KDet Sim – a tool to simulate signal in semiconductor detectors

G. KRAMBERGER,

JOŽEF STEFAN INSTITUTE, LJUBLJANA

Simulations of semiconductor devices

TCAD software

Design of semiconductor devices
Simulation of the production process
Simulation of the device performance

Particle physics is a very tinny market, which in addition has very specific requirements:

- radiation damage
- requirement for possible Monte-Carlo simulations
- large structures
- complex ionization patterns

Major packages used for simulation of particle detectors:

- Synopsis TCAD (USA)
- Silvaco (USA/FR)
- Cogenda* (CHI)

*https://github.com/cogenda/Genius-TCAD-Open

The manuals of few thousand pages …
very flexible, but beware that the simulations are tuned to performance of chips.

https://indico.cern.ch/event/452781/contributions/2297596/attachments/1345247/2028743/Kramberger-Vertex2016-Simulations.pdf

What does TCAD do?

Poisson ⇒ $\nabla(\varepsilon_s \nabla U) = -e_0(p - n + \sum P_t N_D - (1 - P_t) N_A) \quad , \quad \vec{E} = -\nabla U$

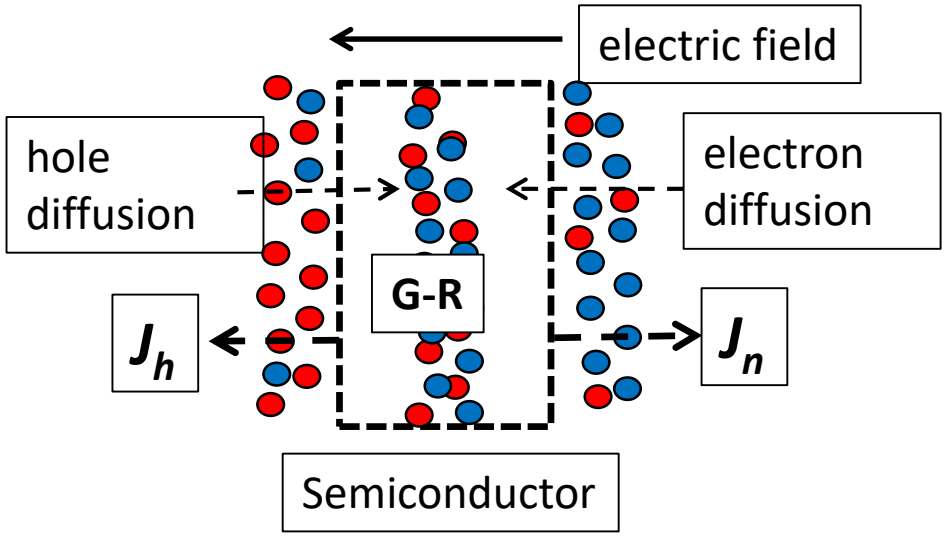
Electron continuity ⇒ $\frac{\partial n}{\partial t} = \frac{1}{q} \nabla . \vec{J}_n + (G - R)$ $\qquad \vec{J}_p = q\mu_p \vec{E} - qD_p \nabla p$

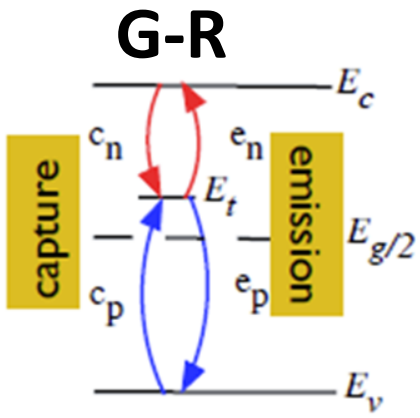
Hole continuity ⇒ $\frac{\partial p}{\partial t} = -\frac{1}{q} \nabla . \vec{J}_p + (G - R)$ $\qquad \vec{J}_n = q\mu_n \vec{E} + qD_n \nabla n$

$$G - R = (n_i^2 - np) \sum_t \frac{c_n c_p}{c_n n + c_p n_i / \chi_t + c_p p + c_n n_i \chi_t} \quad , \quad \chi_t = \frac{1}{1 + \exp(\frac{E_t - E_i}{k_B T})}$$

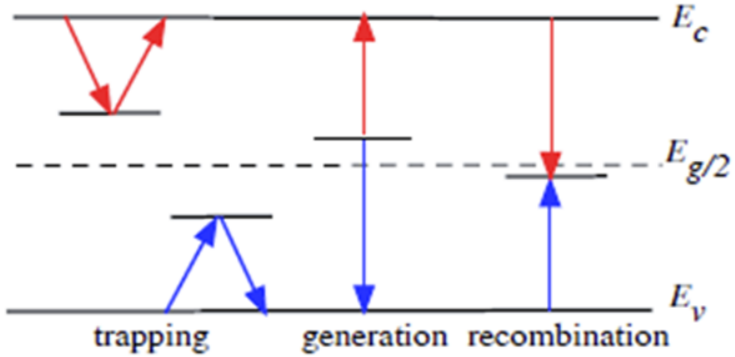
Additional physics processes can be added:

- impact ionization
- trap assisted tunneling
- …





Effects on sensor performance



Where is TCAD less powerful?

http://kdetsim.org/
https://github.com/IJSF9Software/KDetSim

JSI Ljubljana Slovenia
Slovenian Research Agency

BUT:

- Very demanding in terms of CPU and time (4D problem)
- Convergence problems
- Very difficult to do Monte Carlo approach for studying detector properties crucial to particle physics (charge sharing, Lorentz angle, position resolution, …)
- Not so well suited for large multi-electrode systems
- Not easy to include data (e-h distribution) from other packages e.g. GEANT4
- Don't allow fast modeling and fitting of the free parameters to the measurements.
- Although flexible, not so flexible as custom made code

KDetSim is one of the solutions for the "**BUTs**" of TCAD. It is a fast and lightweight ROOT based package for simulation of signal in semiconductor detectors:

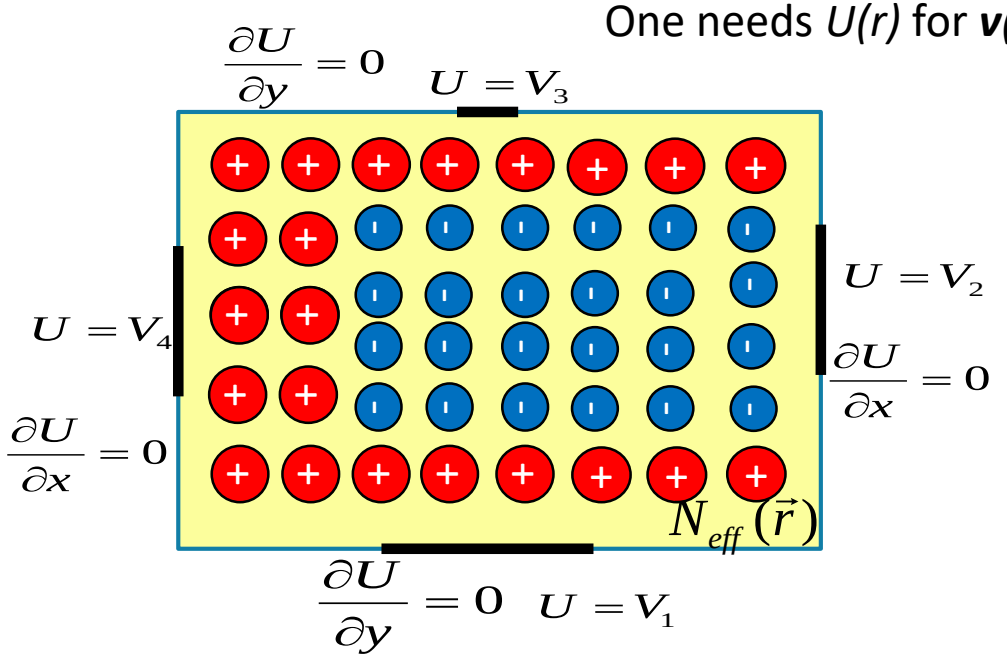
- ROOT interface allows for an easy and standard GUI/IO interface, well integrated with other HEP tools (e.g. GEANT4)
- C++ code in forms of class library is very fast and kind to the computer resources – runs interactively in root
- compile on all OS that run ROOT (Mac, Linux ,Windows, Unix)
- should be easily upgradable/extendable:
  - adding new physics models: mobility (presently 5 different model), impact ionization, radiation damage…
  - new modules – electronics processing of the signal
- extensively used in TCT simulations – direct comparison of the measured and simulated detector response

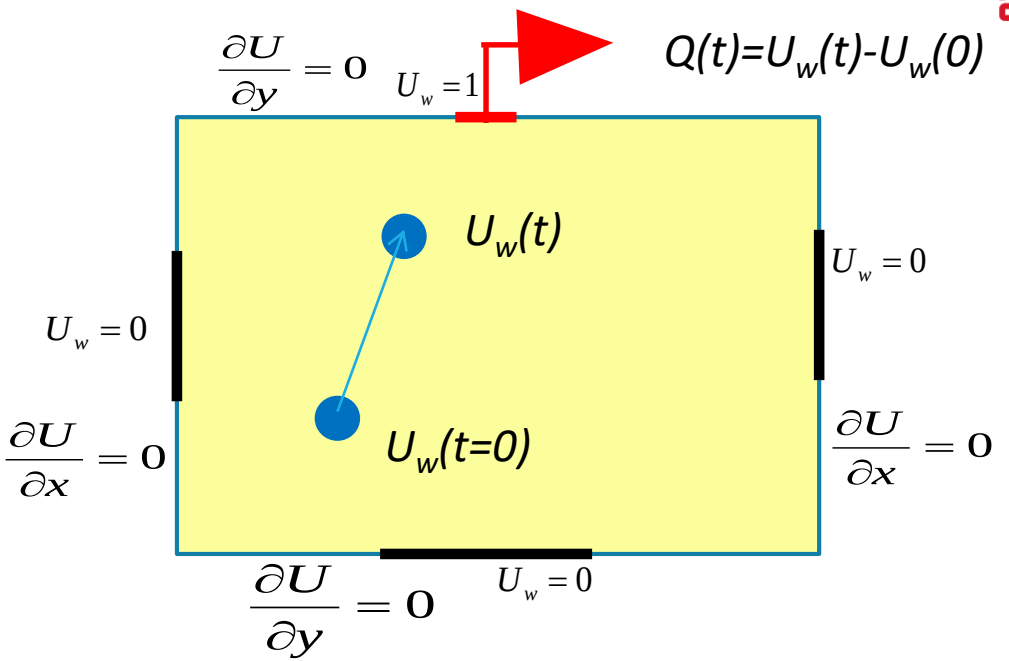
There are other tools developed within RD50:

https://indico.cern.ch/event/456679/contributions/1126330/attachments/1199070/1744044/ComparissonOfSimulators.pdf

Basics – Calculation of Fields (E and E_w)

One needs $U(r)$ for $\boldsymbol{v}(t)$ and U_w for $\boldsymbol{E_w}$!





$$\nabla(\varepsilon(\vec{r})\nabla U(\vec{r})) = -\frac{e_0 N_{eff}(\vec{r})}{\varepsilon_0} \qquad \vec{E} = -\nabla U(\vec{r})$$

$$\Delta U_w(\vec{r}) = 0 \qquad \vec{E}_w = -\nabla U_w(\vec{r})$$

The package doesn't solve continuity/G-R equations in silicon, **but takes $N_{eff}(r)$** as an input!

