TRACKS - MC generation and reconstruction with multiple scattering and noise

Mattia Ivaldi, Luca Quaglia

Master's Degree in Nuclear, Subnuclear and Biomedical Physics

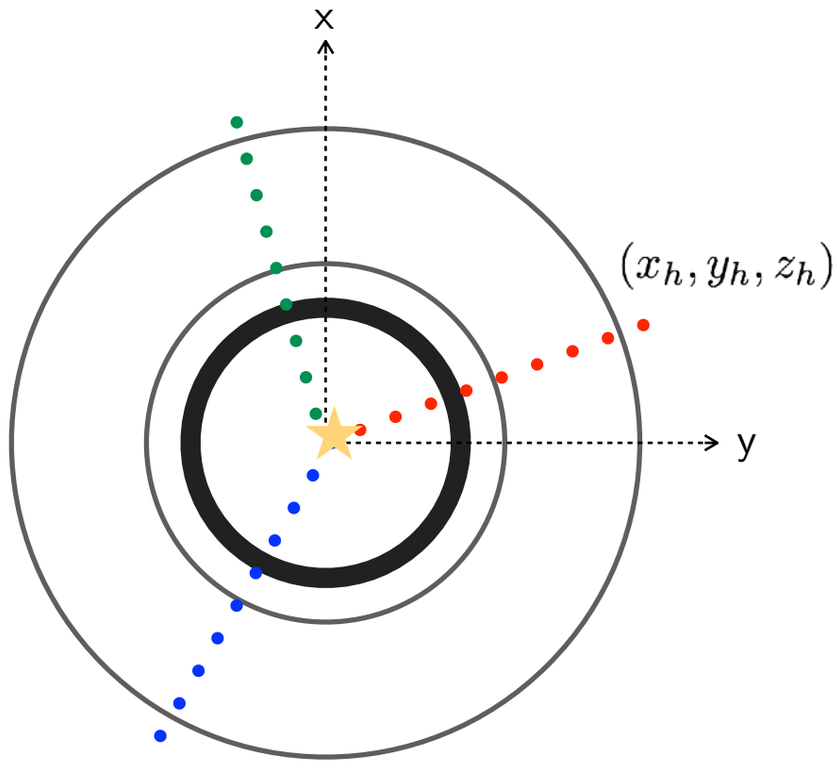
Numeric Analysis and Simulation Technology

2018/2019

Code and documentation at
https://github.com/mattiaivaldi/TRACKS.git

Outline

1. Introduction
2. Setup
   a. Layer
   b. Hit
   c. Particle
   d. Tools
3. Generation
   a. Algorithm
   b. Results and performances
4. Reconstruction
   a. Algorithm
   b. Performances macros
   c. Results and performances



Introduction

TRACKS is a Monte Carlo particles' tracks generation and reconstruction with multiple scattering and noise.

The package contains:

`Layer.h/cxx`
`Hit.h/cxx`
`Particle.h/cxx`
`Tools.h/cxx`

classes and libraries

`tracks_gen.C`
`tracks_reco.C`
`tracks.C`

generation, reconstruction and head macro

`cluster_study.C`

peakfinder study

`spit_performance.C`

performance study

The package returns:

`gen.root`
- `TG` *(TTree)*
  - `HITL1`
  - `HITL2`
  - …
- `z_gen` *(NTuple)*

`reco.root`
`perform.root`

data

`c_gen.eps`
`c_hit.eps`
…

plots in the folder tracksplot

Introduction - how to run TRACKS

TRACKS has an easy 3-steps usage:

1 - open and edit the file detector_info.txt with the format

```
layer_name width radius thickness theta_rms
```

2 - open and edit the macro `tracks.C` with your preferences

```
gROOT->ProcessLine("tracks_gen(a,b,c,d,e,f,g,h)")
```

bool a = verbose mode ON/OFF
bool b = print, save and write on file plots ON/OFF
bool c = multiple scattering ON/OFF
bool d = noise ON/OFF
int e = 5 custom z, 10 custom multiplicity, 15 both custom
int f = # of collisions performed
double g = custom vertex z
double h = custom event multiplicity

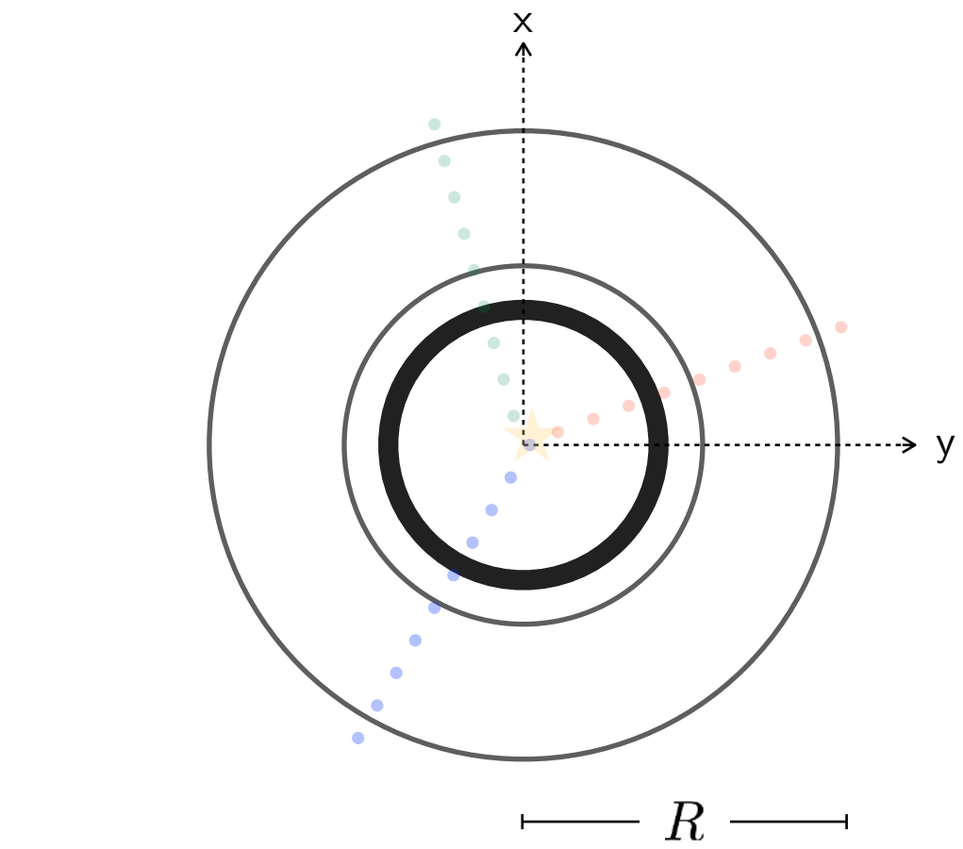
```
gROOT->ProcessLine("tracks_reco(a,b,c,d,e,f)")
```

bool a = verbose mode ON/OFF
bool b = print, save and write on file plots ON/OFF
double c = smearing parameter on z
double d = smearing parameter on phi
double e = ambiguity check amplitude
int f = ambiguity check width

3 - open a ROOT session and interpret `root [0] .x tracks.C`

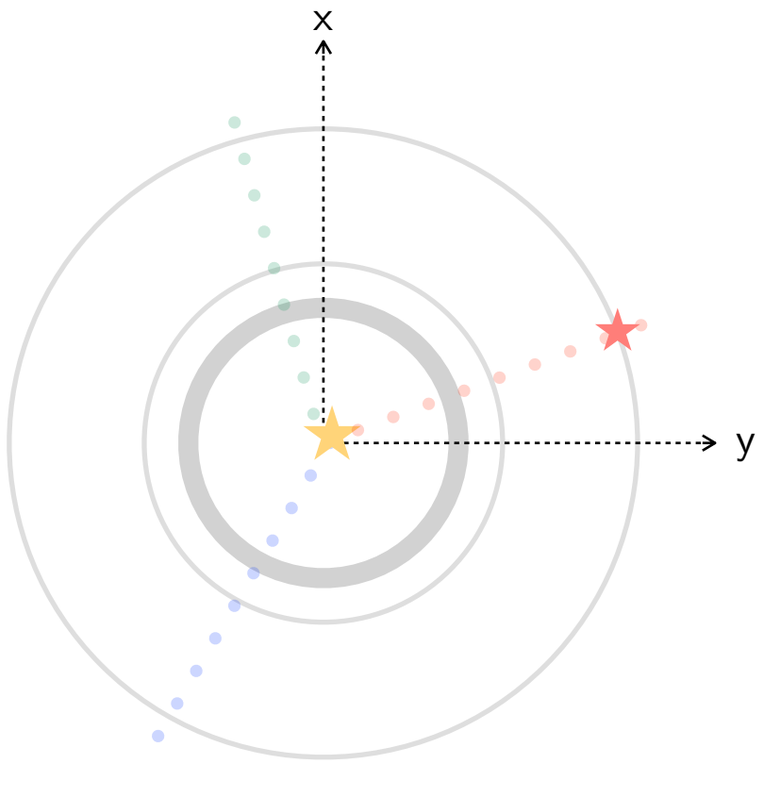
CAVEAT lines 30 and 31 of tracks.C are commented by default, uncomment them to execute the performances study.

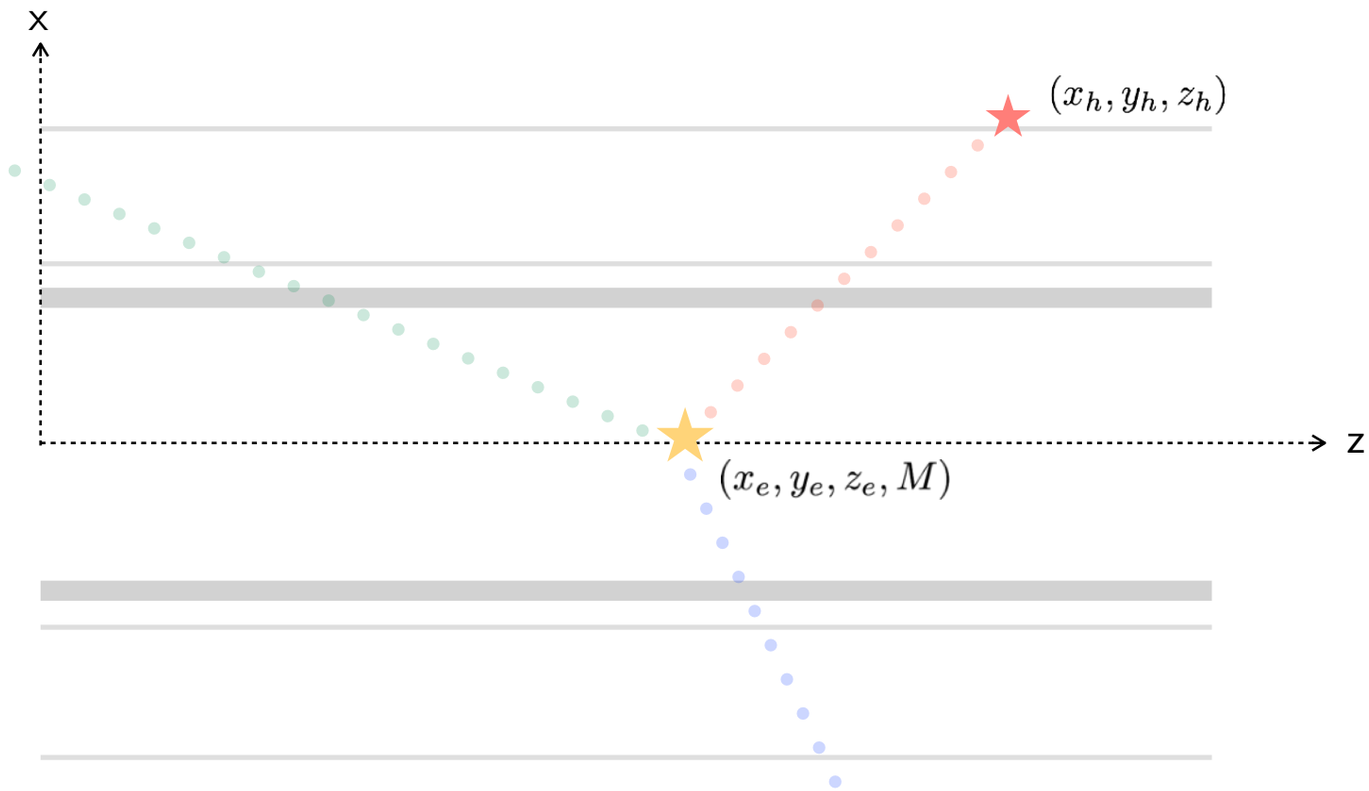
Setup - Layer




```
Layer::Layer(TString N, double W, double R, double T, double RMS): TObject(),
fName(N),
fWidth(W),//cm
fRadius(R),//cm
fThick(T),//mm
fRMS(RMS)//rad
{
  //standard constructor, N W R T RMS are given by the user in the data sheet
}
```

