Charge Transport in a GEM

RD51 Simulation School

January 19, 2011

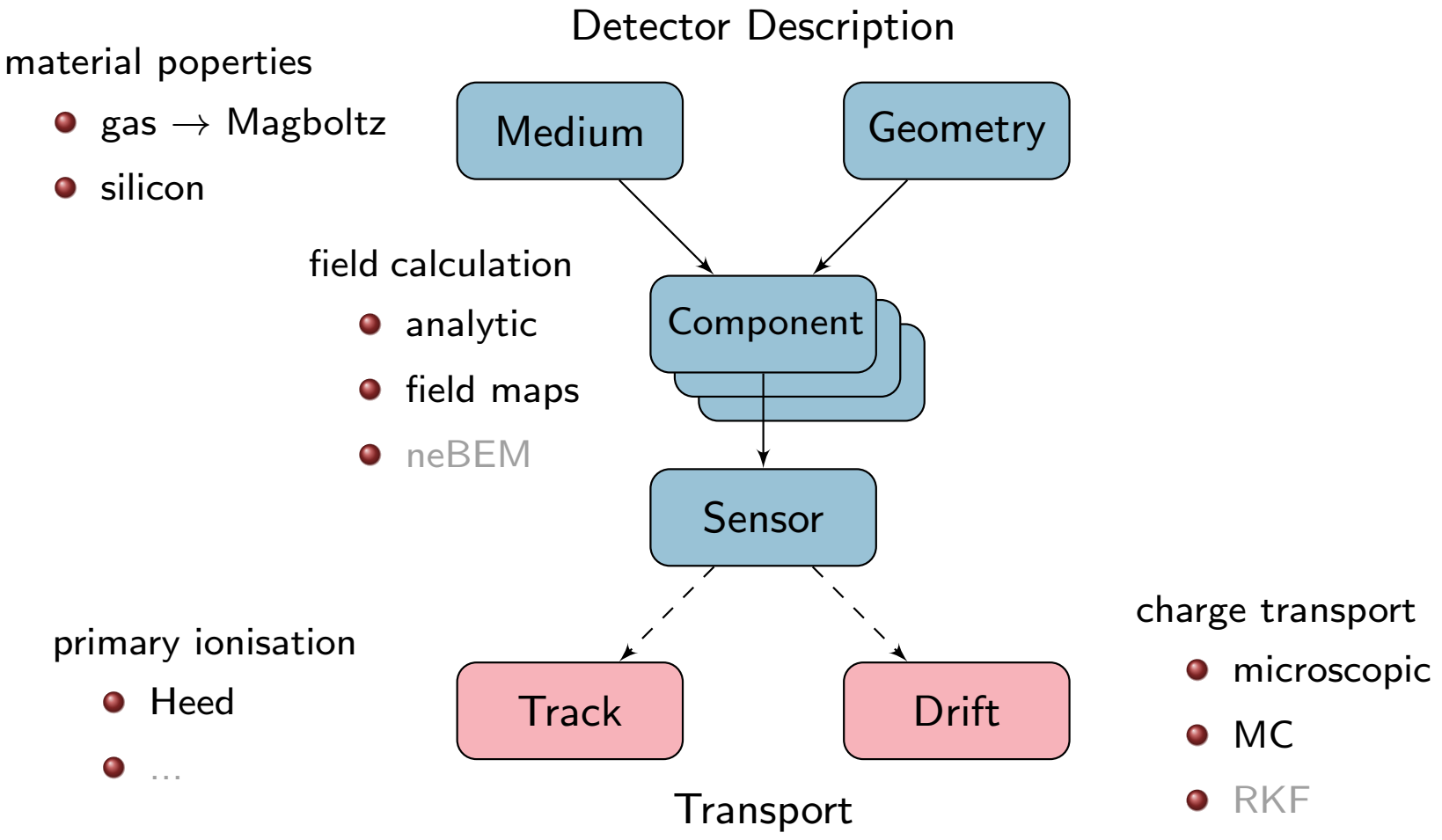
Outline

Using the Ansys field map calculated in the morning, we will simulate the drift of electrons and ions in a GEM.

In particular, we will discuss how to

- import and inspect the field map,
- setup the gas mixture,
- compute an electron avalanche,
- transport the ions created in the avalanche, and
- calculate ion feedback and charging up.