Ramo-Shockley theorem:

$$I(t) = \frac{dQ(t)}{dt} = q\frac{U_w(r(t+dt)) - U_w(r(t))}{dt} = q\nabla U_w \frac{d\vec{r}}{dt}$$

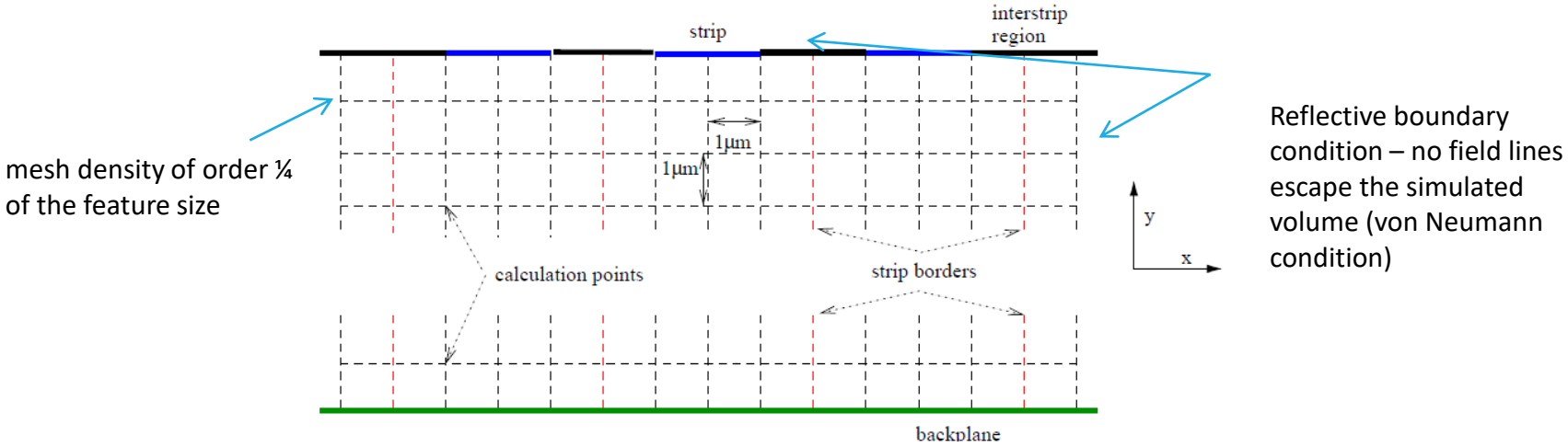
$$I(t) = -q\vec{v}\cdot\vec{E}_w$$

Basics – Calculation of Electric Field

The partial differential equations are solved numerically by using finite difference equation on the mesh (FEM approach):

2D or 3D mesh with complex electrode arrangements/shapes (see examples).

The mesh should be orthogonal but doesn't have to be equidistant (convergence of equation solver is the ultimate judge).

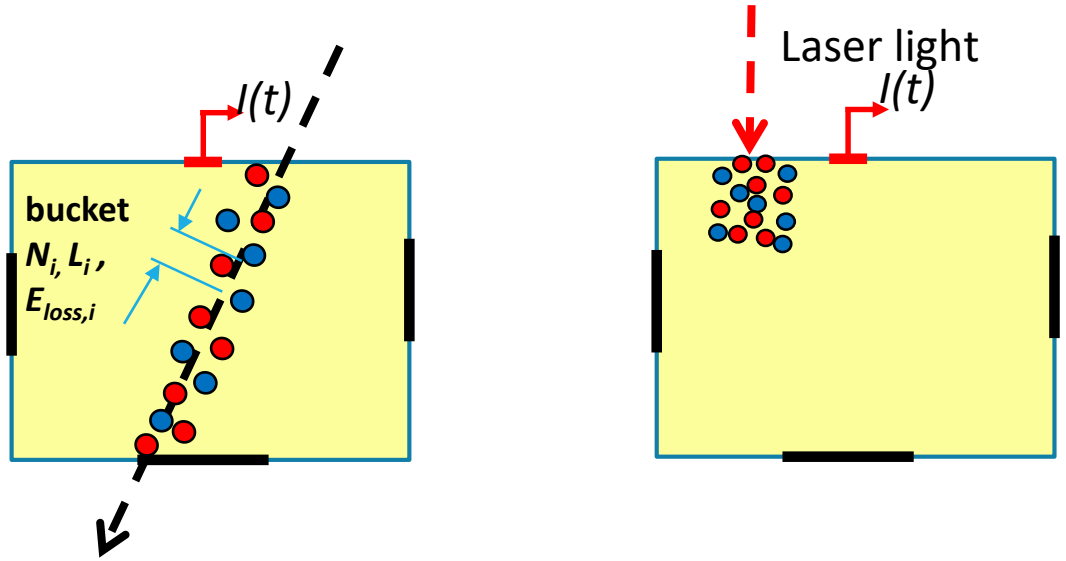


$$\frac{U_{i+1,j,k}-2U_{i,j,k}+U_{i-1,j,k}}{h_i^2}+\frac{U_{i,j+1,k}-2U_{i,j,k}+U_{i,j-1,k}}{h_j^2}+\frac{U_{i,j,k+1}-2U_{i,j,k}+U_{i,j,k-1}}{h_j^2}=-\frac{e_0 N_{eff}(r_{i,j,k})}{\varepsilon\varepsilon_0}$$

The differential equation translates to solving the system of equations for U:

- where every node represents an equation.
- the boundary conditions are essential as they determine the solution of the equations.
- The system of equations is solved by inverting the matrix: 3D structure (Nx*Ny*Nz). The matrix which should be inverted is sparse which significantly speeds up its inverse, so 10^6 node system is solved in the time scale of minutes on Core I7 portable CPU (more RAM helps)

Basics – Charge generation



Generated carriers are grouped in buckets each containing N_i e-h pairs

$$I(t) = \sum_{buckets} N_i \cdot [I_{i,e}(t) + I_{i,h}(t)]$$

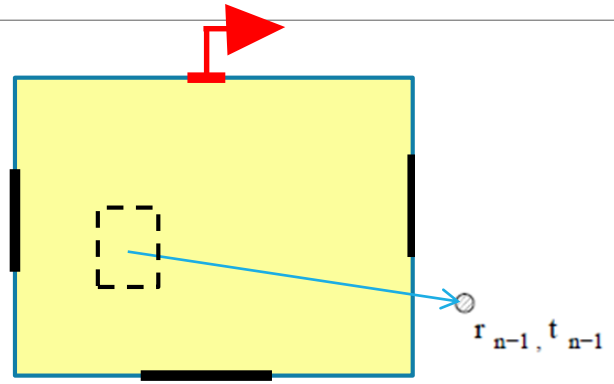
$$N_i = \frac{E_{loss,i}}{E_{ion}}$$ ← Monte-Carlo suitable for GEANT4

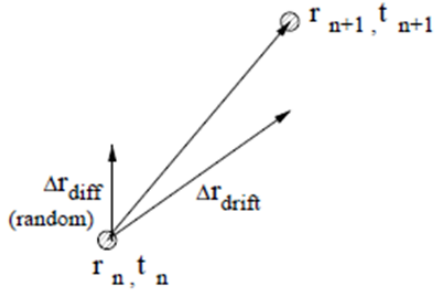
Example of the homogenous distribution of generated e-h pairs in the detector:

$$N_i = \frac{80\text{keV}}{300\,\mu\text{m} \cdot 3.62\text{eV}} = 100\frac{e-h}{\mu\text{m}} \cdot L_i$$ → One can have also different N_i for the same e.g. L_i=1 μm

Different models are already included (Gaussian beam, exponential attenuation of beam, minimum ionizing particle, α), but you can easily make your own function which distributes buckets with arbitrary number of e-h pairs around the sensor

Basics – drift simulation and processes





- **DRIFT + DIFFUSION**

$$\vec{r}_{n+1} - \vec{r}_n = \Delta\vec{r}_{drift} + \Delta\vec{r}_{diff}$$

$$\vec{F} = q(\vec{E} + r_{He,h}\ \vec{E} \times \vec{B})$$

$$\frac{\Delta\vec{r}_{drift}}{\Delta t_n} = \mu_{e,h} \cdot \vec{F}(\vec{r}_n(t_n)) \quad , \quad \Delta\vec{r}_{diff} = \vec{G}aus(\sqrt{2D\Delta t_n})$$

Two approaches:

- set drift distance (simulation drift step – typically 0.1-10 μm) and calculate the required time
- set the time (simulation time step) calculate the drift distance

$$Q_{n+1} - Q_n = q \cdot P(t_n, \tau_{eff}) \cdot [U_w(\vec{r}_{n+1}) - U_w(\vec{r}_n)]$$

PROCESS:

- Trapping of the drifting charge (radiation damage)
- Charge multiplication by impact ionization
- Simulation stopping processes:
  - Finishing drift at the electrode – weighting field =1 or =0
  - Hitting the simulation's volume walls
  - Exceeding the maximum simulation steps
- Anything you may want to put ….

$$dq = \alpha q dx \quad , \quad \alpha_{e,h} = \alpha_{\infty,e,h} \exp\left[-\frac{b_{e,h}}{|E|}\right]$$

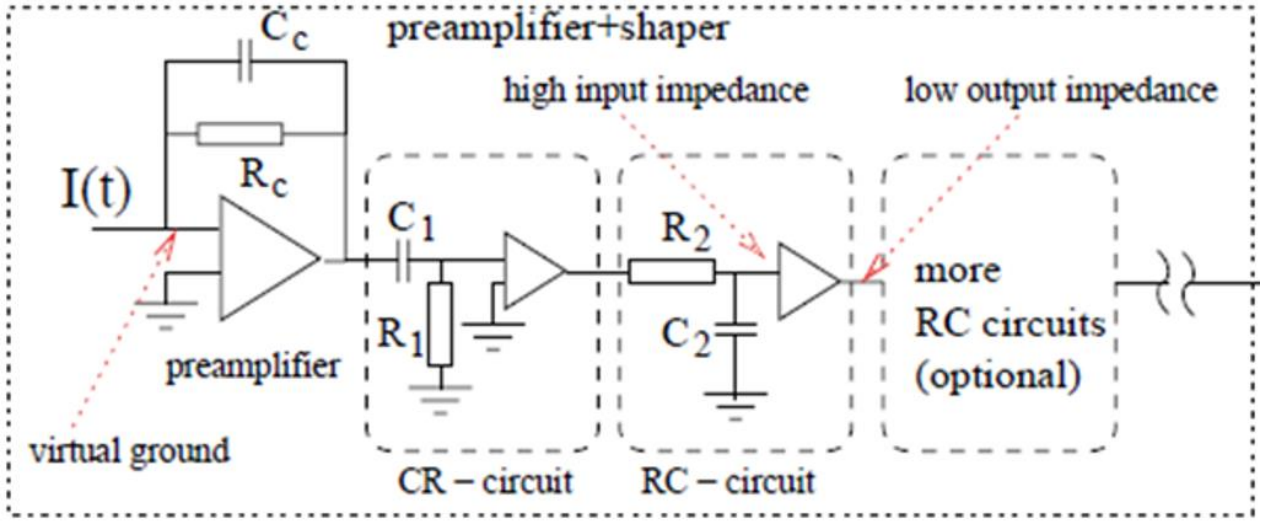
FILLING THE HISTOGRAMS

- Current vs. time for electrons and holes
- Drift path vs. time
- Electric field vs path – allows for offline calculations (e.g. multiplication)