Setup - Hit

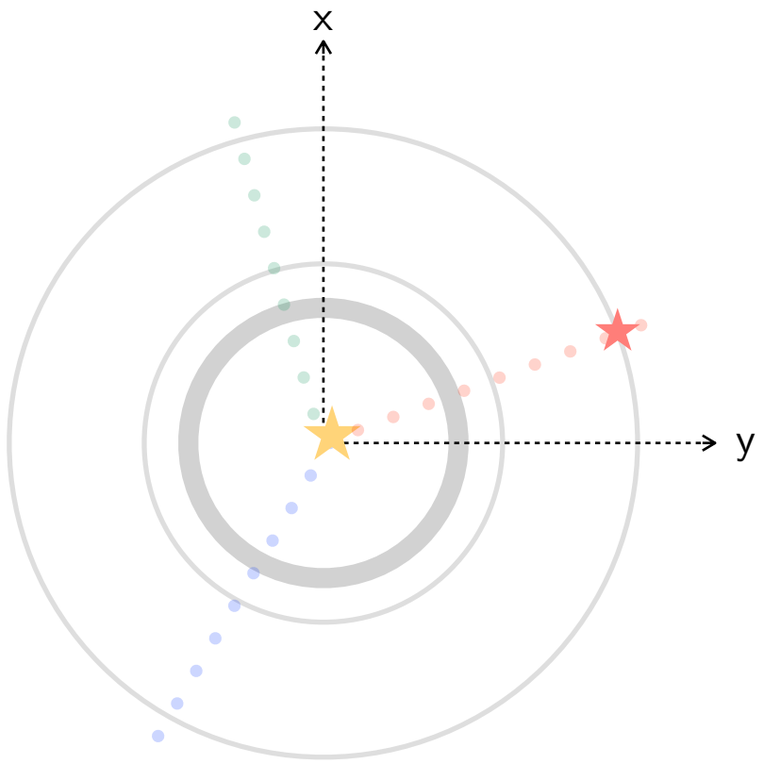


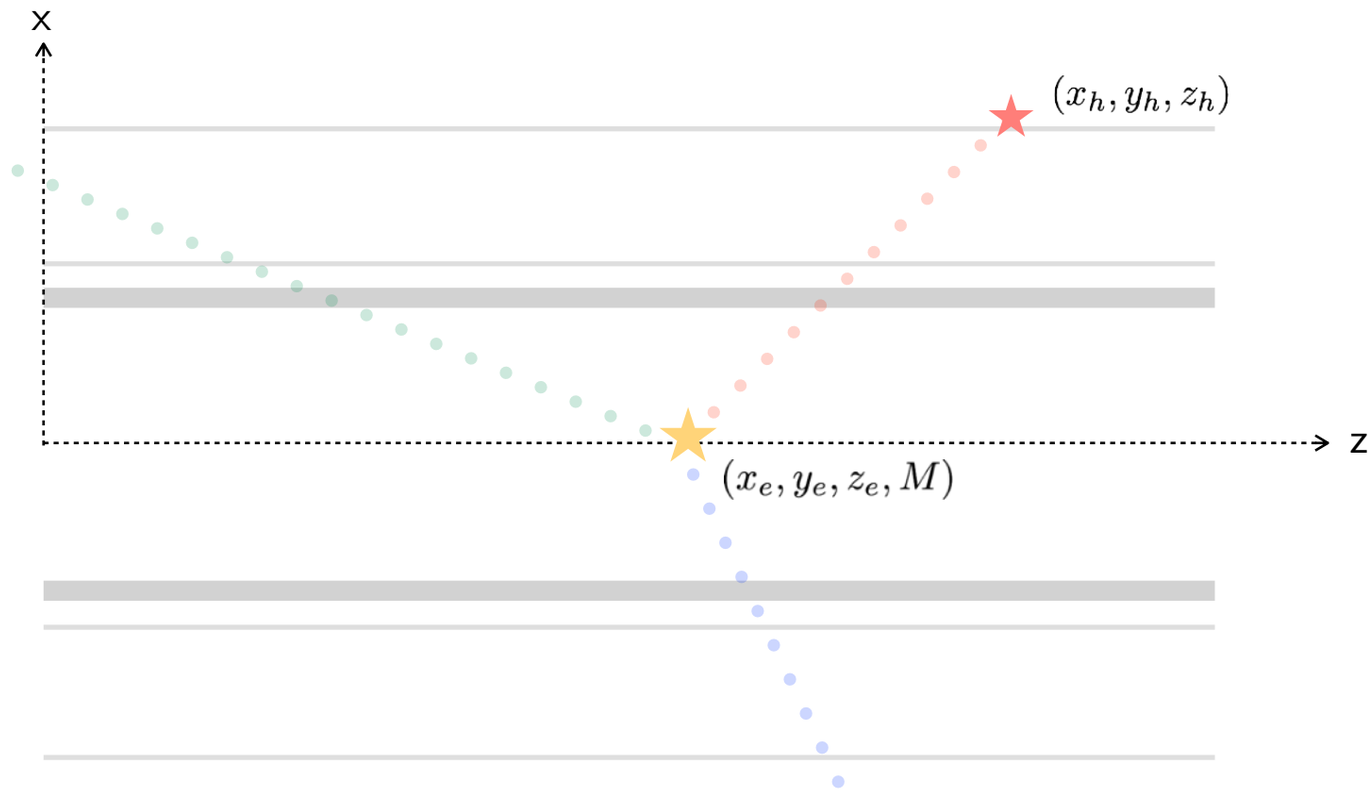



```
Hit::Hit(double x, double y, double z): TObject(),
fX(x),
fY(y),
fZ(z)
{
  //hit constructor, def1
}


Hit::Hit(double R, double H): TObject(),
fX(gRandom->Uniform(-R, R)),
fY(0.),
fZ(gRandom->Uniform(-H/2,H/2))
{
  if(gRandom->Rndm()<0.5){
    fY=TMath::Sqrt(R*R-fX*fX);
  }else{fY=-1*TMath::Sqrt(R*R-fX*fX);}
  //spurious hit constructor, def1
}
```

```
Hit::Hit(double meanv, double sigmaxy, double sigmaz,
TH1F *distr_mult): TObject(),
fX(gRandom->Gaus(meanv,sigmaxy)),
fY(gRandom->Gaus(meanv,sigmaxy)),
fZ(gRandom->Gaus(meanv,sigmaz)),
fMult((int)distr_mult->GetRandom())
{
  //uncomment to impose a z within a specific range
  /*double zgen;
  do{zgen=gRandom->Gaus(meanv,sigmaz);}
  while(Abs(zgen)>5.3);//eg here 1 sigma
  fZ=zgen;*/

  //event constructor, def2
}
```

Setup - Hit

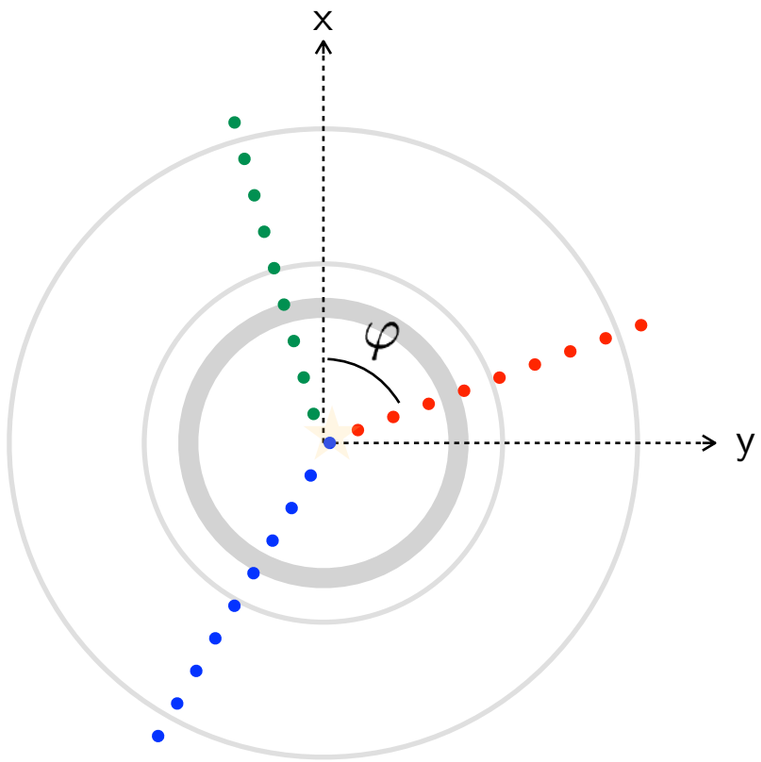


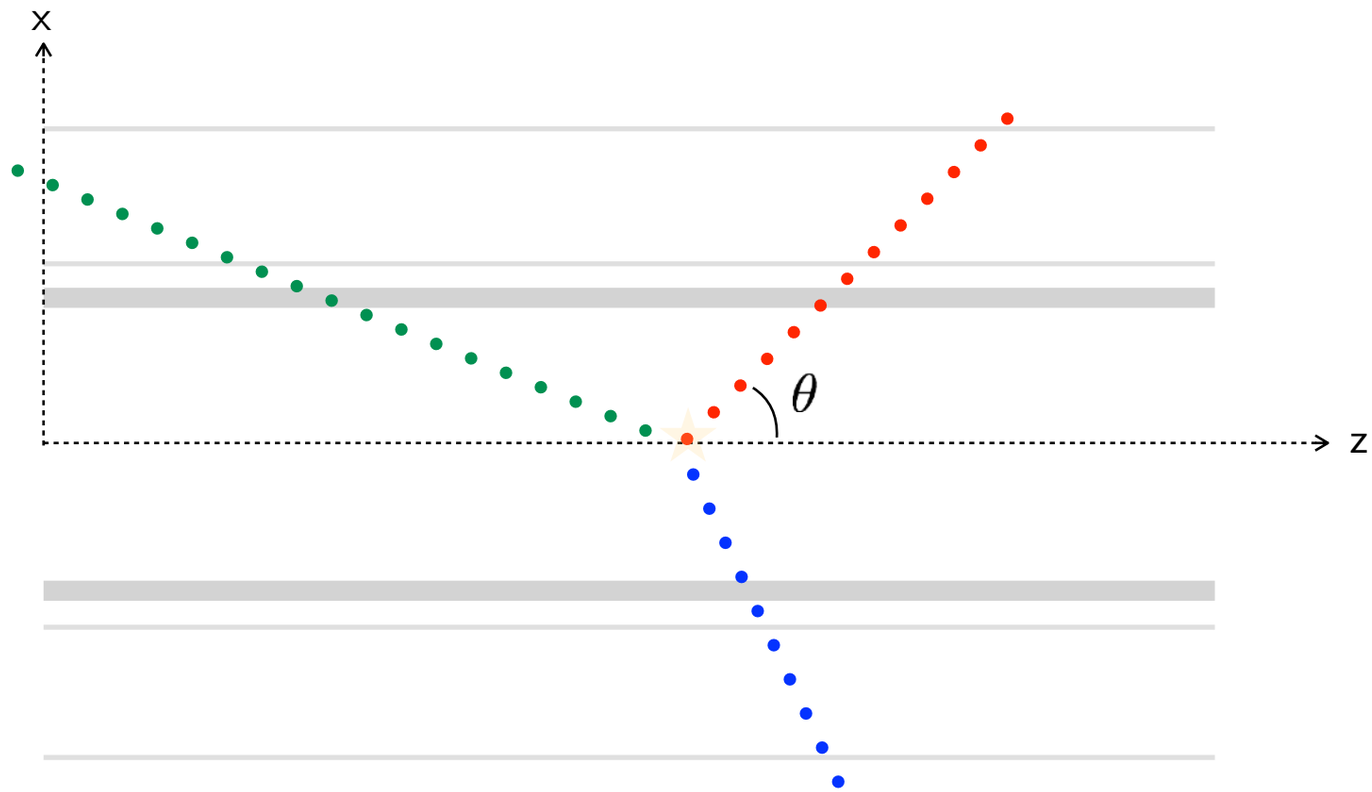



```
Hit::Hit(double meanv, double sigmaxy, double z_custom,
int mult_custom): TObject(),
fX(gRandom->Gaus(meanv,sigmaxy)),
fY(gRandom->Gaus(meanv,sigmaxy)),
fZ(z_custom),
fMult(mult_custom)
{
  //custom event, def2
}
```

```
void Hit::Customize(int custom, double z_custom,
int mult_custom) {
  if(custom==5){
    fZ=z_custom;
  }else if(custom==10){
    fMult=mult_custom;
  }else if(custom==15){
    fZ=z_custom;
    fMult=mult_custom;
  }
}
```