Garfield++
Layout



Garfield++

Classes

In this exercise we will use:

- MediumMagboltz
- ComponentAnsys123
- Sensor
- AvalancheMicroscopic, AvalancheMC
- a couple of classes for visualization purposes (`View...`)
- ROOT histograms

Garfield++

How To Run

1. interactively, on the ROOT command line

```
$GARFIELD_HOME/Examples/Test/garfroot
```

   → useful for simple tests, playing

2. in compiled mode, writing a small program

Note The Garfield++ classes are enclosed in a namespace `Garfield`. In the following examples, either add `Garfield::` in front of the class names or make them globally acessible

```
using namespace Garfield;
```

Field Map
Initialisation

The initialisation of `ComponentAnsys123` consists of

- loading the mesh (`ELIST.lis`, `NLIST.lis`), the list of nodal solutions (`PRNSOL.lis`), and the material properties (`MPLIST.lis`);
- specifying the length unit to be used;
- setting the appropriate periodicities/symmetries.

```
ComponentAnsys123* fm = new ComponentAnsys123();
// Load the field map.
fm->Initialise("ELIST.lis", "NLIST.lis", "MPLIST.lis", "PRNSOL.lis", "mm");
// Set the periodicities.
fm->EnableMirrorPeriodicityX();
fm->EnableMirrorPeriodicityY();
// Print some information about the cell dimensions.
fm->PrintRange();
```

Field Map

Plots

Next, we inspect the field map to make sure it makes sense. Using the class `ViewField`, we first make a plot of the potential along the hole axis (z axis).

```
ViewField* fieldView = new ViewField();
fieldView->SetComponent(fm);
// Plot the potential along the hole axis.
fieldView->PlotProfile(0., 0., 0.02, 0., 0., -0.02);
```

Let's also make a contour plot of the potential in the $x - z$ plane.

```
const double pitch = 0.014;
// Set the viewing plane (normal vector) and ranges.
fieldView->SetPlane(0., -1., 0., 0., 0., 0.);
fieldView->SetArea(-pitch / 2., -0.02, pitch / 2., 0.02);
fieldView->SetVoltageRange(-160., 160.);
fieldView->PlotContour();
```

Gas

We use a gas mixture of 80% argon and 20% CO_2 at room temperature ($T = 293.15$ K) and atmospheric pressure ($p = 760$ Torr).

```
MediumMagboltz* gas = new MediumMagboltz();
gas->SetComposition("ar", 80., "co2", 20.);
// Set temperature [K] and pressure [Torr].
gas->SetTemperature(293.15);
gas->SetPressure(760.);
```

As the name suggests, the class `MediumMagboltz` provides an interface to the Magboltz program.

Magboltz

- Using semi-classical MC simulation, Magboltz (author: S. Biagi) calculates transport properties of electrons in gas mixtures in a given uniform electric and magnetic field.
- It includes a database of electron-atom/molecule cross-sections for a large number of detection gases (see `http://rjd.web.cern.ch/rjd/cgi-bin/cross`).
- Source code of stand-alone program (latest version: 8.93) is available at `http://cern.ch/magboltz`

Microscopic Tracking

Use Magboltz cross-section database and transport algorithm for "event-by-event" electron transport in arbitrary fields.

Magboltz

A description of the program is given in

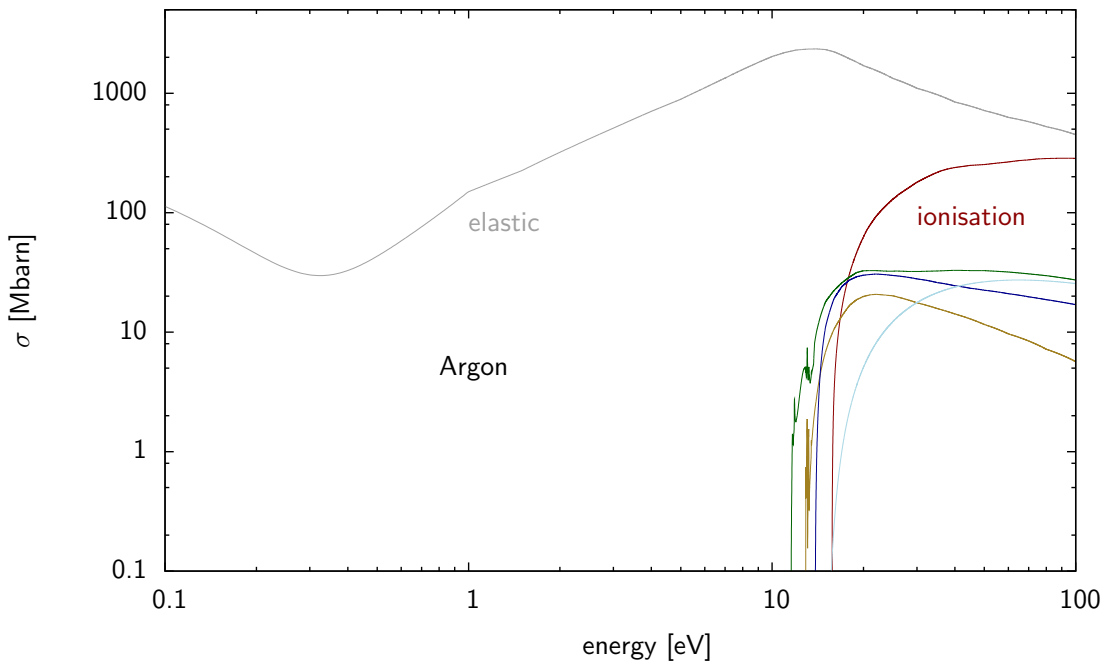
- S. F. Biagi, Nucl. Instr. Meth. A **421** (1999), 234-240