Basics – electronics processing

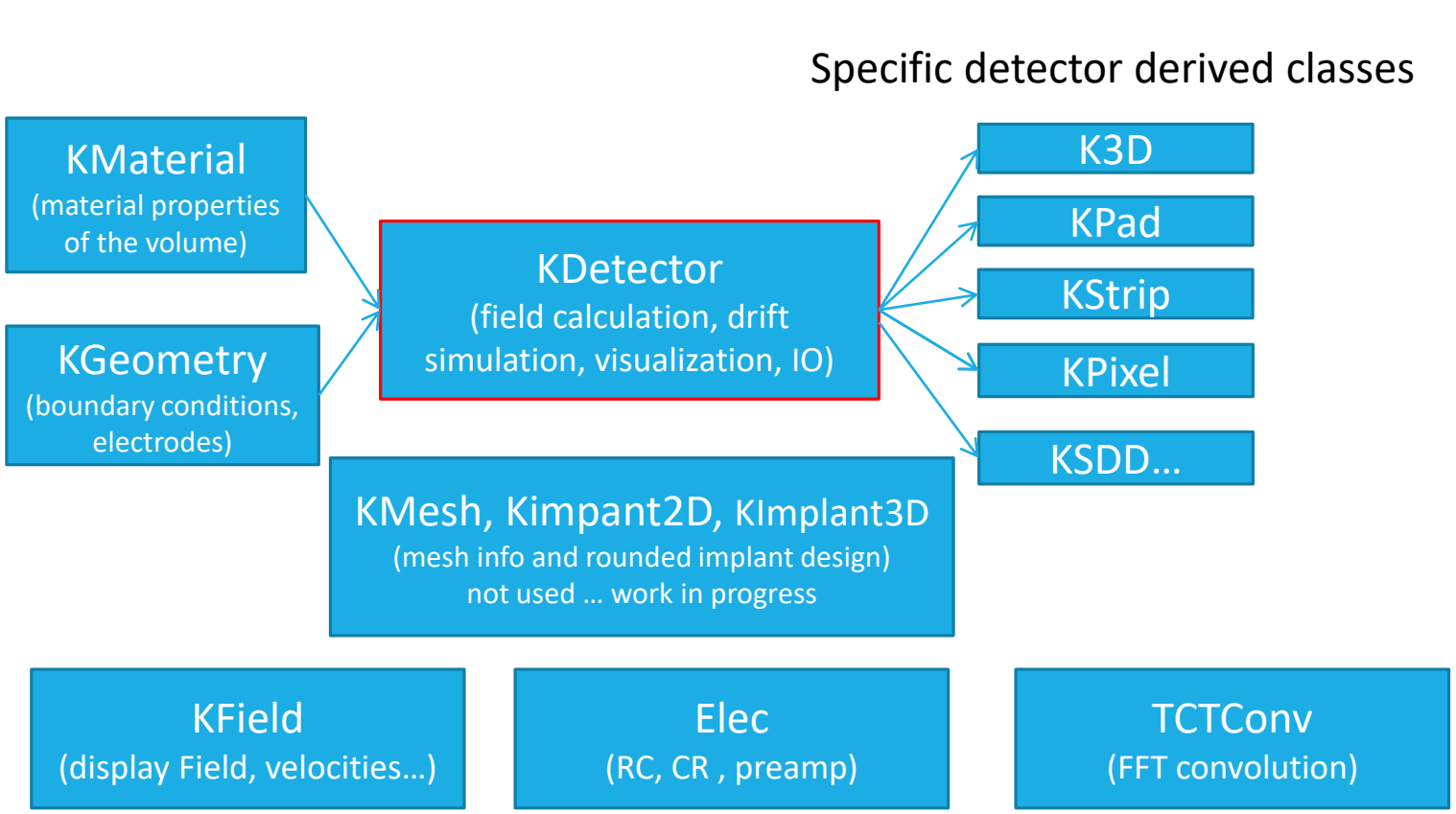
Simulated induced currents can be further processed by electronics (e.g. fed to SPICE simulation):

- Basic electronics models are included:
  - preamp
  - CR, RC filtering / shaping
- Fast Fourier Transform Tools - Processing the induced current in the electrode by electronics (integration/amplification, shaping)



Structure and running the library

The library is a single .dll, .sl which is loaded in the ROOT framework



KDetSim

compiles on all systems (Windows/Mac/Linux)
precompiled dll/sl library can be used

The simulation is typically written in form of **ROOT macros** (examples follow), so a full root machinery can be employed for presentations/fitting analysis – no GUI needed

Building an example

Building a simulation (strip detector)

Everything starts with initialization of KDetector class which invokes creation of geometry/material/space charge – basically matrixes of Nx*Ny*Nz integer numbers – 32 bit (TH3I)

Sensor geometry (all electrodes are assumed to be connected to low impedance)

- bit 0 – sensing material material
- bit 1 – GND electrode
- bit 2 – HV electrode
- **bits 3:13 – reserved for setting the boundary condition**
- bit 14 – the electrode for which the U_w is calculated – sensing electrode
- bit 15 – other voltages (yes=1, no=0):
- bits 16:32 – electrode at potential defined by index given by the number with bits [16:32]

Detector material (value at the node is detector material index)

- 0 = silicon
- 1 = Polysilicon
- 2 = Silicon oxide
- 10 = diamond
- 20 = air
- 100 = aluminum

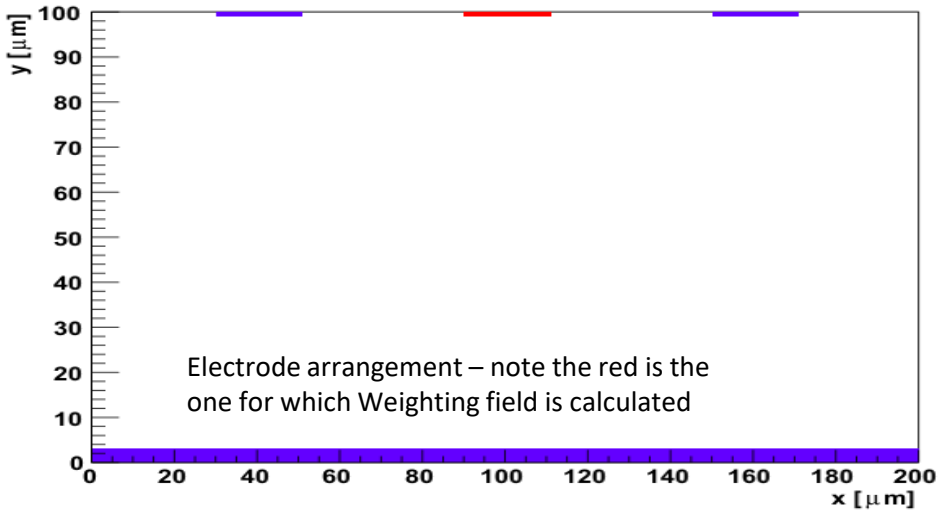
Space charge distribution (remember that does not come from SRH calculation)

- Function (TF3) or matrix with nodes (TH3I) Neff(x,y,z) – Neff is in units [1e12 cm^{-3}]

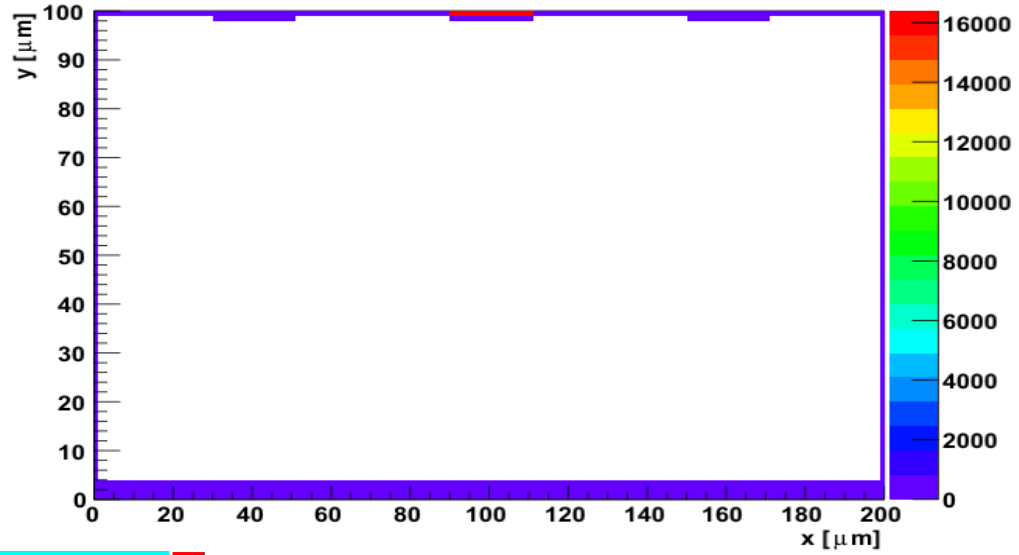
```
{
 KDetector det;
det.Voltage=500;
det.nx=200;
det.ny=100;
det.nz=1;
det.EG=new TH3I("EG","EG",det.nx,0,200,det.ny,0,100,det.nz,0,1);
det.EG->GetXaxis()->SetTitle("x [#mum]");
det.EG->GetYaxis()->SetTitle("y [#mum]");
det.EG->GetZaxis()->SetTitle("z [#mum]");

det.DM=new TH3I("DM","DM",det.nx,0,200,det.ny,0,100,det.nz,0,1);
det.DM->GetXaxis()->SetTitle("x [#mum]");
det.DM->GetYaxis()->SetTitle("y [#mum]");
det.DM->GetZaxis()->SetTitle("z [#mum]");

TF3 *f2=new TF3("f2","x[0]*x[1]*x[2]*0+[0]",0,3000,0,3000,0,3000);
f2->SetParameter(0,-2);
det.NeffF=f2;

//BackPlane
Float_t BackPos[3]={100,1,0.1};
Float_t BackSiz[3]={100,1,0.1};
det.ElRectangle(BackPos,BackSiz,2,0);

//Strips
Float_t BackPos[3]={40,100,0.1};
Float_t BackSiz[3]={10,1,0.1};
for(Int_t i=0;i<3;i++)
 {
  BackPos[0]=i*60+40;
  if(i==1)
        det.ElRectangle(BackPos,BackSiz,16385,0);
  else
        det.ElRectangle(BackPos,BackSiz,1,0);
 }
}
```