Setup - Particle

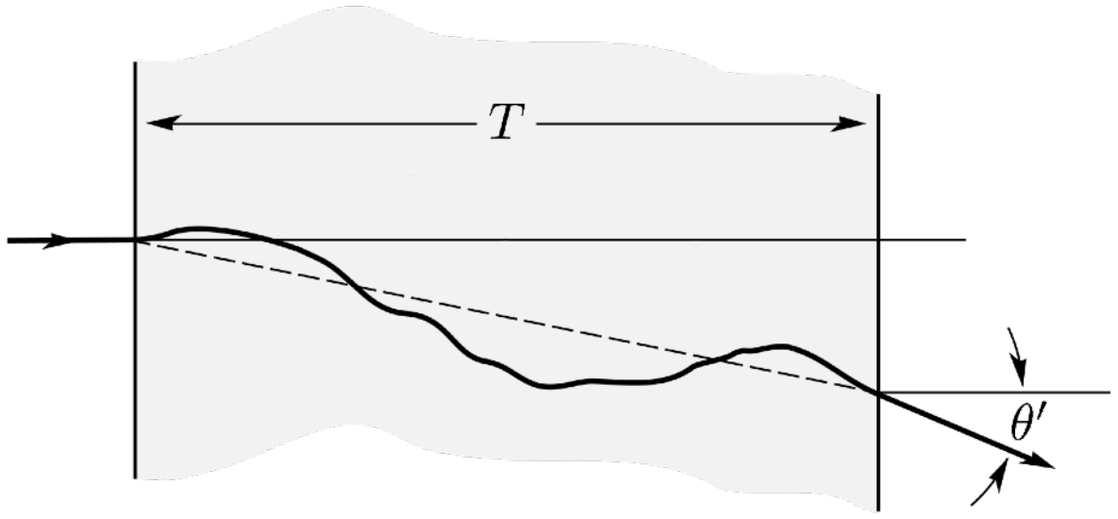





```
Particle::Particle(TH1F *distr_rap): TObject(),
fRap(distr_rap->GetRandom()),
fTheta(2*ATan(Exp(-(double)fRap))),
fPhi(gRandom->Uniform(2*Pi()))
{
  //standard constructor
}
```

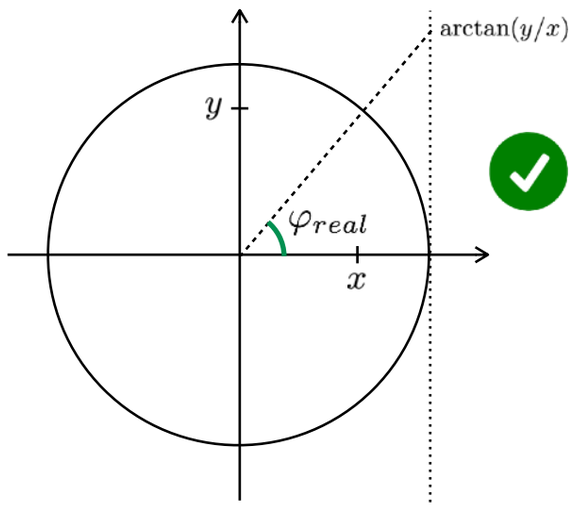
Setup - Particle/multiple scattering

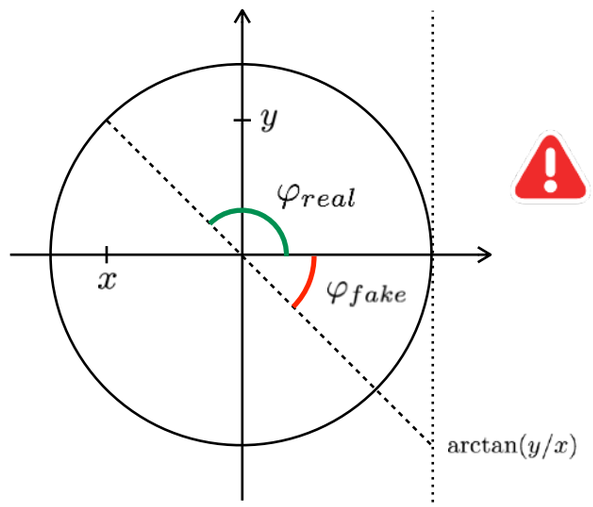
```
void Particle::Rotate(double rms){

  //rotation matrix for multiple scattering

  gRandom->SetSeed(0);

  double theta0=rms/Sqrt(2);
  double thetap=gRandom->Gaus(0,theta0);//scattering angle theta
  double phip=gRandom->Uniform(2*Pi());//scattering angle phi

  double mr[3][3], pol[3], rot[3], r;//rotation matrixes

  mr[0][0]=-Sin(fPhi);
  mr[1][0]=Cos(fPhi);
  mr[2][0]=0;
  mr[0][1]=-Cos(fPhi)*Cos(fTheta);
  mr[1][1]=-Cos(fTheta)*Sin(fPhi);
  mr[2][1]=Sin(fTheta);
  mr[0][2]=Sin(fTheta)*Cos(fPhi);
  mr[1][2]=Sin(fTheta)*Sin(fPhi);
  mr[2][2]=Cos(fTheta);

  pol[0]=Sin(thetap)*Cos(phip);
  pol[1]=Sin(thetap)*Sin(phip);
  pol[2]=Cos(thetap);

  for(int i=0; i<3; i++){
    rot[i]=0;
    for(int j=0; j<3; j++){
      rot[i]+=mr[i][j]*pol[j];
    }
  }

  fTheta=ACos(rot[2]);

  if(rot[0]>0&&rot[1]>0){
    fPhi = ATan(rot[1]/rot[0]);
  }else if(rot[0]>0&&rot[1]<0){
    fPhi = ATan(rot[1]/rot[0])+2*Pi();
  }else if(rot[0]<0&&rot[1]>0){
    fPhi = ATan(rot[1]/rot[0])+Pi();
  }else if(rot[0]<0&&rot[1]<0){
    fPhi = ATan(rot[1]/rot[0])+Pi();
  }

}
```



$$\theta_0 = \frac{13.6\,\mathrm{MeV}}{\beta\, c\, p}\, z \sqrt{\frac{x}{X_0}} \left[1 + 0.038 \ln\left(\frac{x}{X_0}\right)\right] = \frac{1}{\sqrt{2}}\,\theta_{RMS}$$

$$f(\theta') = \frac{1}{\theta_0\sqrt{2\pi}}\, e^{-\frac{\theta'^2}{2\,\theta_0^2}}$$





$$\varphi_{real} = \varphi_{fake} + \pi$$

Setup - Tools

Tools is a library of functions used during both generation and reconstruction.

```
void verbosities(bool b_verbose, bool b_multiscatter, bool b_noise,
int kExp){

//if the verbose mode is ON, it informs the user whether the
multiple scattering and the noise are ON, and whether it will
display the vertex info

}
```

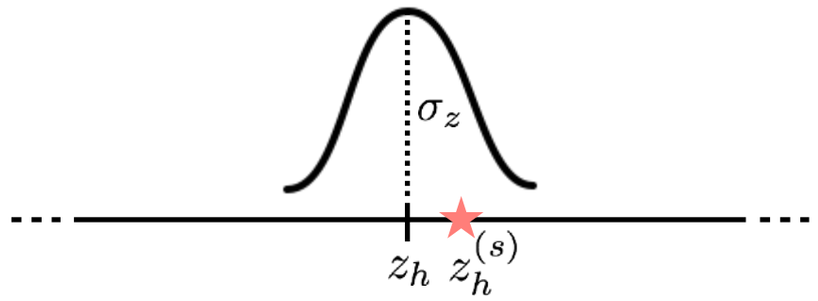
```
void graph/histo/stack/pavestyler(TPaveText &pave, double textsize){

//make up on the pave displaying stats and plot info

}
```

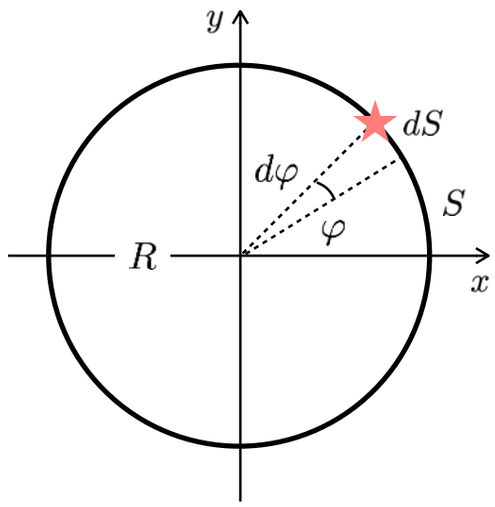
```
double *hit_point(double x0, double y0, double z0, double theta,
double phi, double R){

//given an initial point (x0,y0,z0) and a direction (theta,phi), it
returns a pointer to the first element of the array (x,y,z) of the
intersection coordinates with a layer of radius R

}
```

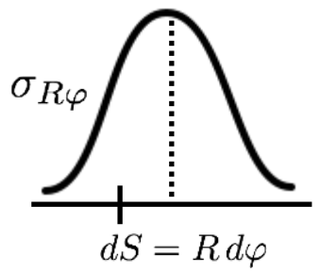
```
bool detect(Hit* vtx, Layer* L, Particle &part, TClonesArray &cross,
bool b_verbose, bool b_multiscatter, int &counter, TH1D** histo){

//given a vertex, if the particle hits the layer a boolean is set to
true, a TClonesArray and an histogram are filled with the hit
coordinates, and a counter is ++; if the multiple scattering is ON
the particle is updated; the boolean is returned

}
```

```
void noise(bool b_verbose, int Noise, int Mult, TClonesArray &cross,
Layer* L){

//fills the TClonesArray Noise-times with the spurious hit
constructor

}
```

```
void smear(int index, double sigmaz, double sigmarf, double R,
TClonesArray &cross){

//applies a gaussian smearing with parameters sigmaz and sigmarf on
the hit coordinates stored in the TClonesArray

}
```