Recent Updates

- detailed modelling of excitation cross-sections (previously lumped together) $\rightarrow$ deexcitation processes (Penning transfer, light emission)
- improved modelling of angular scattering $\rightarrow$ diffusion, δ electron range
- cross-sections extended to high electron energies $\rightarrow$ primary ionisation, δ electron transport

Magboltz

Cross-sections, collision rates, and what to do with it

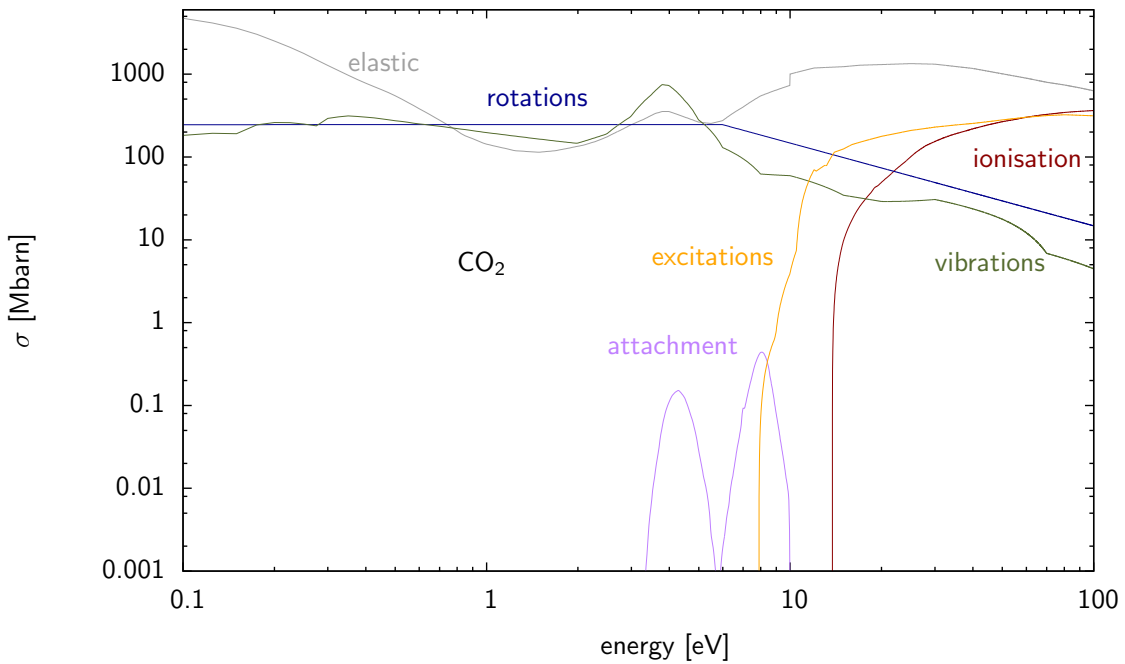


- Cross-sections
  - elastic
  - vibrations, rotations
  - excitations
  - attachment
  - ionisation
- Collision rate

$$\tau_i^{-1}(\epsilon) = N\sigma(\epsilon)\, v$$

Magboltz

Cross-sections, collision rates, and what to do with it



- Cross-sections
  - elastic
  - vibrations, rotations
  - excitations
  - attachment
  - ionisation
- Collision rate

