;
  wy = ((gap - d) / (gap * d)) * exp(-t / tau) / tau;
}

int main(int argc, char *argv[]) {
  //...
  // Set the function for calculating the delayed weighting field.
  cmp.SetDelayedWeightingField(dwfield);
  //...
  sensor.EnableDelayedSignal();
  // Specify the times t - t' at which we want
  // to calculate the delayed signal.
  const std::vector<double> times = {0., 0.1, 0.2, 0.3, 0.4,
     0.5, 0.6, 0.7, 0.8, 0.9, 1.,  2.,  3.,  4.,
     5., 6.,  7.,  8.,  9., 10., 20., 30., 40.,
    50., 60., 70., 80., 90., 100.};
  sensor.SetDelayedSignalTimes(times);
  //...
}
```

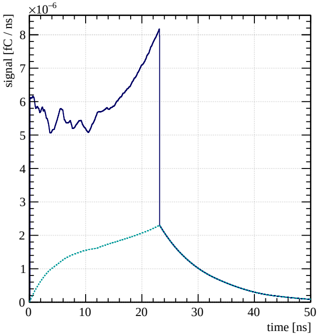


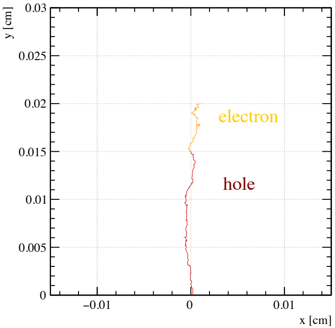
Total induced current as function of time and delayed component.

Simple example (continued)

- In a realistic simulation we will of course want to switch on diffusion.

```
int main(int argc, char *argv[]) {
  //...
  // drift.DisableDiffusion();
  //...
}
```





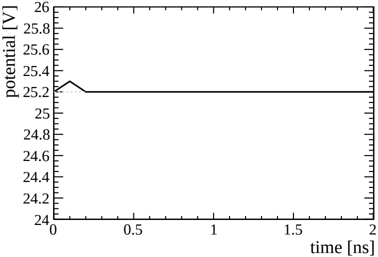
Simple example (continued)

- Let's now do the same simulation using field maps for the drift and weighting fields.
- In order to calculate the weighting fields in TCAD, we use the following recipe.
  - Calculate the quasi-stationary solution $\mathbf{E}_0$ with all electrodes at their "real" potentials.
  - Run a transient simulation applying a short triangular voltage pulse (duration $2 \times \Delta t$, peak ΔV) at the electrode we want to read out. Save the field $\mathbf{E}_+$ at different moments in time.
  - The prompt weighting field is given by

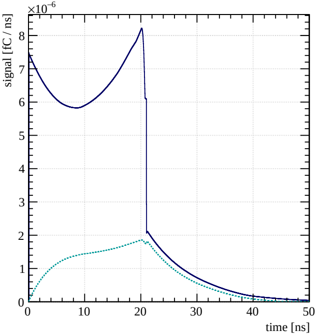
$$\frac{1}{\Delta V} \left[\mathbf{E}_+ \left(t = \Delta t\right) - \mathbf{E}_0\right].$$

  - The delayed weighting field is given by

$$\frac{1}{\Delta V \Delta t} \mathbf{E}_+ \left(t > 2\Delta t\right).$$



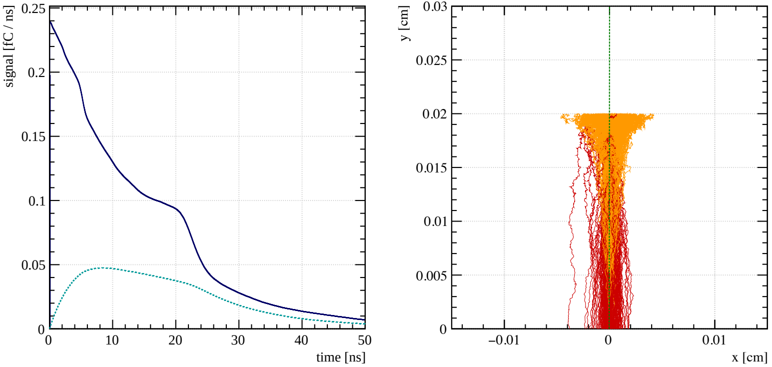
```
int main(int argc, char *argv[]) {
  //...
  Garfield::ComponentVoxel cmp;
  cmp.SetMesh(nX, nY, 1, xMin, xMax, yMin, yMax, zMin, zMax);
  cmp.LoadElectricField("Efield.txt", "XY", false, false);
  cpmp.LoadWeightingField("Weighting_00.txt", "XY", false);
  for (unsigned int i = 0; i < nTimes; ++i) {
    char filename[50];
    sprintf(filename, "Weighting_%02d.txt", i + 1);
    cmp.LoadWeightingField(filename, "XY", times[i], false);
  }
  cmp.EnableInterpolation();
  //...
}
```



Simple example (continued)

- Finally, let's simulate the induced signal from a charged particle track.

```
int main(int argc, char *argv[]) {
  //...
  TrackHeed track;
  track.SetSensor(&sensor);
  // Set the particle type and momentum [GeV/c].
  track.SetParticle("muon");
  track.SetMomentum(10.e9);
  // Simulate a track at perpendicular incidence.
  track.NewTrack(0, 0, 0, 0, 0, 1, 0);
  double xc = 0., yc = 0., zc = 0., tc = 0., ec = 0., extra = 0.;
  int nc = 0;
  while (track.GetCluster(xc, yc, zc, tc, nc, ec, extra)) {
    for (int i = 0; i < nc; ++i) {
      double xe = 0., ye = 0., ze = 0., te = 0., ee = 0.;
      double dx = 0., dy = 0., dz = 0.;
      track.GetElectron(i, xe, ye, ze, te, ee, dx, dy, dz);
      drift.DriftElectron(xe, ye, ze, te);
      drift.DriftHole(xe, ye, ze, te);
    }
  }
  //...
}
```


signal [fC / ns]
0.25
0.2
0.15
0.1
0.05
0
0
10
20
30
40
50
time [ns]
y [cm]
0.03
0.025
0.02
0.015
0.01
0.005
0
−0.01
0
0.01
x [cm]

Summary and outlook

- GARFIELD++ is a toolkit that can be used for the detailed simulation of silicon sensors.
- We have implemented the calculation of induced signals in resistive geometries based on the delayed weighting field formalism.
- Some optimisation in terms of speed and accuracy remains to be done.
- As a next step, apply the method to realistic devices.