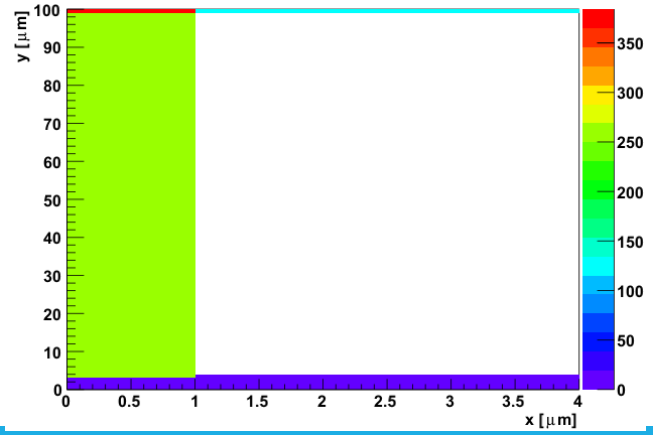
Building a simulation (strip detector cont.)

det.Draw("G").Draw("COLZ");



det->SetBoundaryConditions(); // set boundary conditions
det.Draw("G").Draw("COLZ");

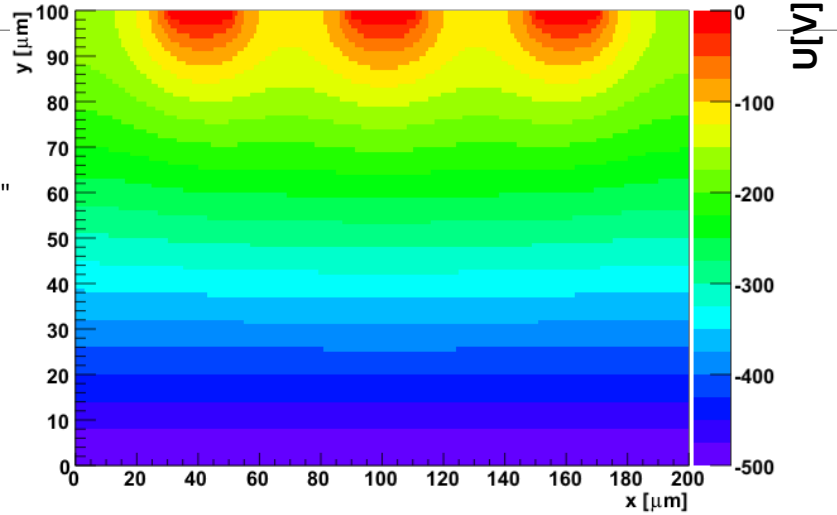


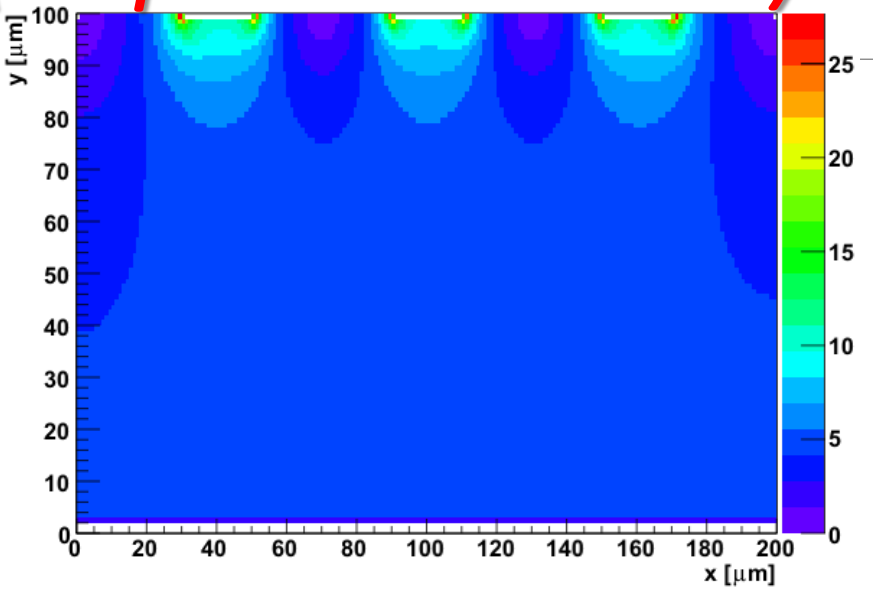


additional “colors” are BC

Building a simulation (strip detector cont.)

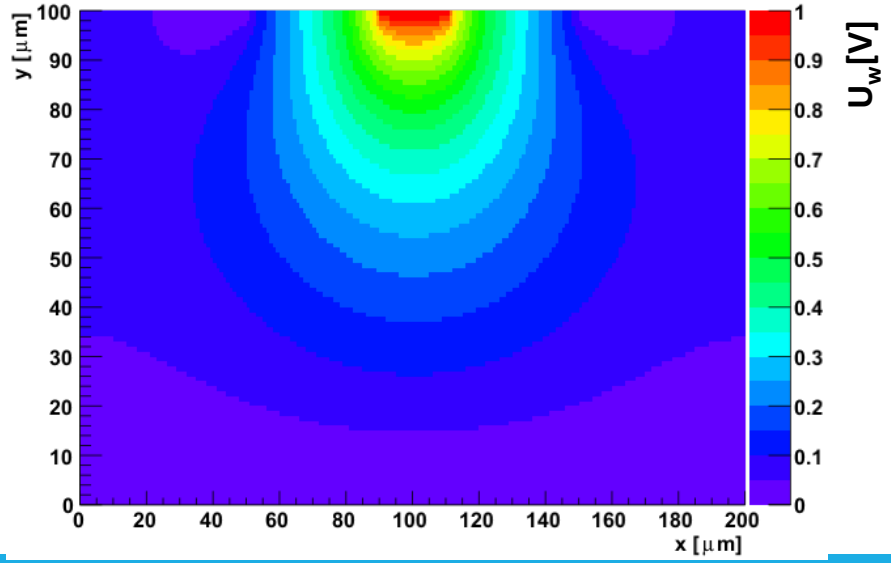
det->CalField(0);
det->CalField(1);
det.Draw("EP").Draw("COLZ"

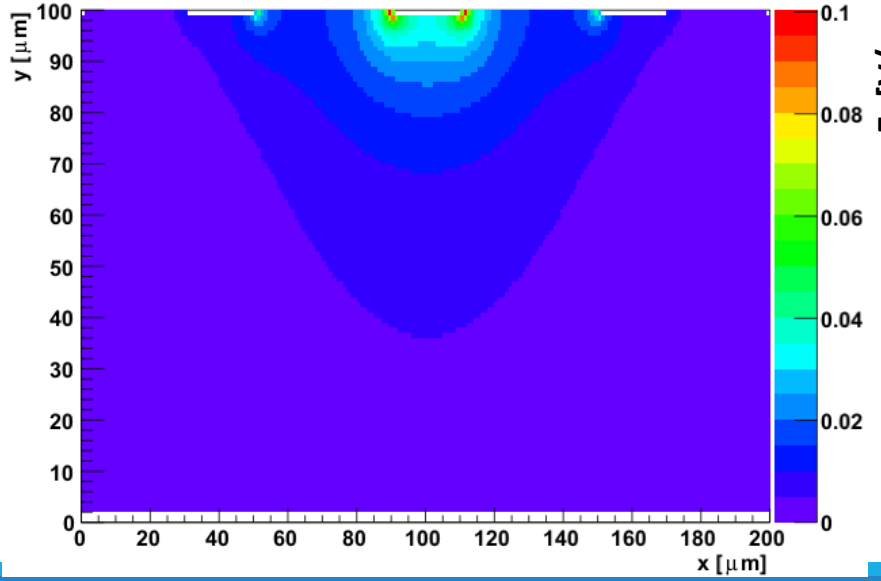




det.Draw("EF").Draw("COLZ");

det.Draw("WP").Draw("COLZ");

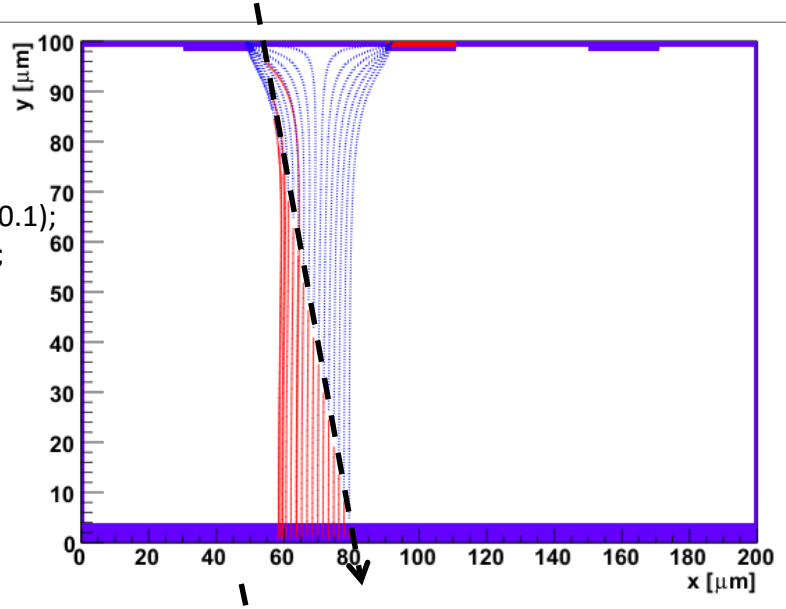


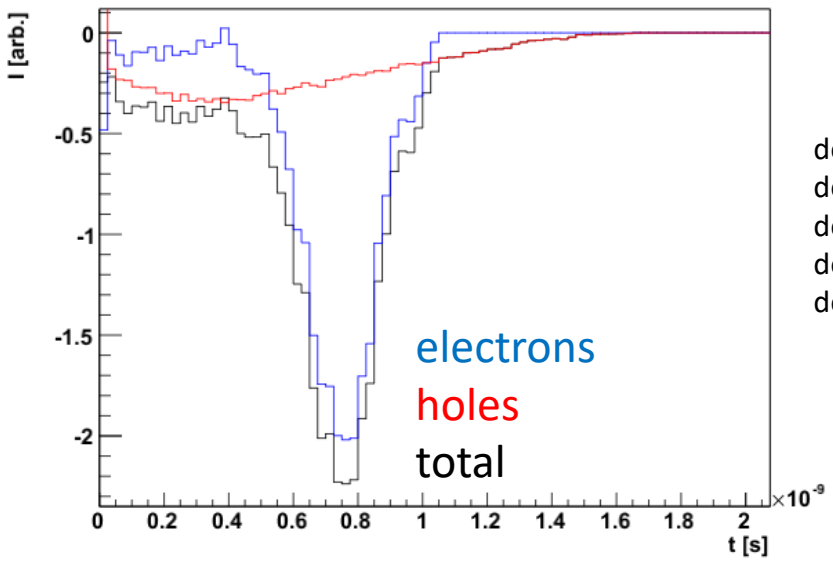


det.Draw("WF").Draw("COLZ");

Building a simulation (strip detector cont.)