$$\tau_i^{-1}(\epsilon) = N\sigma(\epsilon)\, v$$

Magboltz

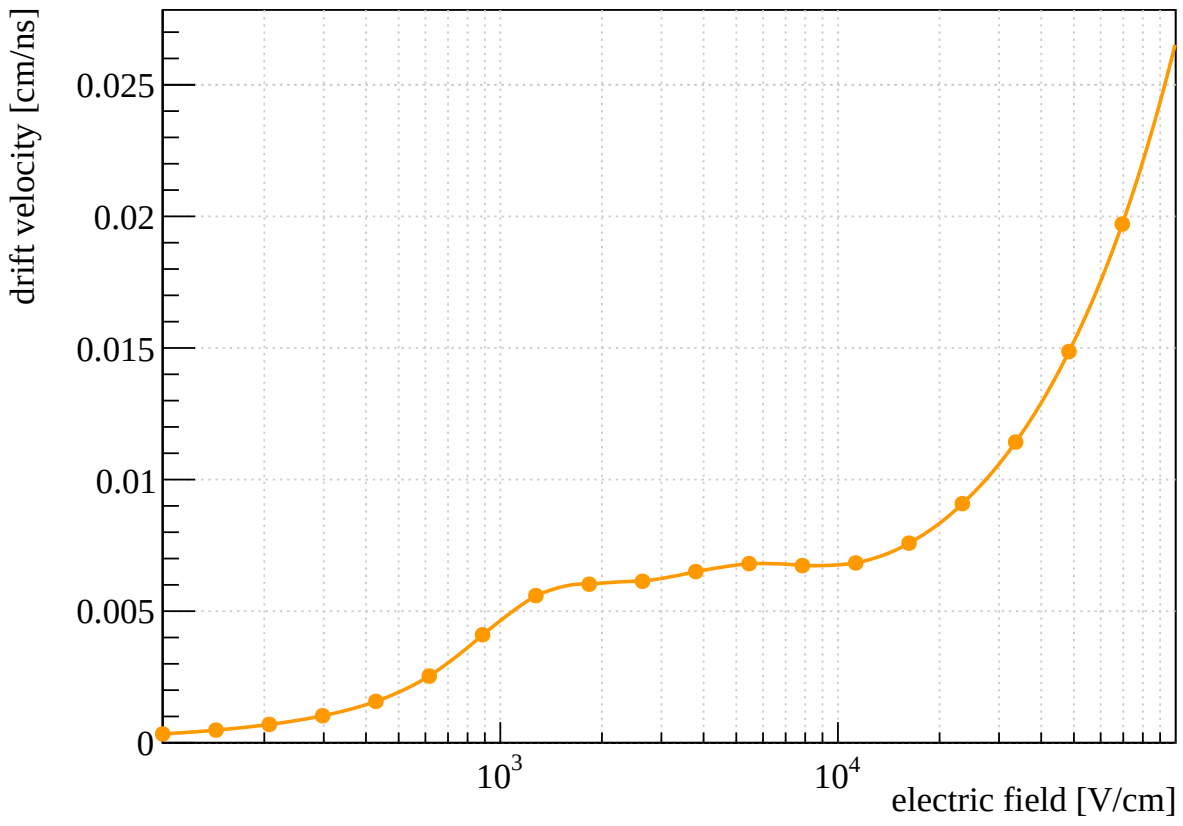
Cross-sections, collision rates, and what to do with it

Transport Algorithm

- Between collisions, electrons are traced on a vacuum trajectory, according to the local **E** and **B** field.
- The duration of a free-flight step is controlled by the total collision rate, $\tau^{-1} = \sum_i \tau_i^{-1}(\epsilon)$.
  change of energy during step $\rightarrow$ null-collision technique
- Calculate energy, direction and position after the step.
  $\epsilon(t) \rightarrow \epsilon(t + \Delta t), \mathbf{r}(t) \rightarrow \mathbf{r}(t + \Delta t), \mathbf{v}(t) \rightarrow \mathbf{v}(t + \Delta t)$
- Choose a scattering process.
  probability of scattering process i is proportional to $\tau_i^{-1}(\epsilon)$
- Depending on the type of collision, update the energy and the direction of motion, and continue stepping.