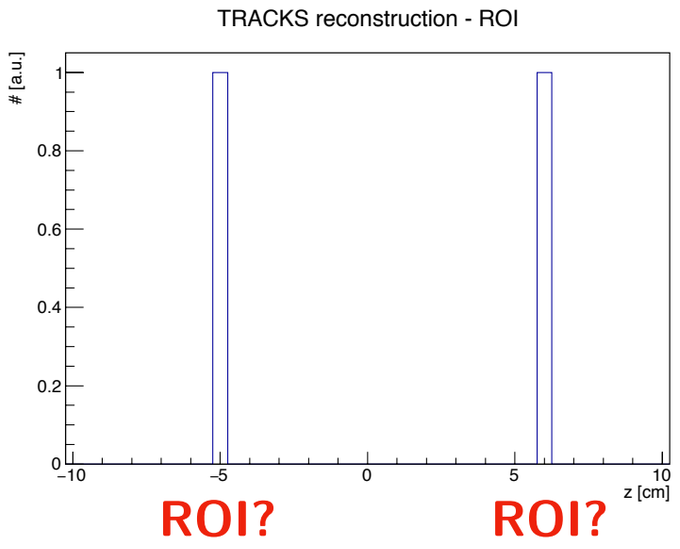


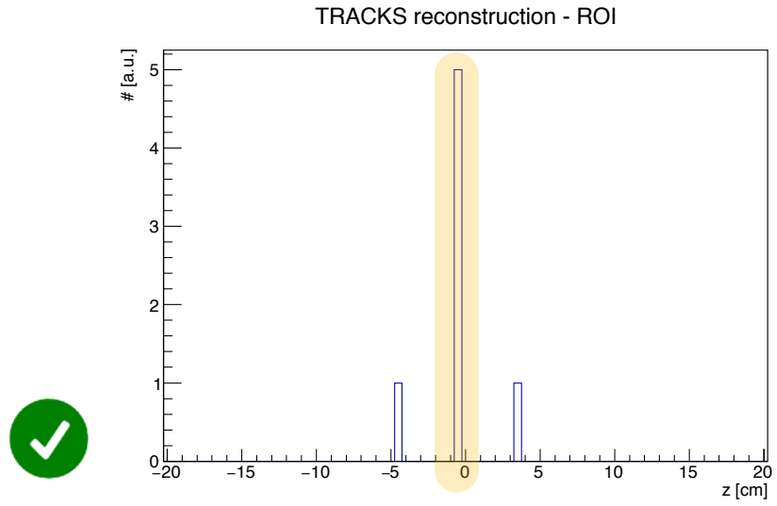


Setup - Tools/peakfinder

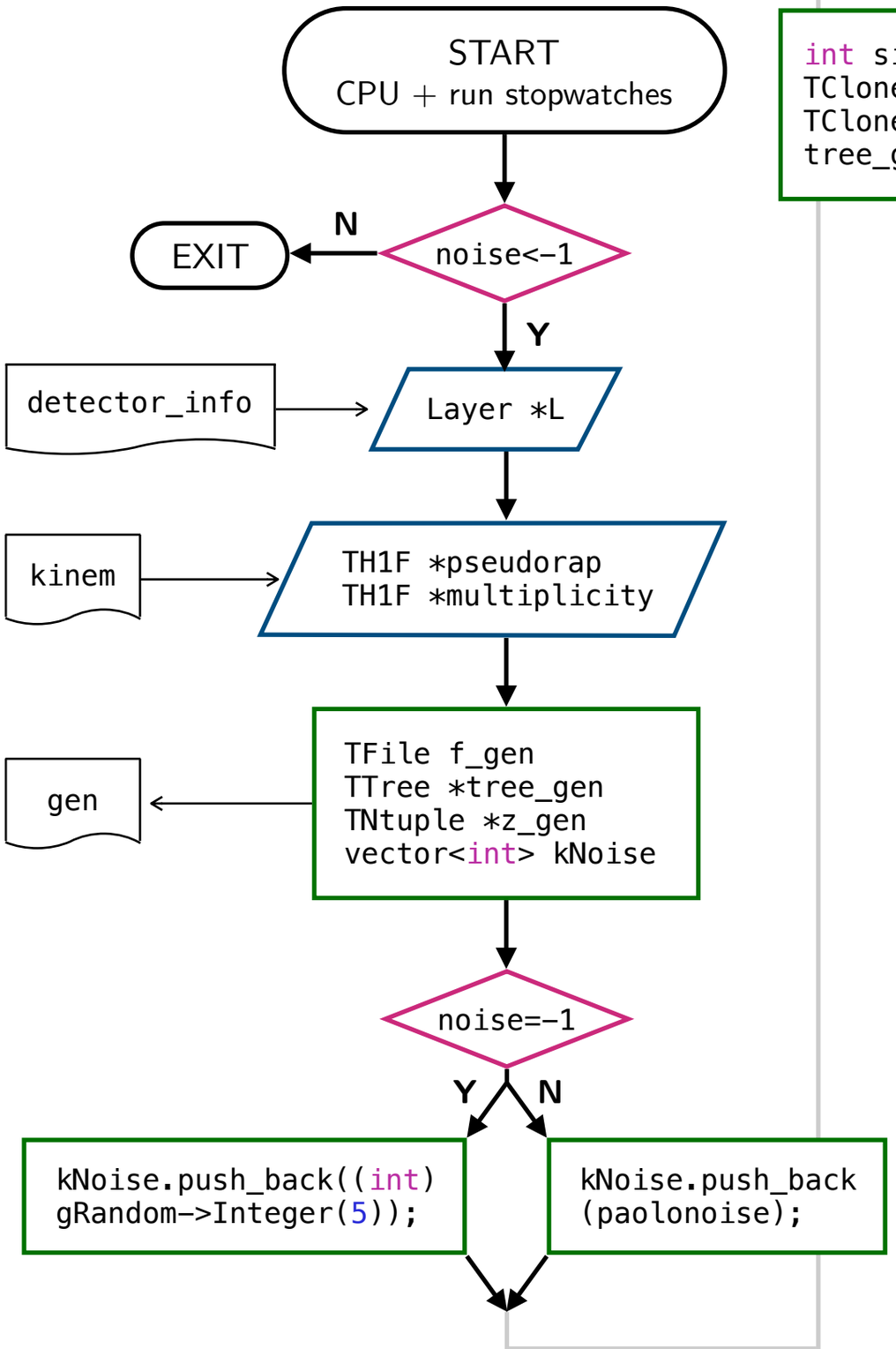
```
bool peakfinder(TH1D* histo, double ampli, int
width){//tracklet ambiguity check

  int fisrtbin=(int)(ampli/2+0.5);
  int lastbin=(int)(ampli/2+0.5)-1;//define the
check range
  bool peakit=false;
  int kBin=histo->GetSize()-2;//number of bin
excluding under- and overflow
  int binC=histo->GetMaximumBin();//bin index of
the first global maximum
  double Max=histo->GetBinContent(binC);//value of
the first global maximum
  double ClusterSize=0;

  for(int i=fisrtbin;i<kBin-lastbin;i++){//loop
over the check range
    for(int j=-lastbin;j<=lastbin;j++){//loop over
width
      ClusterSize+=histo->GetBinContent(i-j);
    }//end loop over width
    if(ClusterSize>=ampli*Max&&((i<=binC-width)||
      (i>=binC+width))){//this is the case in
      which we have ambiguity
      peakit=false;
      ClusterSize=0;
      break;
    }else{//this is the case in which we don't
          have ambiguity - the event is good
          ClusterSize=0;
          peakit=true;
    }
  }//end loop over the check range

  return peakit;
}
```

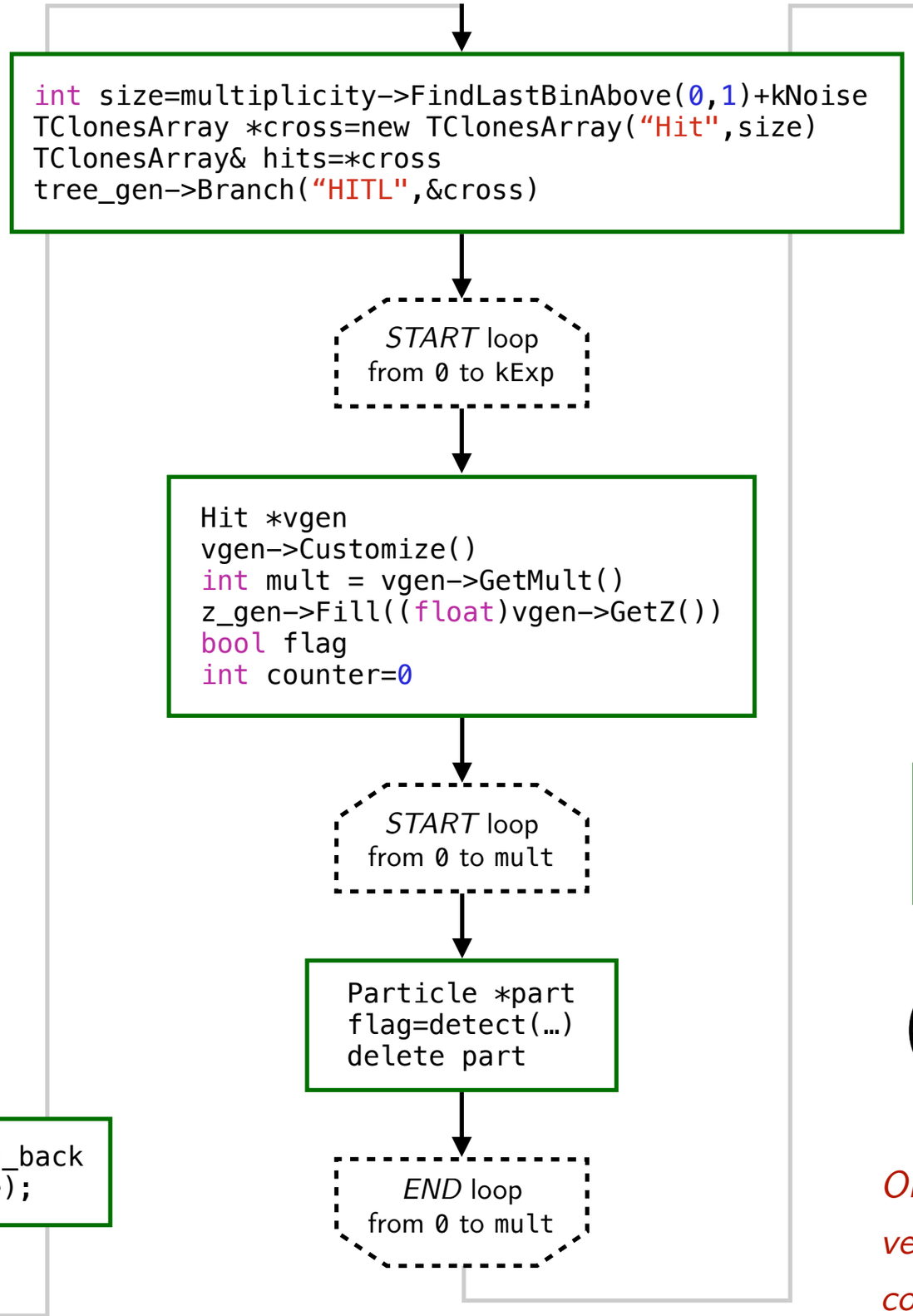
The idea of this check if to verify whether, *e.g.* with ampli=1 and width=5, there is more than one region of the histogram where the sum of the contents of 5 bins is at least equal to 1*Max, where Max is the value of the first global maximum, excluding the maximum itself. In this case we are not able to choose where is most probable the vertex is and discard the event.

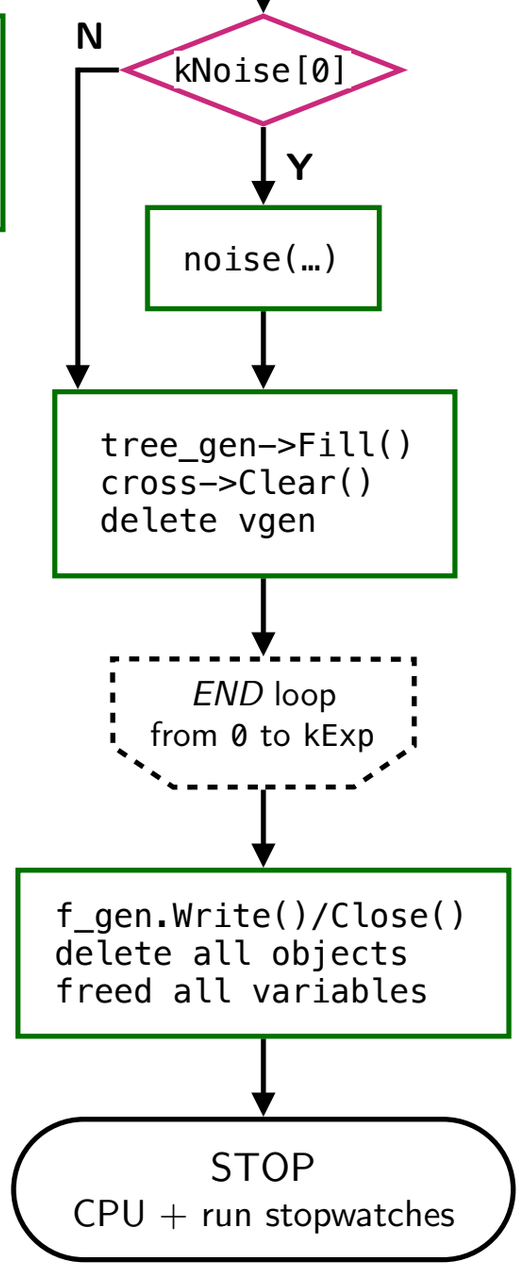




Generation - Algorithm



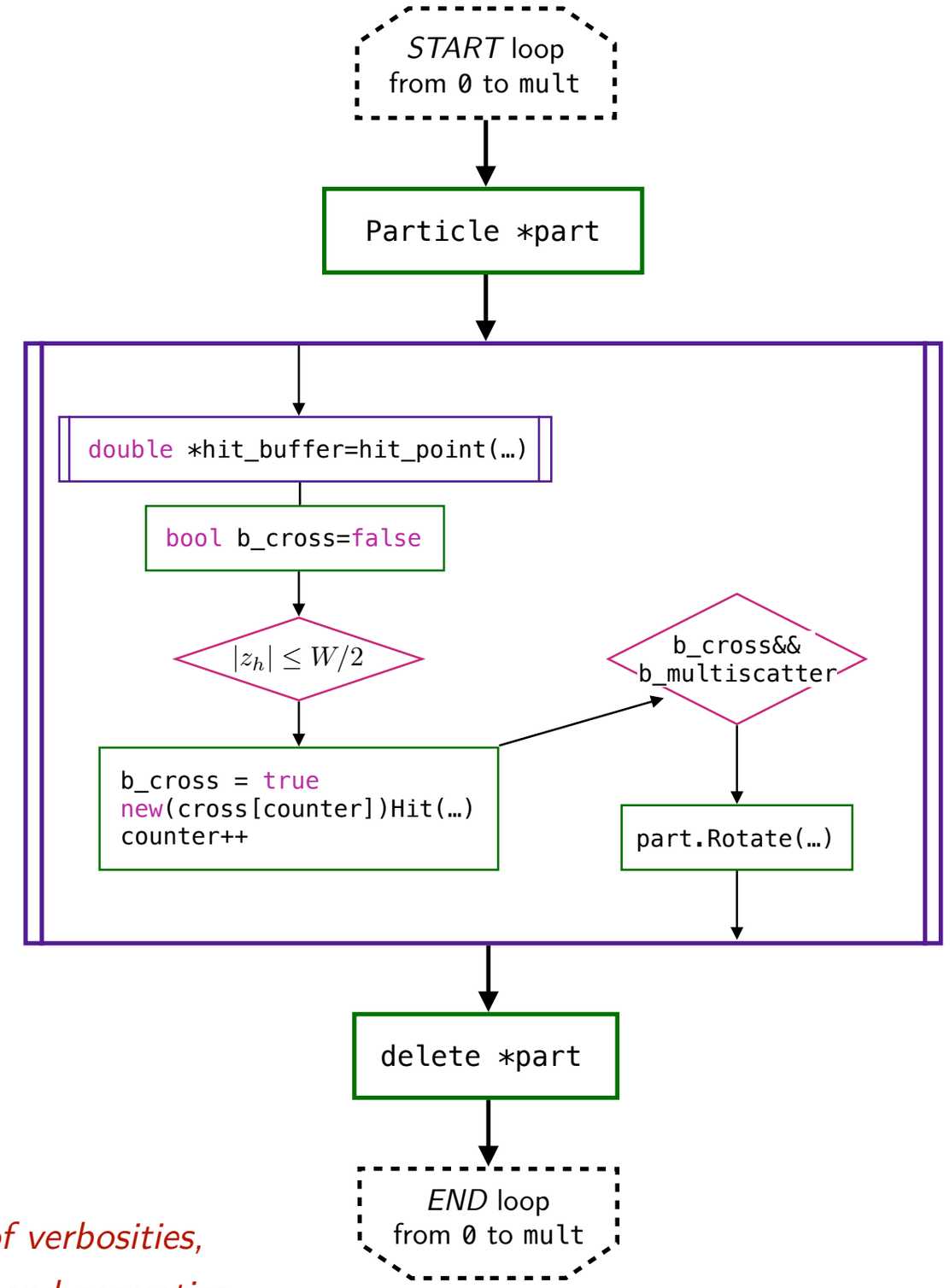




Only for 1 layer, net of verbosities, plots and cosmetics.