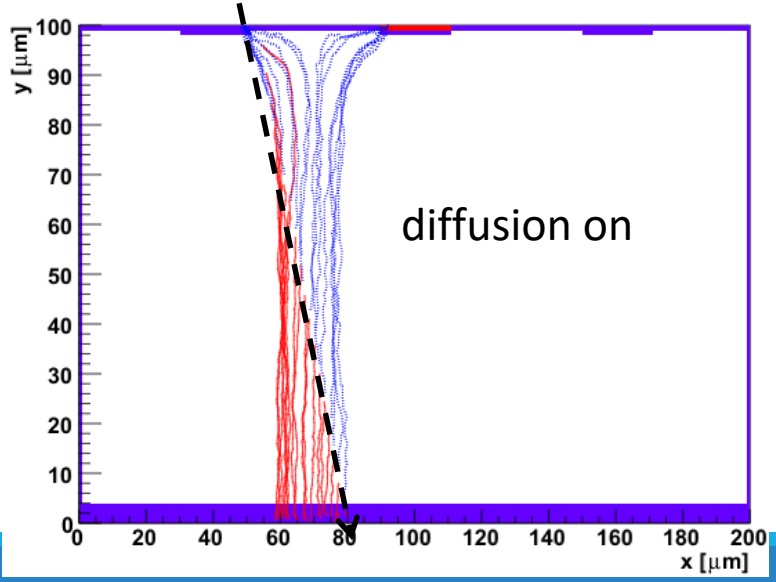
```
det->SetEntryPoint(50,100,0.1);
det->SetExitPoint(80,1,0.1);
det->ShowMipIR(20);
```

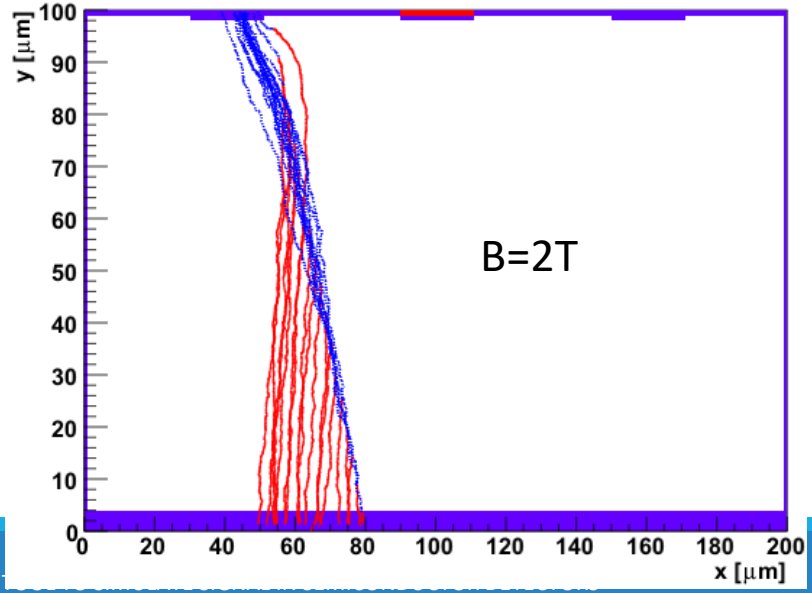





```
det.SetDriftHisto(5e-9); det.Sstep=0.1;
det.MipIR(100);
det.sum.Draw()
det.neg.Draw("SAME");
det.pos.Draw("SAME");
```

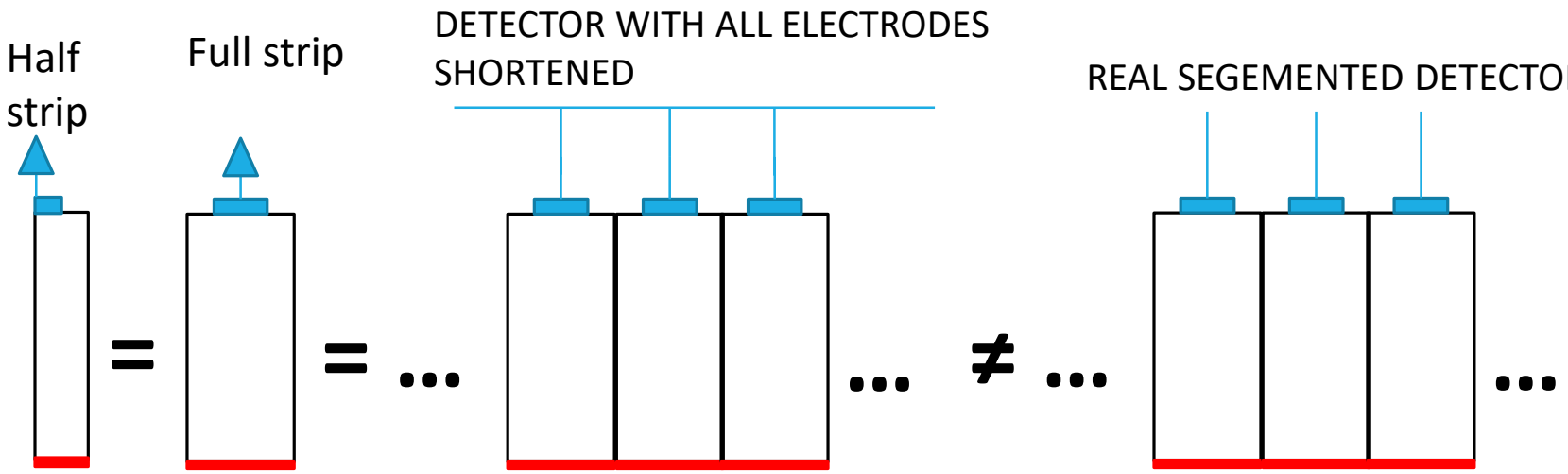
```
det->diff=1;
```






```
det->B[2]=2e-4;
```

Choice of boundary conditions



All neighboring electrodes also with U_w=1 –> all strips shortened

All neighboring electrodes at U_w=0 –> individual channels

Unlike for electric field where for the symmetry reasons only a half strip can be used to calculate the field one should simulate a much larger section for the weighting field.

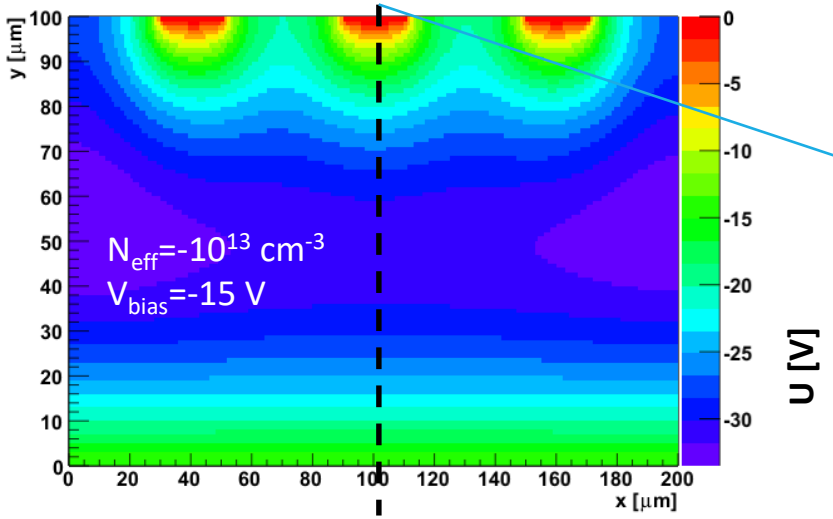
A lot of effects in irradiated silicon detectors – such as e.g. “trapping induced charge sharing” can not be simulated without proper weighting field.

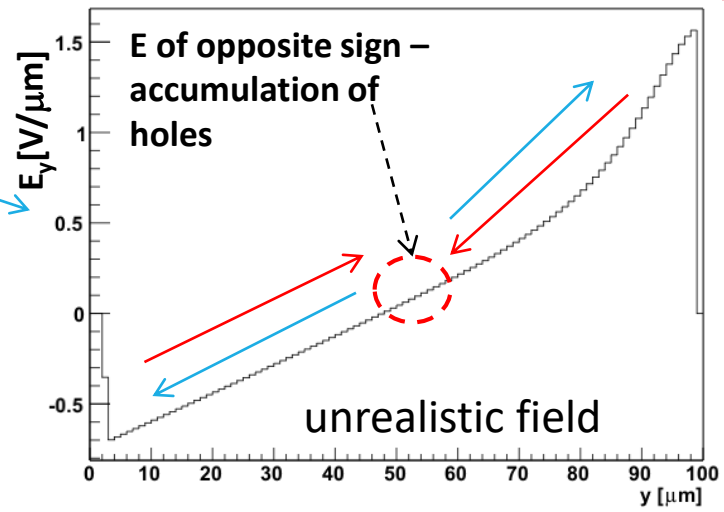
Un-depleted silicon and dynamic field adjustment

The solver doesn't solve whole set of equations – so N_{eff} should be correctly adjusted

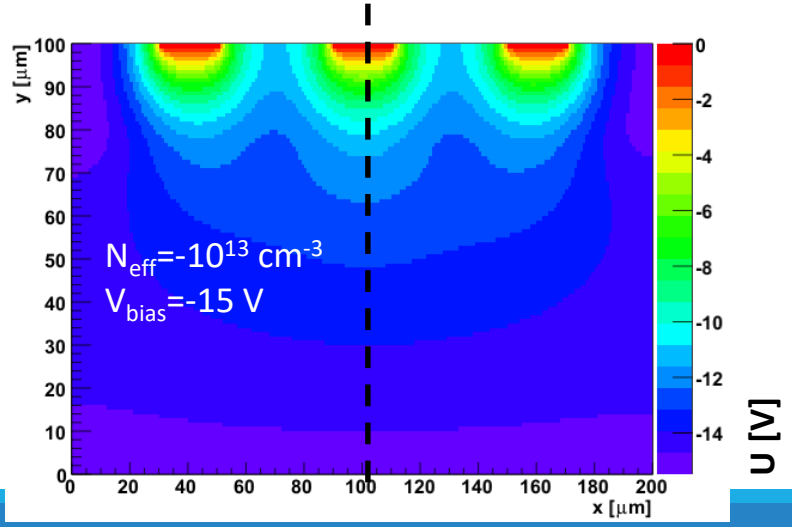
DYNAMIC field adjustment (a simple solution to 4D problem)

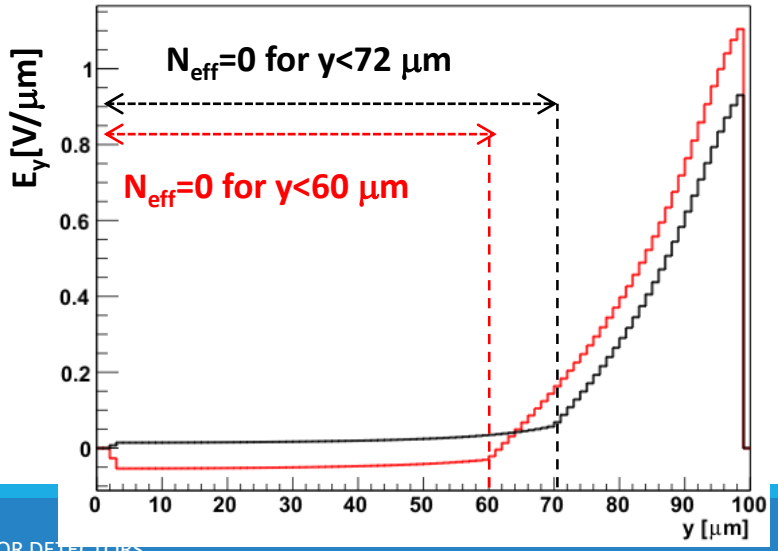
In each drift step you can calculate amount of trapped charge its distribution, recalculate the field and continue the drift of the remaining charges





Iterative method setting the point of un-depleted bulk with the requirement of monotonous field -> e.g. bisection or even more complex algorithms.