Magboltz

- Example: drift velocity v_d in Ar/CO_2 (80:20), at $p = 760$ Torr, $T = 293.15$ K



Gas

- For microscopic transport, we skip the calculation of the table of transport parameters ($v_d, D_L, D_T, \alpha, \eta$ etc.) and load directly the cross-sections from the Magboltz database.
- The collision rates τ_i^{-1} are stored on an evenly spaced energy grid ($0 < \epsilon < \epsilon_{max}$), where ϵ_{max} can be set by the user.
- For avalanche calculations, $\epsilon_{max} \approx 50 - 200$ eV is usually a reasonable choice.

```
gas->SetMaxElectronEnergy(200.);
gas->EnableDebugging();
gas->Initialise();
gas->DisableDebugging();
```

Gas $\leftrightarrow$ Field Map

- In order to track a particle through the detector we have to tell the `Component` which field map material corresponds to which `Medium`.
- Print a list of the field map materials:

```
fm->PrintMaterials();
```

- The gas can be identified by its dielectric constant, in our case $\varepsilon = 1$.

```
const int nMaterials = fm->GetNumberOfMaterials();
for (int i = 0; i < nMaterials; ++i) {
  const double eps = fm->GetPermittivity(i);
  if (eps == 1.) fm->SetMedium(i, gas);
}
```

Sensor

- Finally, we have to create a `Sensor` class, which is basically an assembly of "components".
- In general, a detector can be described by several "components", thus allowing
  - electric, magnetic and weighting fields to be calculated using different techniques;
  - fields from different components to overlap.
- In our case, the `Sensor` has only one `Component`.

```
Sensor* sensor = new Sensor();
sensor->AddComponent(fm);
```

- The `Sensor` class acts as an interface to the transport classes (and also takes care of signal calculation).

```
AvalancheMicroscopic* aval = new AvalancheMicroscopic();
aval->SetSensor(sensor);
```

Electron Transport

We are now ready to track an electron through the GEM.

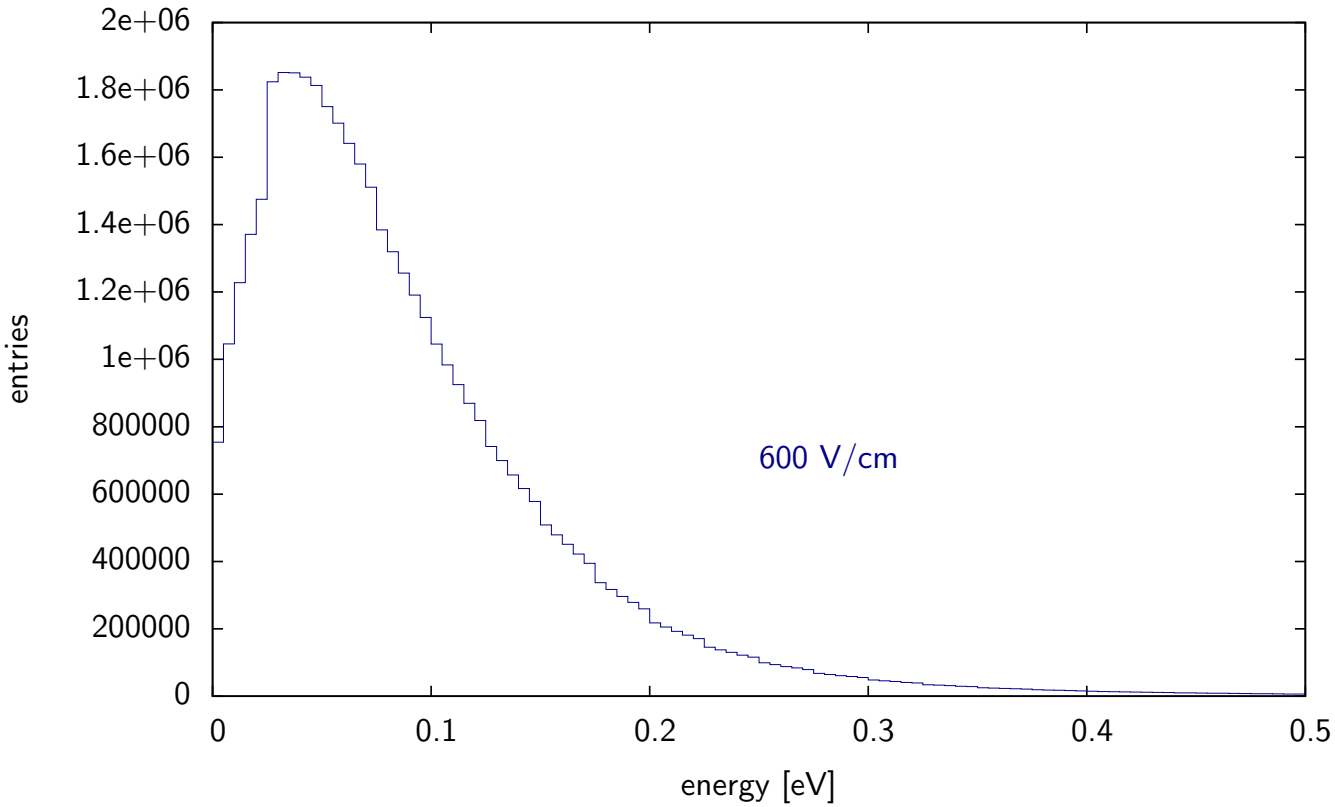
```
// Set the initial position [cm] and starting time [ns].
double x0 = 0., y0 = 0., z0 = 0.02, t0 = 0.;
// Set the initial energy [eV].
double e0 = 0.1;
// Calculate an electron avalanche (randomized initial direction).
aval->AvalancheElectron(x0, y0, z0, t0, e0, 0., 0., 0.);
...
```