Generation - Algorithm/detect

```
bool detect(Hit* vtx, Layer* L, Particle &part, TClonesArray &cross,
bool b_verbose, bool b_multiscatter, int &counter, TH1D** histo){

    double *hit_buffer;
    bool b_cross=false;

    hit_buffer=hit_point(vtx->GetX(),vtx->GetY(),
    vtx->GetZ(),part.GetTheta(),part.GetPhi(),L->GetRadius());
    //pointer to the first element of an array with hit coordinates

    if(Abs(*(hit_buffer+2))<=(L->GetWidth()/2.)) {

        b_cross = true;//yes we have detection
        new(cross[counter])Hit(*(hit_buffer+0),*(hit_buffer+1),
        *(hit_buffer+2));//fill TCA with hit coordinates

        for(int i=0;i<=2;i++){histo[i]->Fill(*(hit_buffer+i));}
        //fill TH1D with hit coordinates

        if (b_cross&&b_multiscatter) {
            part.Rotate(L->GetRMS());
            //if multiscattering ON updates part with new angles
        }

        if (b_verbose) {
            cout<<"Hit with "<<L->GetLayerName();
            printf(" at (%f, %f, %f)\nAngles after: theta %f -
            phi %f\n\n",((Hit*)cross[counter])->GetX(),
            ((Hit*)cross[counter])->GetY(),
            ((Hit*)cross[counter])->GetZ(),
            part.GetTheta(),part.GetPhi());
        }

        counter++;//is passed to detect next time, only if there
        is detection the TCA is really filled

    }else{
        if(b_verbose){
            cout<<"Does not hit "<<L->GetLayerName()<<endl<<endl;
        }
    }

    return b_cross;

}
```



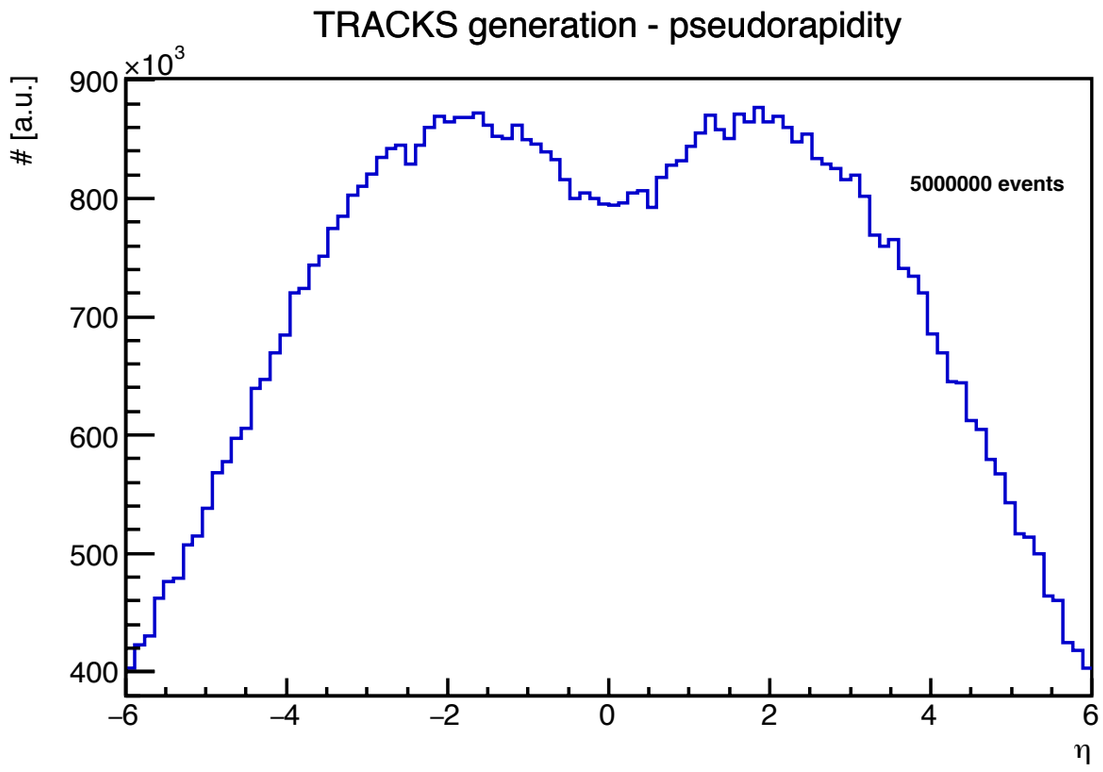
Net of verbosities, plots and cosmetics.

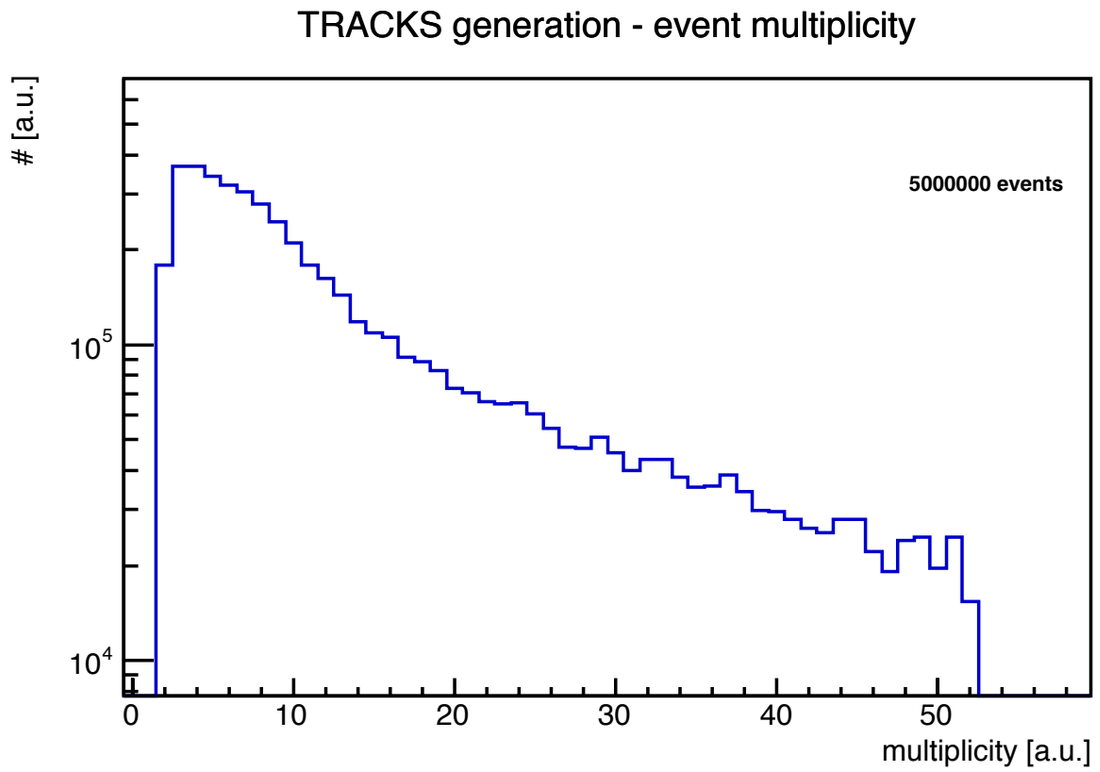
Generation - results and performances

- pp collisions, $p = 0.7$ GeV c^{-1}
- 3 layers: 1 x Be Beam Pipe, 2 x Si tracker

fName	fWidth	fRadius	fThick	fRMS
BP	27	3	0.8	0.0001
L1	27	4	0.2	0.0001
L2	27	7	0.2	0.0001