Meshing-"Stitching" of the E and E_w

- Calculation of the electric field is done for the smallest segment exploiting the symmetry (KDetector d1;) – fast and precise calculation on smallest symmetry element due to small number of nodes
- Calculation of the weighting field is done for a full segment taking connected electrodes into account (KDetector d2;) – the meshing of that can be much coarser as the electrode shape effects are smaller/negligible for U_w
- Drift is done for d2 where each drift step is calculated with electric field from d1;

Several advantages in terms of:

- Time needed for calculation or/and
- Larger density of meshing points for the same calculation time
- Smaller storage required for saving the fields

Geometry design

Several shapes are predefined in the software and can be used to draw electrodes
- Column (3D)
- Box (3D)

These are used to define several standard detector types
- Pad (width, thickness)
- Strip (pitch, strip width, electrode thickness, segment size, thickness)
- Pixel (number of pixels, X,Y,Z dimensions)
- 3D (number of circular columns, X,Y,Z dimensions)

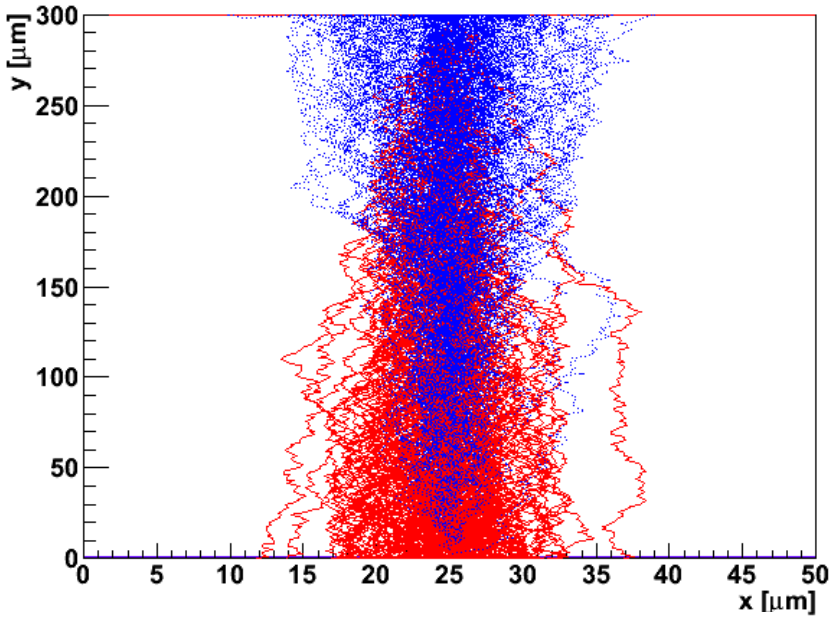
Mesh size determines the edges of electrodes (rounded to the nodes) - fields can be too high
- Large effect on multiplication of charge carriers
- Small effect on shape of the induced current (saturated velocities in high fields and short distances)

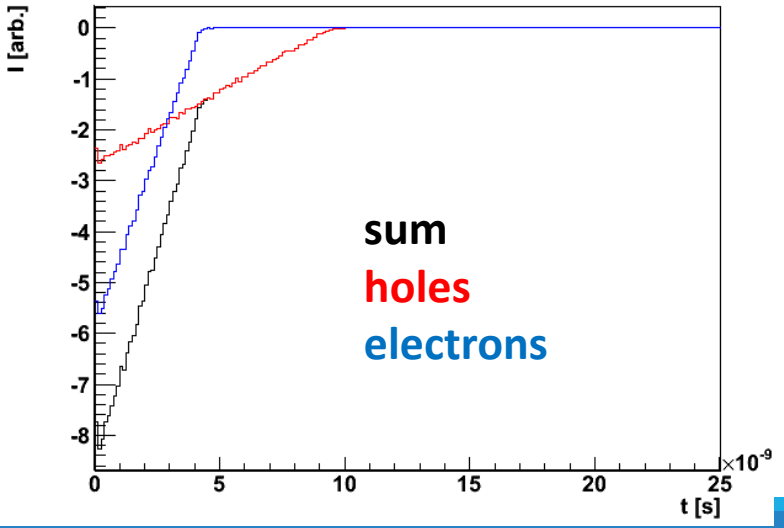
2D - examples

Examples of simulation – pad detectors

Simple example run as a root script:

```
{
 TF1 *neff=new TF1("neff","[0]",0,1000);
 neff->SetParameter(0,1);    // set Neff [1e12 cm-3]
 KPad det(50,300);           // dimensions of the sample
 det->Neff=neff;
 det->Voltage=-200;          // Set voltage
 det->SetUpVolume(1);        // Setup electrodes
 det->SetUpElectrodes();

 TCanvas c1;
 det->SetEntryPoint(25,299.9,0.5); // define track entry point
 det->SetExitPoint(25,1.,0.5);     // define track exit point
 det->Temperature=253;             // set temperature
 det->diff=1;                      // switch on diffusion
 det->ShowMipIR(200);              // draw drift paths for
                                   //  200 buckets

 TCanvas c2;
 det->MipIR(200,1);                // simulate mip 200 bucket
 det->sum->Draw();                 // draw current
 det->pos->Draw("SAME");           // draw current for holes
 det->neg->Draw("SAME");           // draw current for electrons
}
```



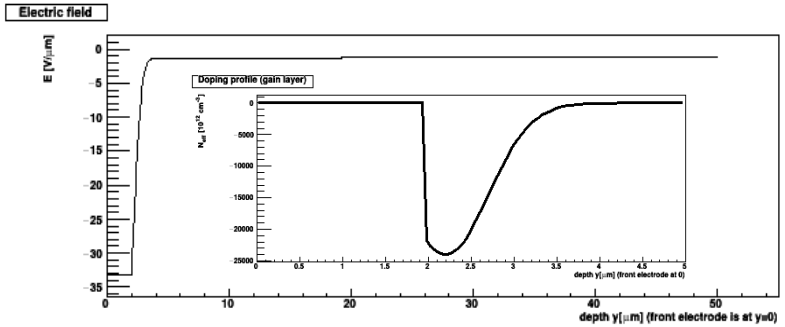


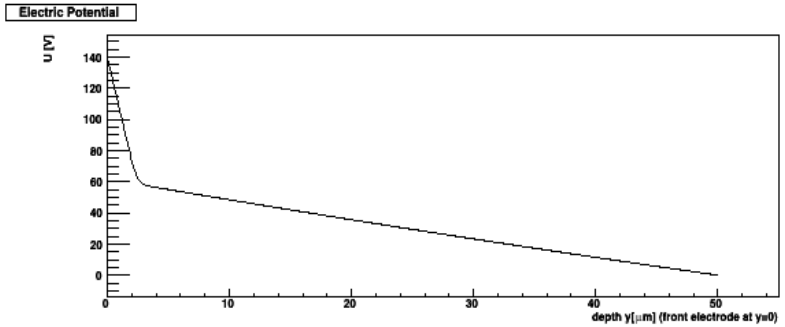
p-n pad detector/diode

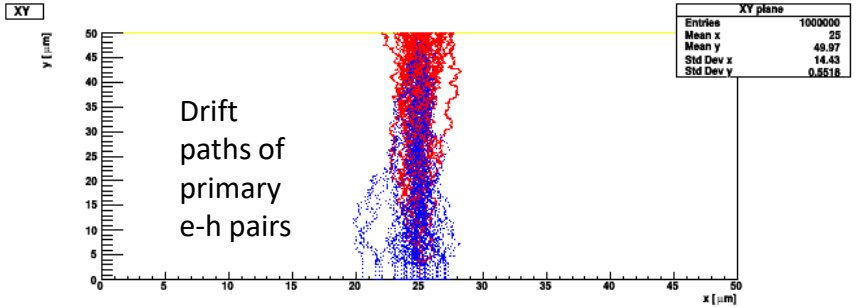
LGAD example

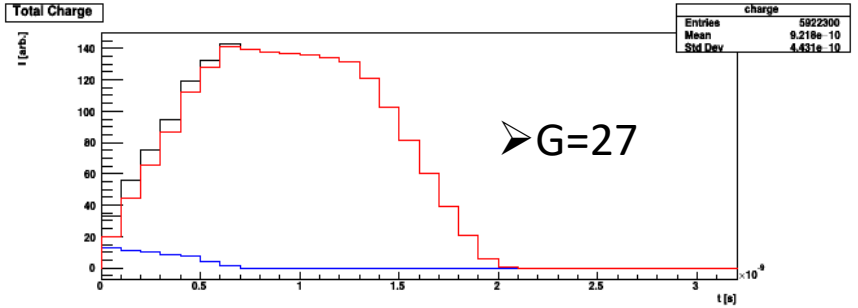
- Example of δ-ray (1/3 of total energy deposit) generation and its impact of induced current shape – Landau fluctuations – in timing applications
- Drift paths of secondary holes are not shown.