In order to visualise the drift lines, we use the class `ViewDrift`.

```
ViewDrift* driftView = new ViewDrift();
driftView->SetArea(-2 * pitch, -2 * pitch, -0.02,
                   2 * pitch, 2 * pitch, 0.02);
aval->EnablePlotting(driftView);
aval->AvalancheElectron(x0, y0, z0, t0, e0, 0., 0., 0.);
// Plot the drift lines.
driftView->Plot();
```

Electron Transport
Initial Energy

Energy distribution in Ar/CO_2 (80:20) in a constant drift field:



In the drift gap, $E \gtrsim 600$ V/cm. At 600 V/cm, the mean electron energy is ≈ 0.09 eV.

Electron Transport

After the calculation, we can extract information such as the number of electrons/ions and their drift paths from `AvalancheMicroscopic`.

```
int ne, ni;
// Get the number of electrons and ions produced in the avalanche.
aval->GetAvalancheSize(ne, ni);
// Get the number of electron tracks.
int np = aval->GetNumberOfElectronEndpoints();
// Get the starting and endpoint of the first electron.
double x1, y1, z1, t1, e1;
double x2, y2, z2, t2, e2;
int status;
aval->GetElectronEndpoint(0, x1, y1, z1, t1, e1,
                          x2, y2, z2, t2, e2, status);
```

Penning Transfer

- Excited Ar levels with energy greater than the ionisation threshold of the admixture can enhance the gain due to photo-ionisation or collisional ionisation.
- In the simulation, this effect can be described in terms of a probability r that an excitation is converted to an ionising collision.
- Transfer efficiency r can be determined by gain curve fits, as described in
  - Ö. Sahin et al., JINST **5** (2010), P05002.
- For Ar/CO_2 (80:20), $r \approx 0.51$

```
// Probability of Penning transfer.
double rPenning = 0.51;
// Mean distance from the point of excitation.
double lambdaPenning = 0.;
gas->EnablePenningTransfer(rPenning, lambdaPenning, "ar");
```

Ion Transport

Microscopic transport is not available for ions. Unfortunately, there is no "Magboltz" for ions either, so we have to set the transport parameters "by hand".

Mobility

```
double emin = 100., emax = 100000.;
int ne = 20;
gas->SetFieldGrid(emin, emax, ne);
gas->LoadIonMobility("IonMobility_Ar+_Ar.txt");
```

Diffusion We assume thermal diffusion (default), i. e.

$$D_L = D_T = \sqrt{\frac{2k_B T}{qE}}.$$

Ion Transport
Mobility

Mobility of Ar^+ ions in Ar (at $T \approx 300$ K)

E/N $[10^{-17}\text{V cm}^2]$	μ_0 $[\text{cm}^2\text{V}^{-1}\text{s}^{-1}]$
< 12	1.53
15	1.52
20	1.51
25	1.49
30	1.47
40	1.44
50	1.41
60	1.38
80	1.32
100	1.27
120	1.22

E/N $[10^{-17}\text{V cm}^2]$	μ_0 $[\text{cm}^2\text{V}^{-1}\text{s}^{-1}]$
150	1.16
200	1.06
250	0.99
300	0.95
400	0.85
500	0.78
600	0.72
800	0.63
1000	0.56
1200	0.51
1500	0.46
2000	0.40