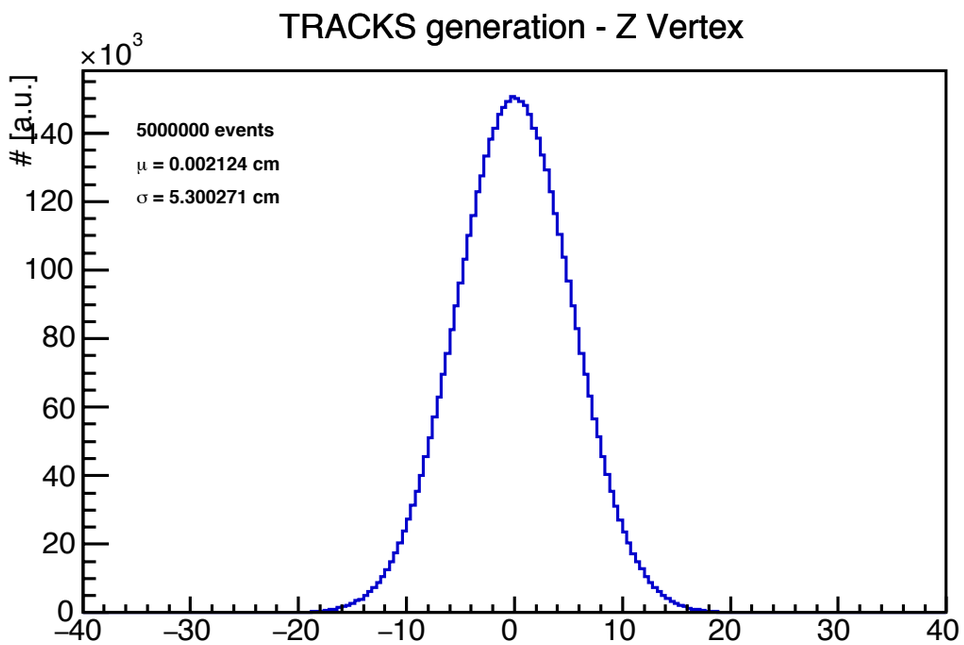
- 5×10^6 events

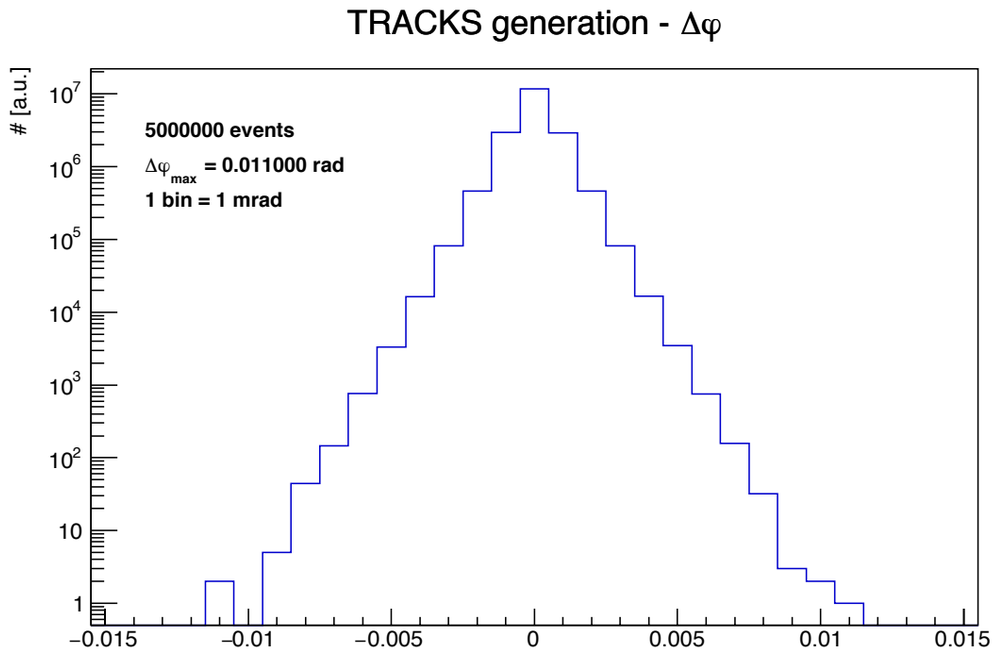


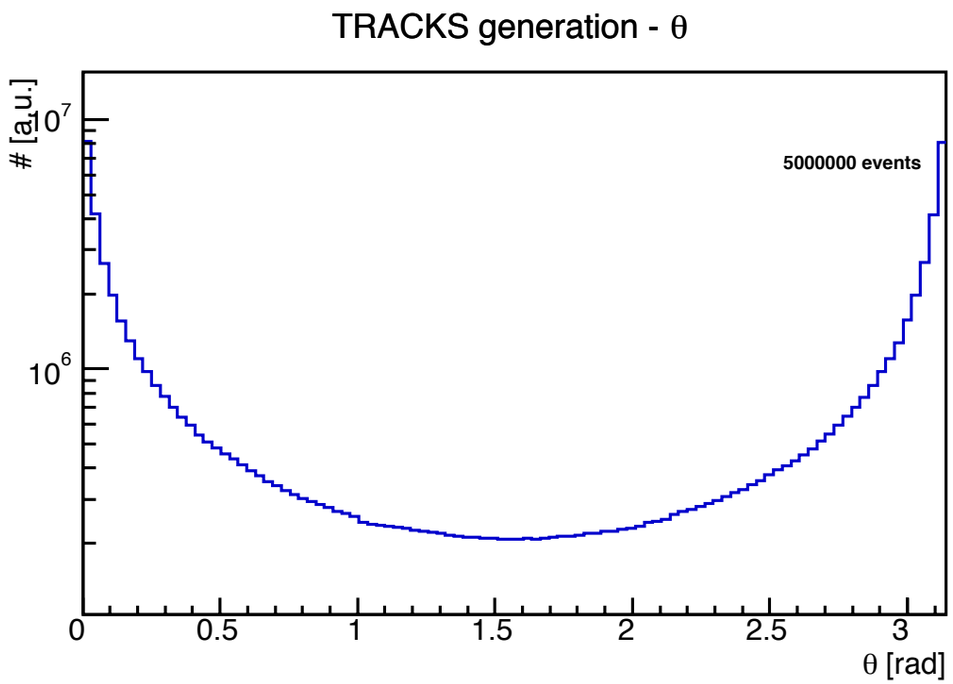


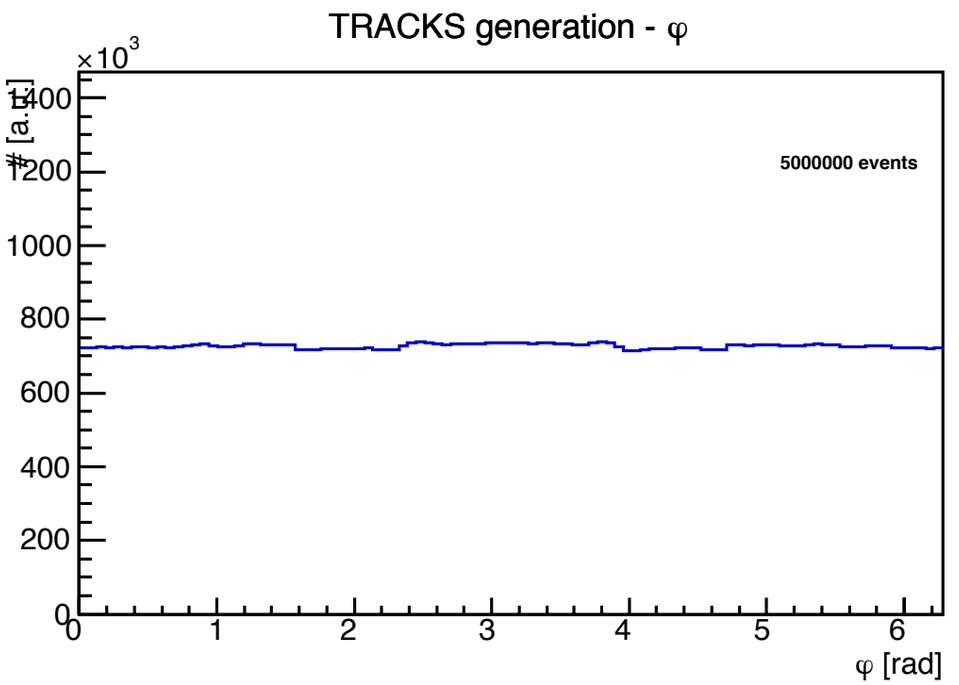
Generation - results and performances

Vertex Z and angles



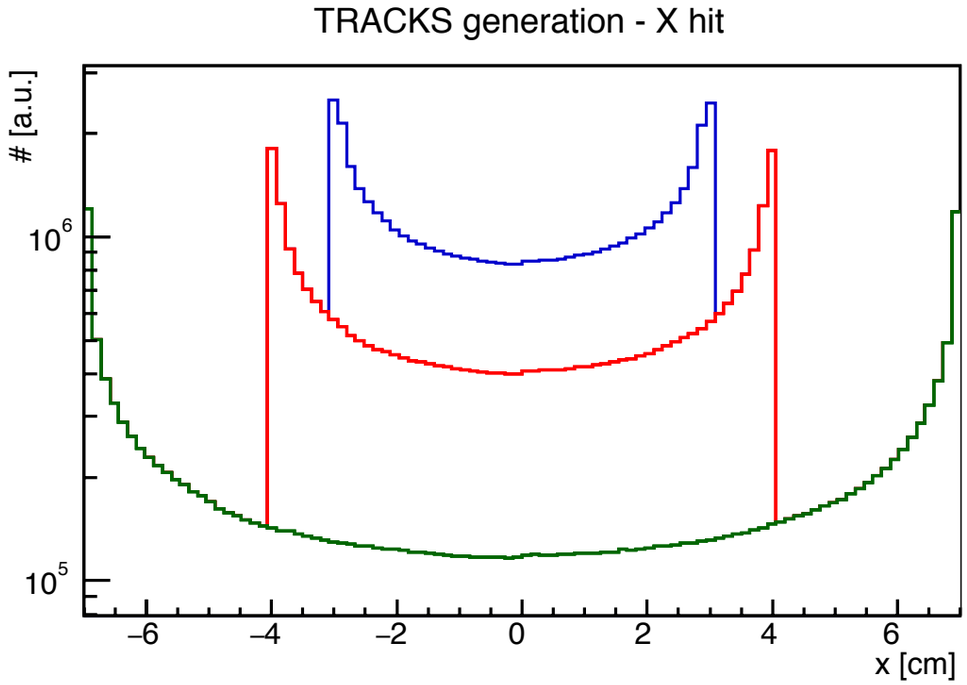


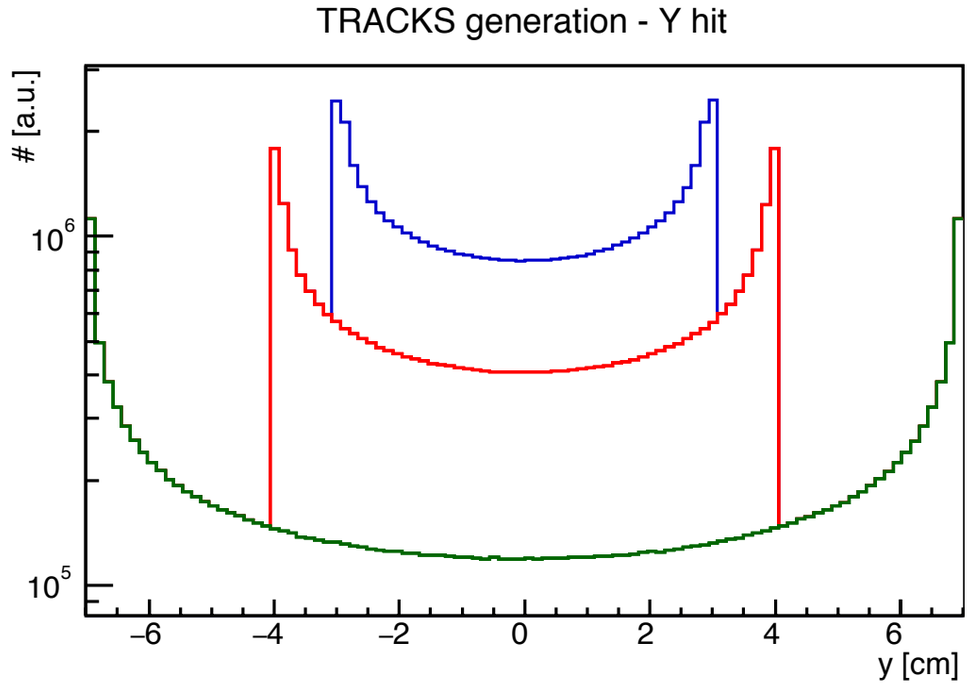


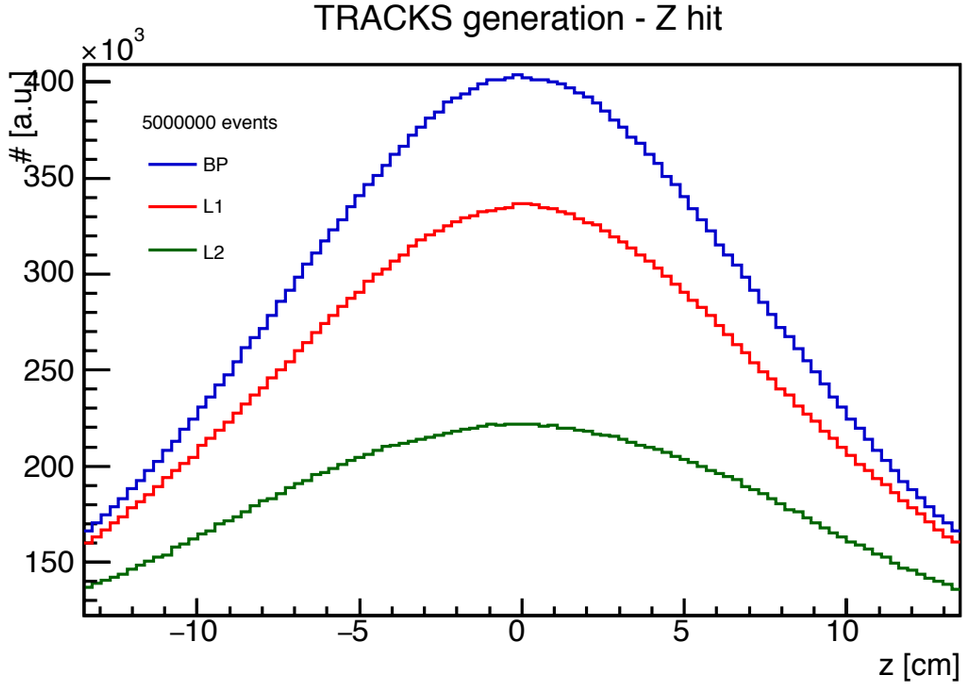


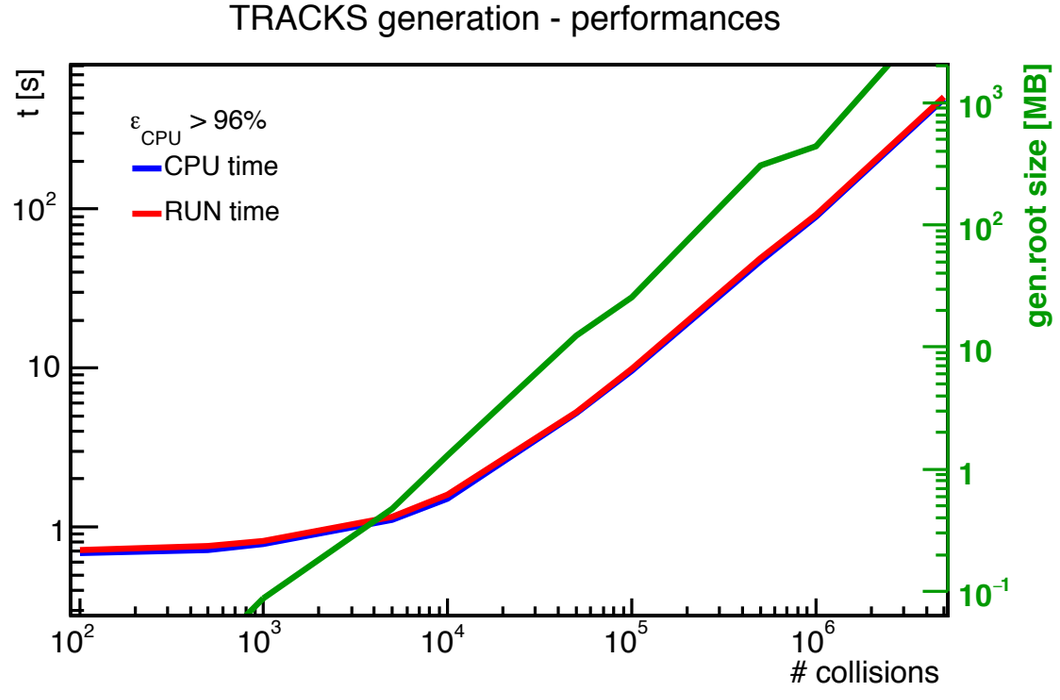
Generation - results and performances

Hits' coordinates and CPU performances





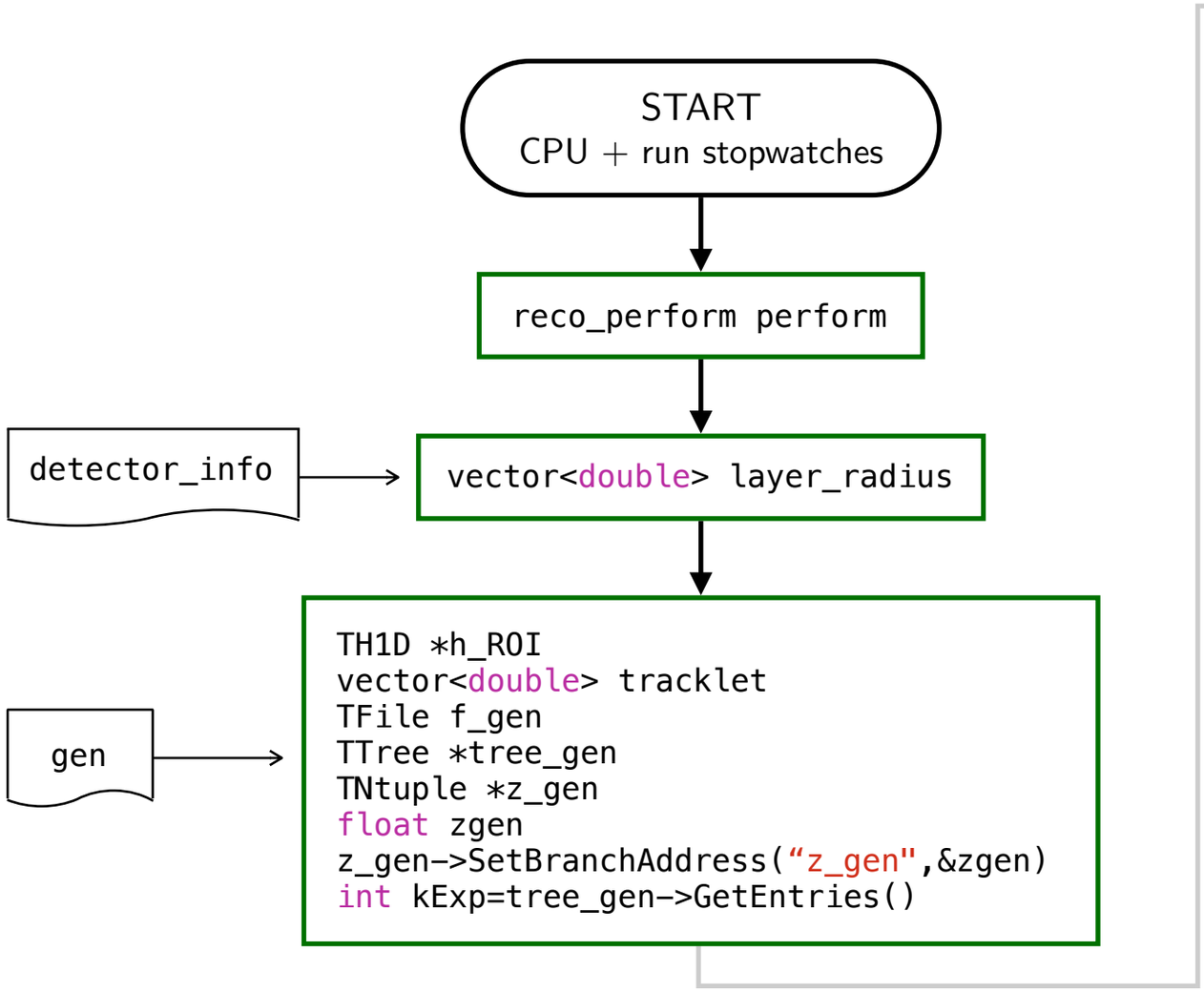




Reconstruction - Algorithm

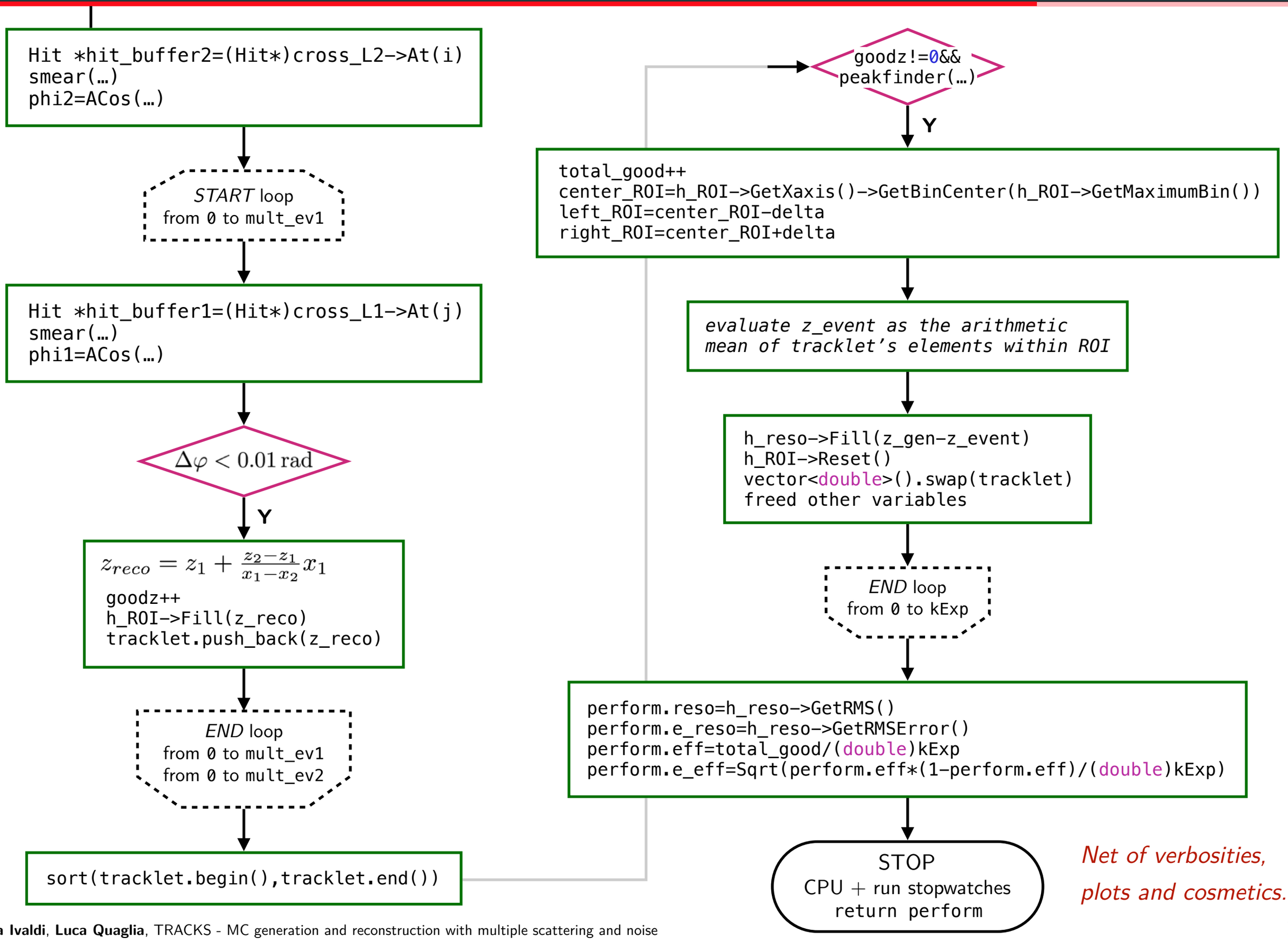
```
struct reco_perform{
  double reso;
  double e_reso;