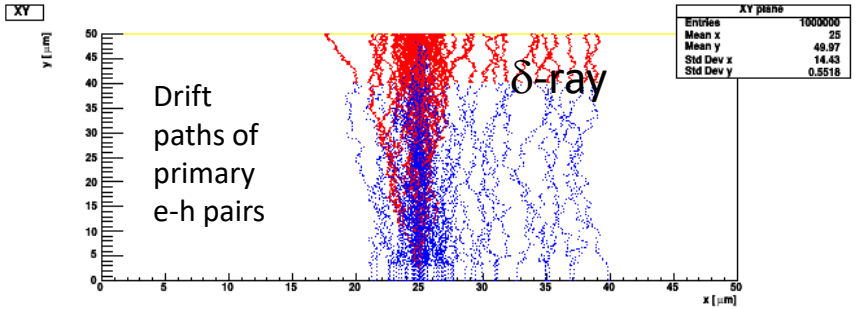
n+-p+-p detector

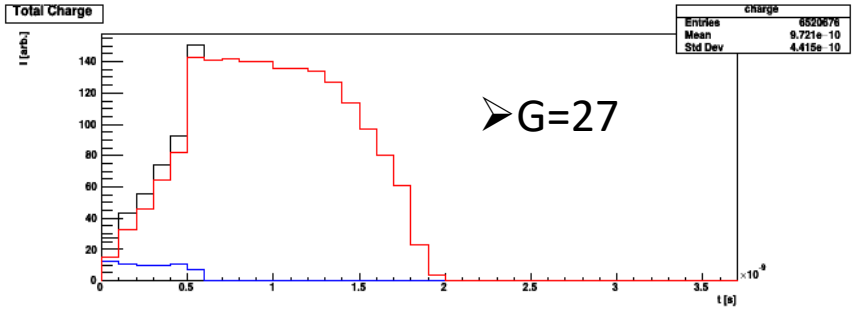








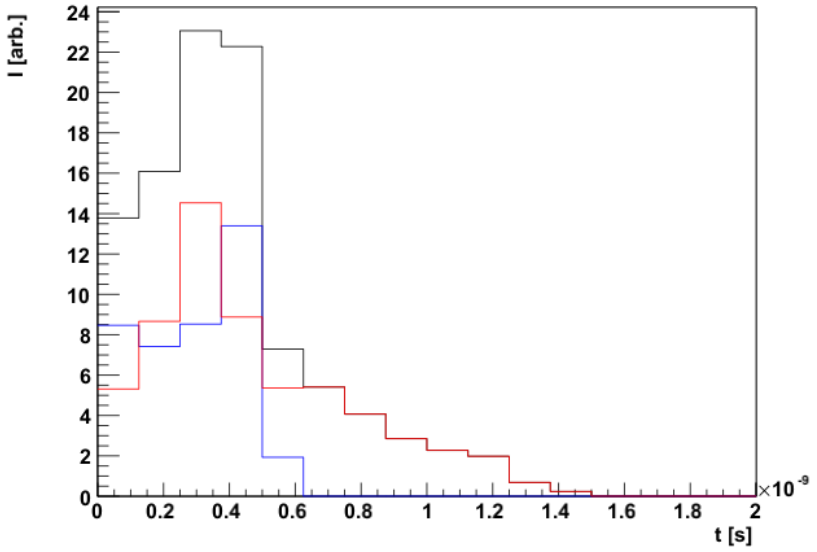


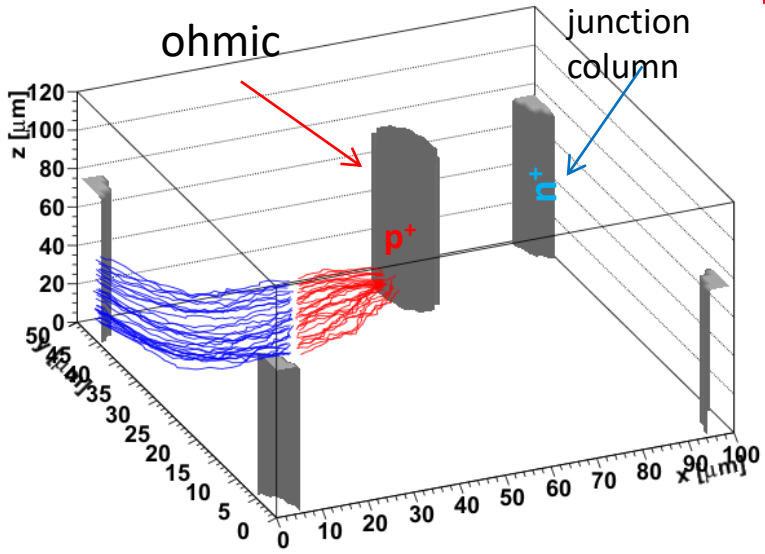


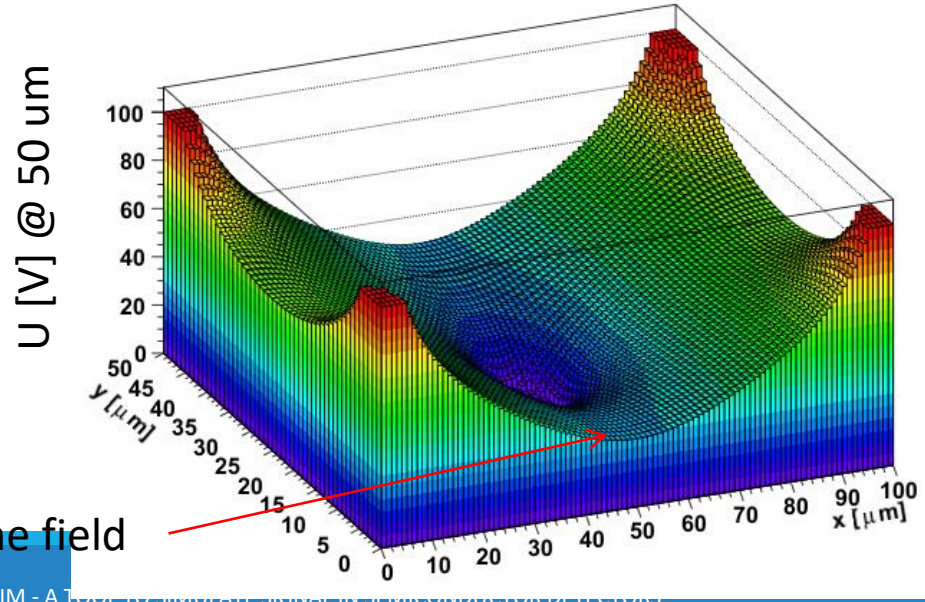
3D - examples

3D detector

```
{
 gStyle->SetCanvasPreferGL(kTRUE);

 // define a 3D detector with 5 electrodes
 // x=100 , y is 50 and thickness 120
 K3D *det=new K3D(5,100,50,120);
 // define the voltage
 det->Voltage=100;
 // define the drift mesh size and simulation mesh size in microns
 det->SetUpVolume(1,1);
 // define  columns #, postions, weigthing factor 2=0 , material Al=1
 det->SetUpColumn(0,0,0,5,75,2,1);
 det->SetUpColumn(1,100,0,5,75,2,1);
 det->SetUpColumn(2,0,50,5,75,2,1);
 det->SetUpColumn(3,100,50,5,75,2,1);
 det->SetUpColumn(4,50,25,5,-75,16385,1);
 Float_t Pos[3]={100,50,1};
 Float_t Size[3]={100,50,2};
 det->ElRectangle(Pos,Size,0,20);
 det->SetUpElectrodes();
 det->SetBoundaryConditions();
 //define the space charge
TF3 *f2=new TF3("f2","x[0]*x[1]*x[2]*0+[0]",0,3000,0,3000,0,3000);
 f2->SetParameter(0,-2);

 det->NeffF=f2;
 det->CalField(0); // calculate weigting field
 det->CalField(1); // calculate electric field

 // set entry points of the track
 det->enp[0]=30;   det->enp[1]=30;   det->enp[2]=50;
 det->exp[0]=30;   det->exp[1]=30;   det->exp[2]=10;

 // switch on the diffusion
 det->diff=1;
 // Show mip track
 TCanvas c1; c1.cd();
 det->ShowMipIR(30);
 // Show electric potential
 TCanvas c2; c2.cd();
 det->Draw("EPxy",60).Draw("COLZ");
 // calcualte induced current
 TCanvas c3; c3.cd();
 det->MipIR(100);
 det->sum.Draw();   det->neg.Draw("SAME");   det->pos.Draw("SAME");
}
```



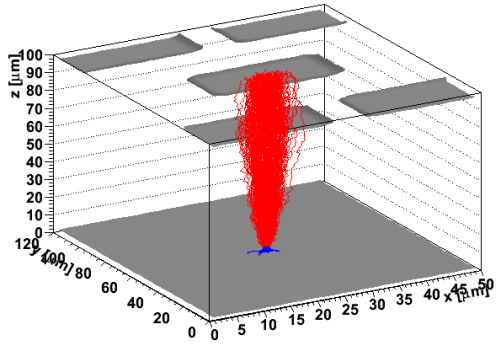


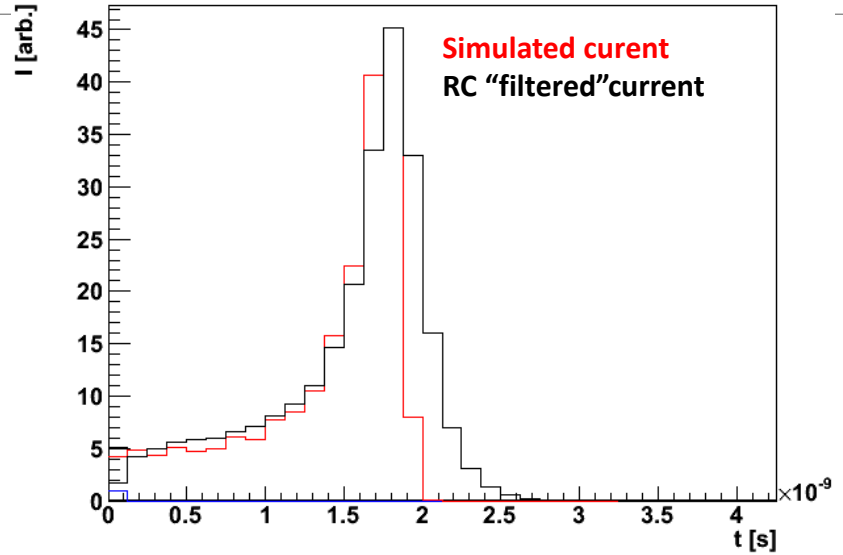


saddle in the field

Examples - Pixel sensor

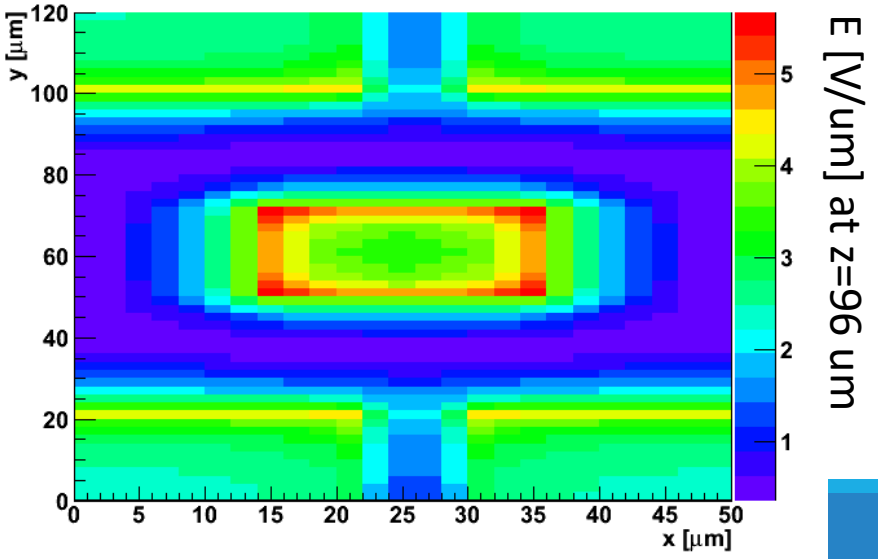
TCT generation of red light






```
{
 KPixel *det=new KPixel(5,50,120,100); // 5 pixels, x=50 um,y=120 um, z =100 um
 det->Voltage=200;                     //  setup the voltage
 det->SetUpVolume(2,2,1);              //  2 um step for calculation of field in x,y and 1 um in z
 det->SetUpPixel(0,25,60,10,10,2,16385); // setup the center pixel for readout
 det->SetUpPixel(1,10,10,10,10,2,1);     // setup other pixels
 det->SetUpPixel(2,40,10,10,10,2,1);
 det->SetUpPixel(3,40,110,10,10,2,1);
 det->SetUpPixel(4,10,110,10,10,2,1);
 det->SetUpElectrodes();               // init all electrodes
 det->SetBoundaryConditions();         // setup boundary conditions
 TF3 *f2=new TF3("f2","x[0]*x[1]*x[2]*0+[0]",0,3000,0,3000,0,3000);
 f2->SetParameter(0,-2);               // space charge distribution
 det->NeffF=f2;
 det->CalField(0);  det->CalField(1);  //calculated fields
 det->diff=1;                          //diffusion is on
 det->enp[0]=25;   det->enp[1]=60;   det->enp[2]=1;
 det->exp[0]=25;   det->exp[1]=60;   det->exp[2]=3;
 det->MipIR(200,3);                    // mip simulations
 det.sum.DrawCopy(); det.neg.Draw("SAME"); det.pos.Draw("SAME");
 //electronics processing - RC filter
 Elec el(3e-12);                       // init electronics class
 el->preamp(det.sum);                  // RC filtering
 det->sum.Scale(det.pos.GetMaximum()/ det->sum.GetMaximum());
 det->sum.Draw("SAME");
}
```