H. W. Ellis, R. Y. Pal, and E. W. McDaniel,
At. Data and Nucl. Data Tables **17** (1976), 177-210

Ion Transport

For tracking the ions we use the Monte Carlo integration technique.

```
AvalancheMC* drift = new AvalancheMC();
drift->SetSensor(sensor);
// Integrate in constant (2 um) distance intervals.
drift->SetDistanceSteps(2.e-4);
```

MC Algorithm

- Compute a step length Δs according to the velocity at the local field and the specified time step.
- Generate a diffusion step, based on D_L and D_t, scaled by $\sqrt{\Delta s}$, randomize according to 3 uncorrelated Gaussian distributions.
- Update the location by adding the step due to the velocity and the random step due to diffusion.

Ion Transport

To calculate an ion drift line, we do

```
double x0 = 0., y0 = 0., z0 = -0.02;
double t0 = 0.;
drift->DriftIon(x0, y0, z0, t0);
// Get the initial and final location of the ion.
double x1, y1, z1, t1;
double x2, y2, z2, t2;
int status;
drift->GetIonEndpoint(0, x1, y1, z1, t1, x2, y2, z2, t2, status);
```

Plotting the drift line also works in the same way as for electrons

```
drift->EnablePlotting(driftView);
...
driftView->Plot();
```

Electrons and Ions

```
double x1, y1, z1, t1, e1;
double x2, y2, z2, t2, e2;
int status;
aval->AvalancheElectron(x0, y0, z0, t0, e0, 0., 0., 0.);
const int np = aval->GetNumberOfElectronEndpoints();
// Loop over the endpoints, i. e. the electron drift lines.
for (int j = np; j--;) {
  // Get the start and end position of the electron.
  aval->GetElectronEndpoint(j, x1, y1, z1, t1, e1,
                               x2, y2, z2, t2, e2, status);
  // Calculate an ion drift line from the creation point of each electron.
  drift->DriftIon(x1, y1, z1, t1);
}
```

Normally, particles are transported until they exit the mesh. To speed up the calculation we restrict the drift region to $-100\mu\text{m} < z < +200\mu\text{m}$.

```
sensor->SetArea(-3 * pitch, -3 * pitch, -0.01,
                3 * pitch, 3 * pitch, 0.02);
```

Your Turn...

- In `$GARFIELD_HOME/Examples/Gem` you find a basic program `gem.C`.
- Open the file with your favourite editor (emacs, vi, pico, ...) and modify the code.
- Compile the program (`make gem`), watch out for compiler warnings/errors.
- Execute the program (`./gem`).

Your Turn...

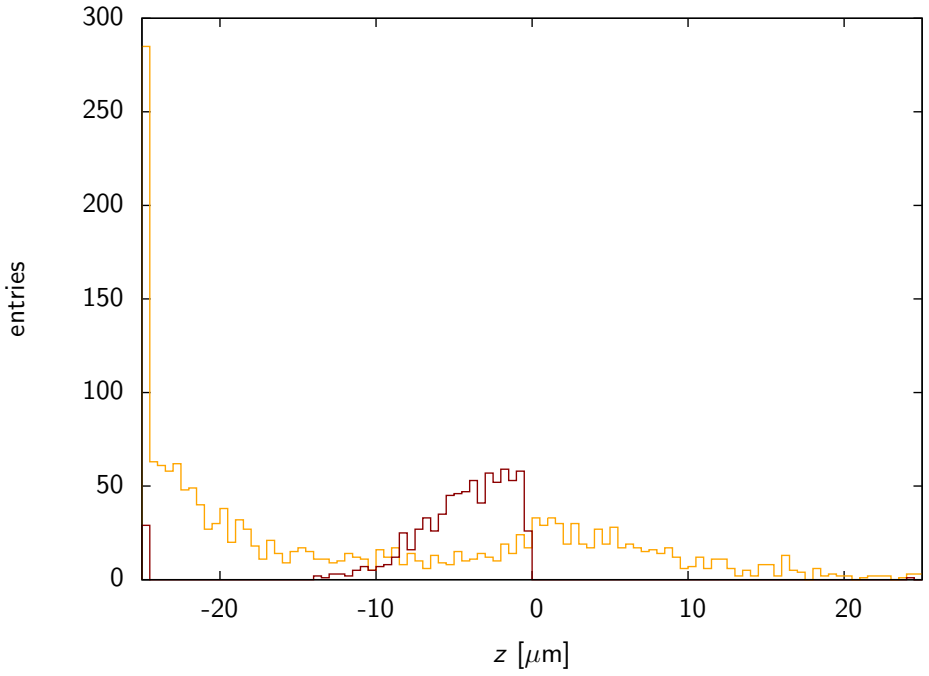
Exercise

- How many electrons/ions are on average produced in the avalanche?
- What is the fraction of ions drifting back to the cathode plane?
- Where on the plastic do the electrons/ions end up (histogram)?