  double eff;
  double e_eff;
};

reco_perform tracks_reco(…){…}
```

The function tracks_reco returns an object of type reco_perform, containing resolution and efficiency with uncertainties of the algorithm.



Net of verbosities, plots and cosmetics.

Reconstruction - Algorithm



Net of verbosities, plots and cosmetics.

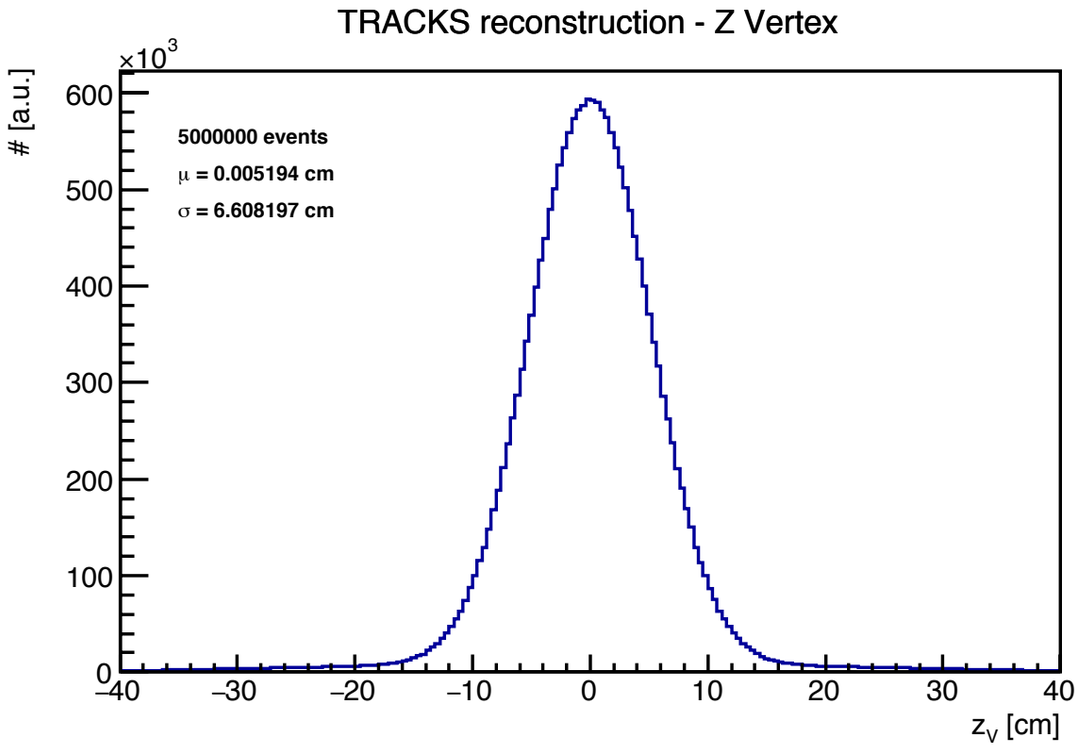
Performances macros

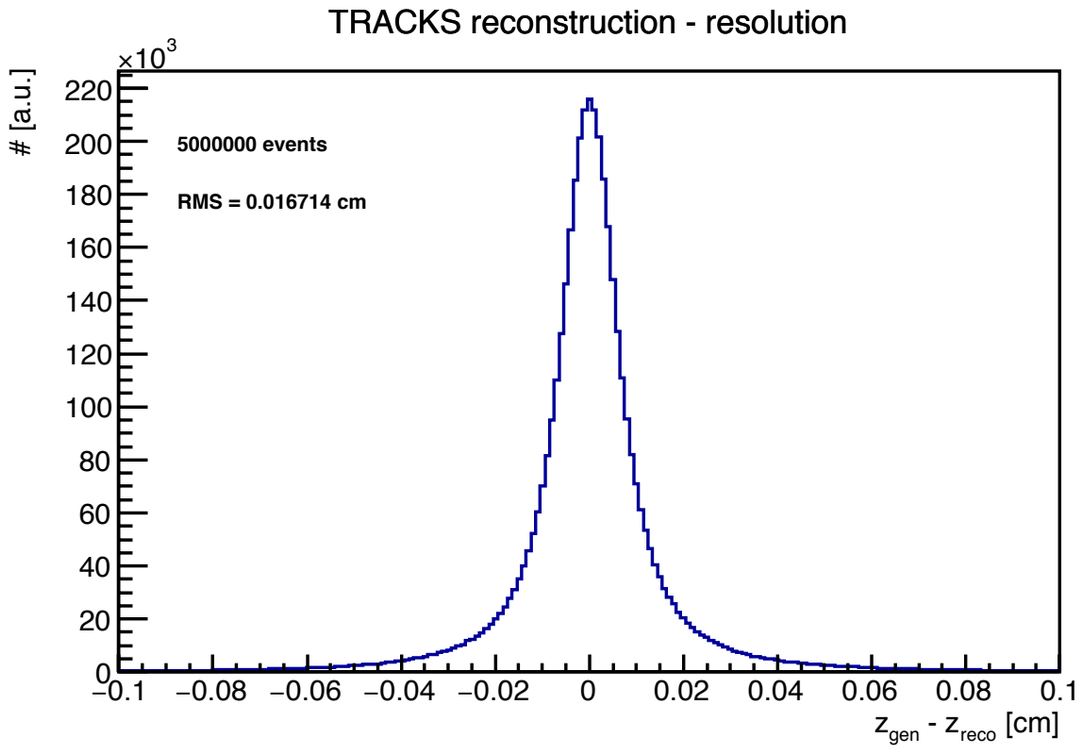
`spit_perform.C` to study the resolution and the efficiency of the algorithm with different values of vertex Z and multiplicity; the macro repeatedly performs tracks_gen and tracks_reco imposing different values of z_custom and mult_custom.