Electric field strength



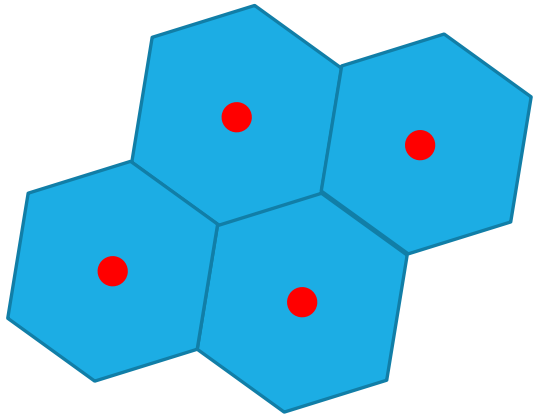
Hexagon 3D

A BNL design for gamma ray detector

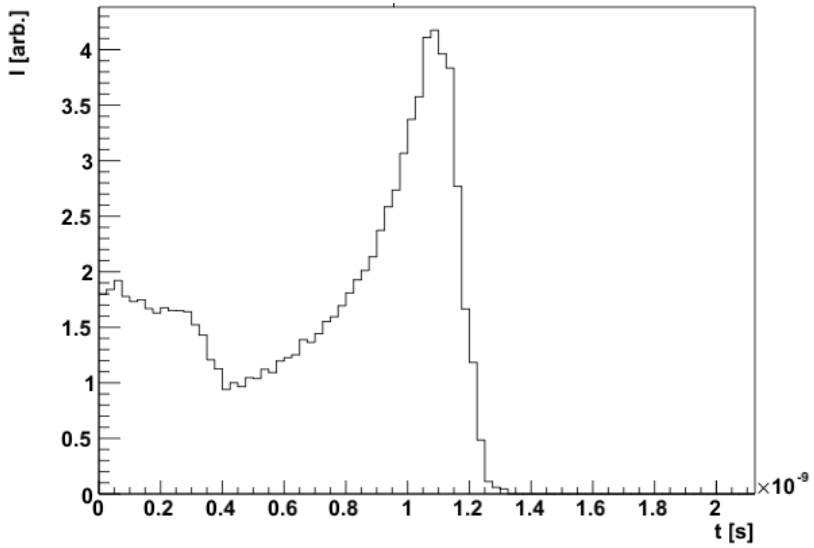
central collecting electrode is completely isolated by hexagon – no cross talk to neighbors

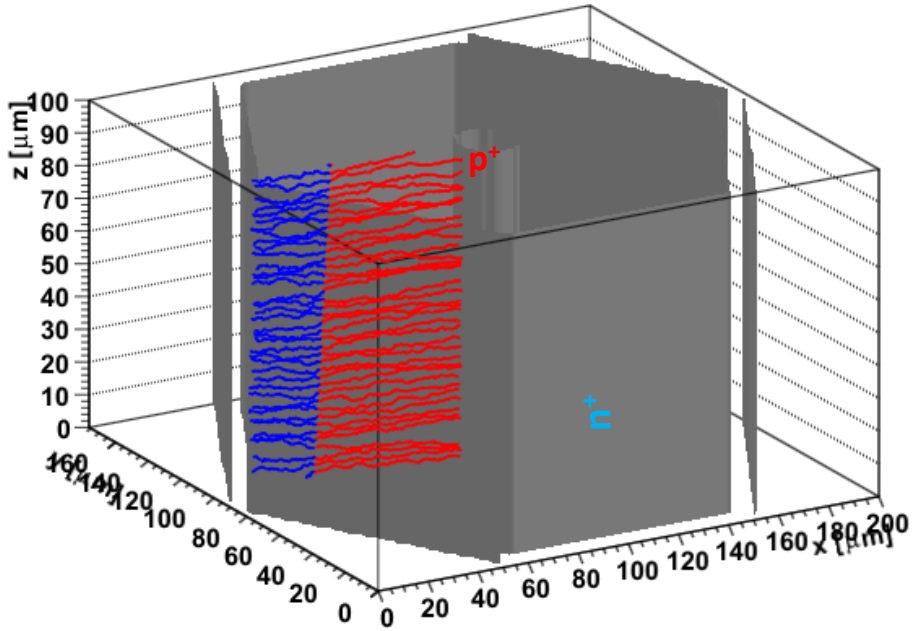
Very short:

- collection time
- full depletion voltage

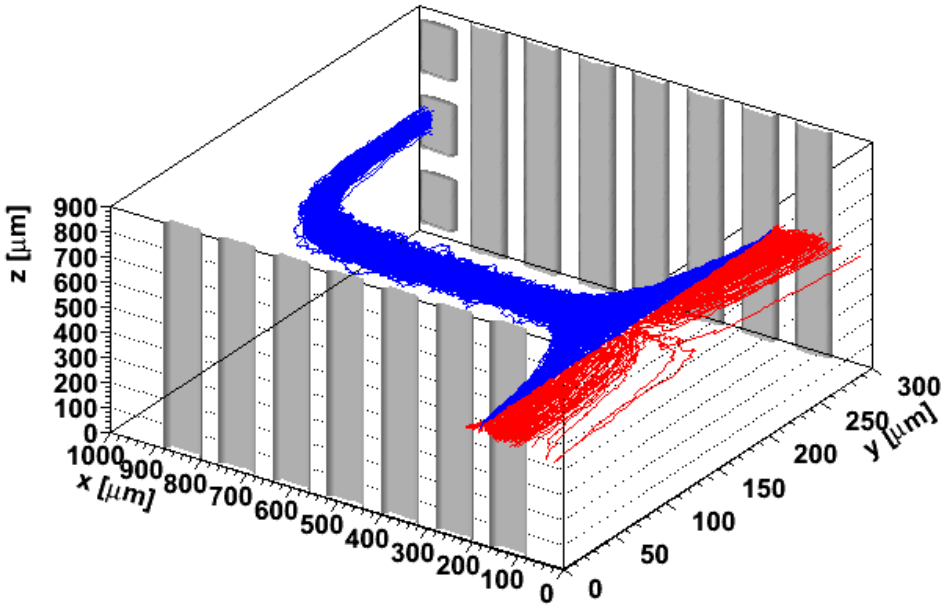


at 500V





Silicon drift detector – 3D



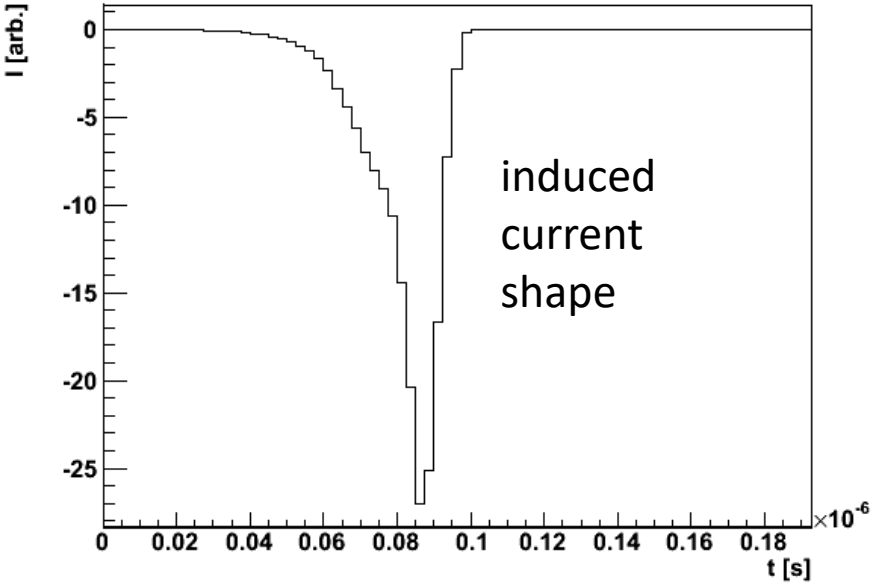
3D simulation of the ALICE SDD geometry

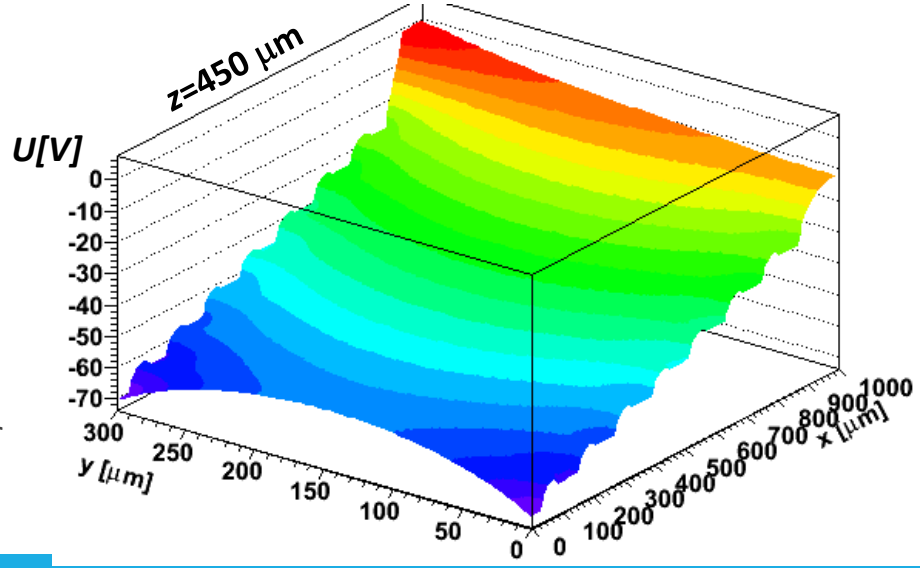
A segment of three anodes

- 300 μm pitch
- 200 μm implant width

Similar results as for 2D but some simulations require 3D simulation

A map of potential along the drift plane at *z*=450 μm similar to 2D simulation.





Conclusions

- KDetSim is fast, lightweight and highly portable ROOT based code for simulation of signal in semiconductor detectors in 2D and 3D.
- It is a complementary to full TCAD simulations.
- It has been adopted also for simulations different semiconductor structures (from few tens ps to μs durations) enabling studies of
  - charge sharing/crosstalk
  - trapping – charge collecting efficiency
  - resolution for inclined tracks
  - …

There are several tasks that are going on:

- **Import of E fields from TCAD into the simulator** – an ongoing RD50 project
- Possible parallelization of the calculation (not sure if required?)
- **writing manual is ongoing … - now poorly documented**

Anyone interested is welcome to join also not only for using but also to writing/debugging/improving code…