Your Turn...

With a statistics of 1000 avalanches:

- $\overline{n_e} \approx 10, \qquad \overline{n_i} \approx 9$
- Ion backdrift: $\approx 24\%$
- Charge distribution



Questions?

The source code is hosted on a Subversion (svn) repository managed by the CERN Central SVN service.

- Make sure that ROOT is installed.
- Define an environment variable `GARFIELD_HOME` pointing to the directory where the Garfield classes are to be located. If you are using bash, type

```
export GARFIELD_HOME=/home/mydir/Garfield
```

  (replace `/home/mydir/Garfield` by the path of your choice). Add this line also to your `.bashrc` or `.bash_profile`.
- Download ("check out") the code from the repository. For SSH access, give the command

```
svn co svn+ssh://<usrname@>svn.cern.ch/reps/garfield/trunk $GARFIELD_HOME
```

  Alternatively, you can also download the tarballs from `http://svnweb.cern.ch/world/wsvn/garfield`.

- Change to `$GARFIELD_HOME` and compile the classes.

```
cd $GARFIELD_HOME; make
```

  If necessary, adapt the makefile according to your configuration.

- Heed requires an environment variable `HEED_DATABASE` to be set.

```
export HEED_DATABASE=$GARFIELD_HOME/Heed/heed++/database
```

  Add this line to your `.bashrc` as well.

At present, the code is still frequently modified. To get the latest version, use the command `svn update`, followed by `make` (in case of problems, try `make clean; make`).

Field Map
Element Checks

```
ComponentAnsys123* fm = new ComponentAnsys123();
...
// Create histograms for aspect ratio and element size.
TH1F* hAspectRatio = new TH1F("hAspectRatio", "Aspect␣Ratio",
                              100, 0., 50.);
TH1F* hSize = new TH1F("hSize", "Element␣Size",
                       100, 0., 30.);
const int nel = fm->GetNumberOfElements();
double volume;
double dmin, dmax;
for (int i = nel; i--;) {
  fm->GetElement(i, volume, dmin, dmax);
  if (dmin > 0.) hAspectRatio->Fill(dmax / dmin);
  hSize->Fill(volume * 1.e9);
}
TCanvas* c1 = new TCanvas();
hAspectRatio->Draw();
TCanvas* c2 = new TCanvas();
c2->SetLogy();
hSize->Draw();
```

Electron Transport
Energy Distribution

Calculate the electron energy distribution in a flat field ($E = 100$ V/cm):

```
// Setup the gas.
MediumMagboltz* gas = new MediumMagboltz();
...
// Define the geometry (box with half-length 1 cm and half-width 10 um).
SolidBox* box = new SolidBox(0., 0., 0., 1., 1., 10.e-4);
GeometrySimple* geo = new GeometrySimple();
// Add the box to the geometry, together with the medium inside.
geo->AddSolid(box, gas);
// Create a component with constant electric field (100 V/cm along z).
ComponentConstant* component = new ComponentConstant();
component->SetGeometry(geo);
component->SetElectricField(0., 0., 100.);
// Assemble the sensor.
Sensor* sensor = new Sensor();
sensor->AddComponent(component);
...
```

Electron Transport

Energy Distribution

```
// Make a histogram (100 bins between 0 and 1 eV).
TH1D* hEnergy = new TH1D("hEnergy", "Electron energy", 100, 0., 1.);
// Microscopic tracking
AvalancheMicroscopic* aval = new AvalancheMicroscopic();
aval->SetSensor(sensor);
aval->EnableElectronEnergyHistogramming(hEnergy);
// Initial energy
double e0 = 1.5;
const int nEvents = 1000;
for (int i = nEvents; i--;) {
  aval->AvalancheElectron(0., 0., 0., 0., e0, 0., 0., 0.);
  // Draw a new initial energy.
  e0 = hEnergy->GetRandom();
  if (i % 100 == 0) std::cout << i << "/" << nEvents << "\n";
}
// Draw the histogram.
hEnergy->Draw();
```