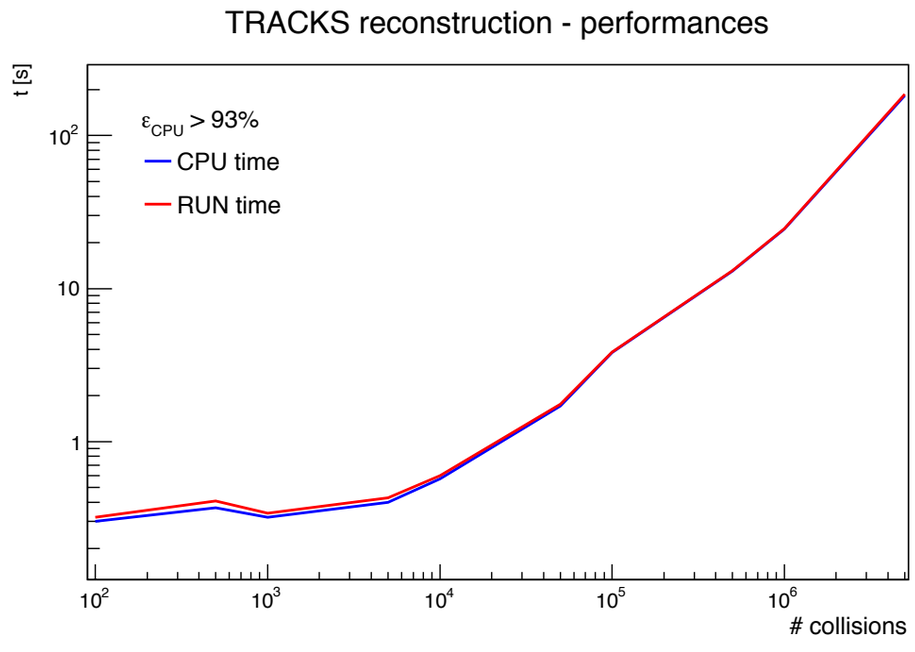
`cluster_study.C` to study the behaviour of the reconstruction algorithm with different values of the peakfinder arguments; the macro has the same functioning of spit_perform.C, but also the amplitude and width parameters are variable.

Reconstruction - results and performances

Reconstructed Z, resolution and CPU performances - h_ROI has 1 bin = 5 mm

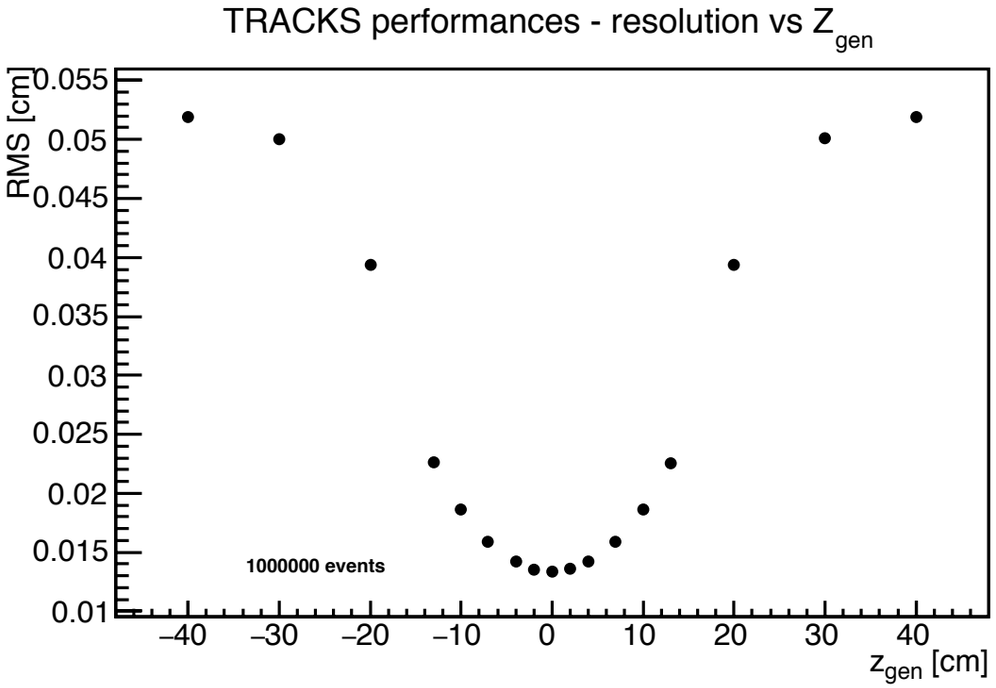


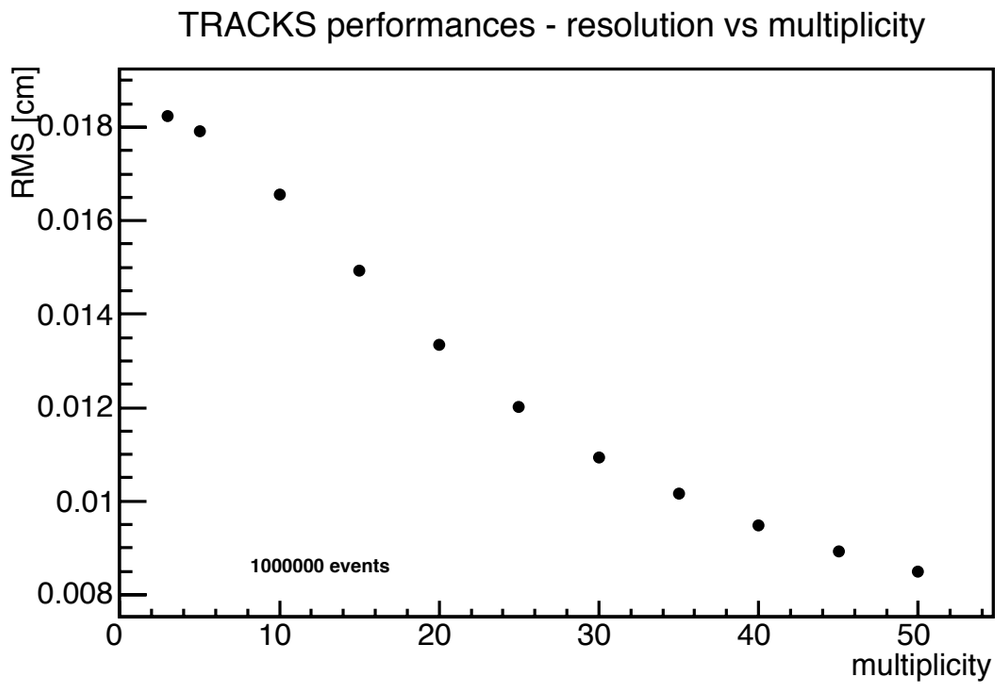


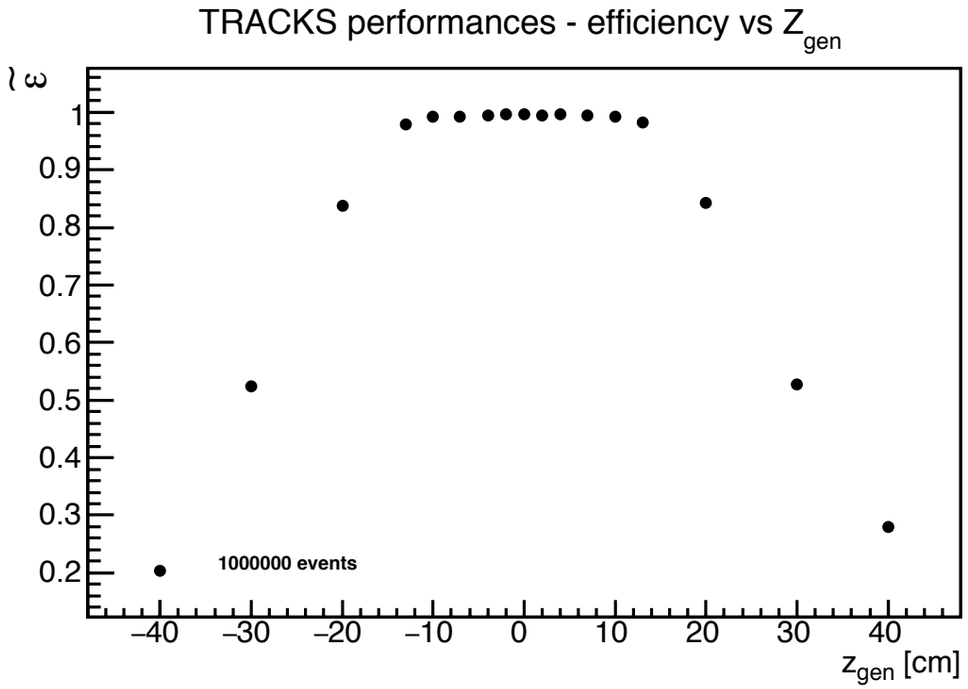


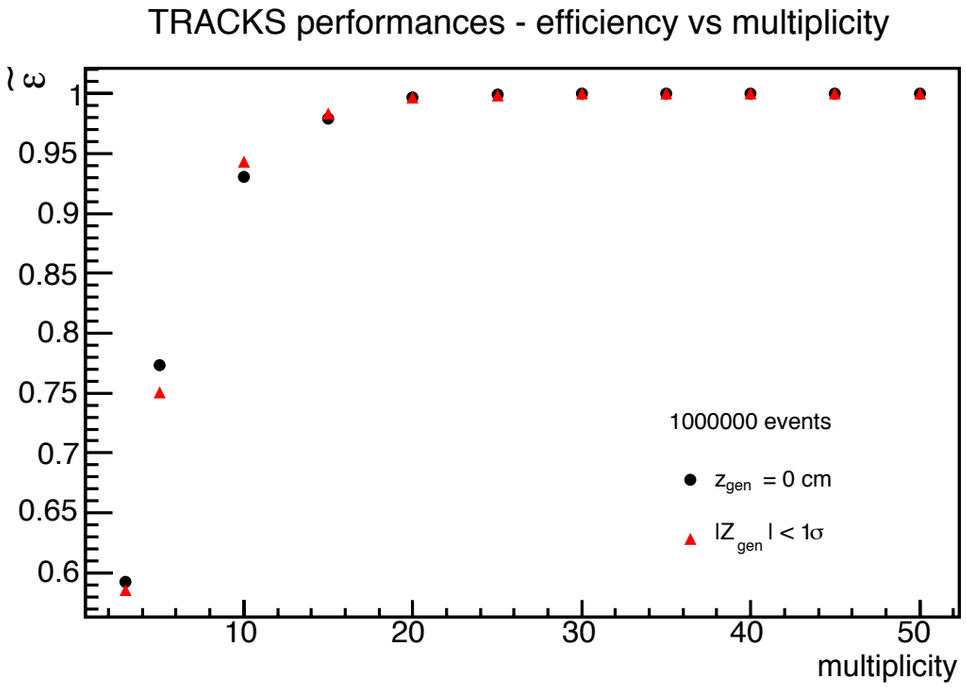
Reconstruction - results and performances

Resolution and efficiency









Reconstruction - results and performances

Peakfinder effects on reconstruction - the resolution differences with different values of amplitude and width are negligible, the values (1,5) are chosen during the simulation since they are the most limiting.