GATE Optical Imaging

SHFJ Status Report

September 25th 2012

HGATE

Gate V6.2 – Optical Module

« GATE_USE_OPTICAL »

- geometry/src/GateOpticalSystem.cc (OpticalSystem)

- digits_hits/src/GateToOpticalRaw.cc + Messenger - the result of the projection is written in a binary output file (+ hdr)

- examples/example_OPTICAL
  - macros for a simple simulation + drawBranches.C

- digits_hits/src/GateToRoot.cc
  - new branches particle momentum direction and m_rootOpticalFlag
  - /gate/output/root/setRootOpticalFlag 0 or 1

- Surface.xml and Materials.xml (values in increasing order of energy – Important if using Geant4.9.5 and Geant4.9.5.p01)

Optical Imaging :
Status of the GPU code

Improvements in Mie scattering

Mie scattering Improvements

The previous code was done for *only for one* value of the photon scattering length. In the new code, we created a table of scattering lenghts versus the photon energy (E, L_{mie}) for each of the 15 materials:

```
__constant__ float Mie_scatteringlength_Table[15][6] =
{
{ 5.0000E-06,  5.3000E+00, 6.0000E-06,  6.2000E+00, 7.0000E-06,  6.7000E+00  } ,
{ 5.0000E-06,  5.3000E+00, 6.0000E-06,  6.2000E+00, 7.0000E-06,  6.7000E+00  } ,
{ 5.0000E-06,  6.3000E+00, 6.0000E-06,  7.2000E+00, 7.0000E-06,  7.7000E+00  } ,
{ 5.0000E-06,  2.3000E+00, 6.0000E-06,  3.2000E+00, 7.0000E-06,  4.7000E+00  } ,
{ 5.0000E-06,  8.3000E+00, 6.0000E-06,  4.2000E+00, 7.0000E-06,  3.7000E+00  } ,
{ 5.0000E-06,  5.7000E+00, 6.0000E-06,  6.9000E+00, 7.0000E-06,  6.7000E+00  } ,
{ 5.0000E-06,  2.3000E+00, 6.0000E-06,  8.2000E+00, 7.0000E-06,  6.2000E+00  } ,
{ 5.0000E-06,  1.3000E+00, 6.0000E-06,  1.5000E+00, 7.0000E-06,  4.9000E+00  } ,
{ 5.0000E-06,  5.8000E+00, 6.0000E-06,  4.2000E+00, 7.0000E-06,  6.7000E+00  } ,
{ 5.0000E-06,  2.3000E+00, 6.0000E-06,  6.9000E+00, 7.0000E-06,  6.0000E+00  } ,
{ 5.0000E-06,  5.3000E+00, 6.0000E-06,  1.2000E+00, 7.0000E-06,  4.7000E+00  } ,
{ 5.0000E-06,  7.3000E+00, 6.0000E-06,  3.7000E+00, 7.0000E-06,  2.9000E+00  } ,
{ 5.0000E-06,  1.3000E+00, 6.0000E-06,  3.7000E+00, 7.0000E-06,  1.7000E+00  } ,
{ 5.0000E-06,  9.3000E+00, 6.0000E-06,  1.2000E+00, 7.0000E-06,  6.5000E+00  } ,
{ 5.0000E-06,  3.3000E+00, 6.0000E-06,  2.2000E+00, 7.0000E-06,  8.7000E+00  }
};
```

+ Log Log interpolation:

```
float loglog(float x, float x0, float y0, float x1, float y1) {
        if (x < x0) {return y0;}
        if (x > x1) {return y1;}
        x0 = 1.0f / x0;
        return powf(10.0f, log10f(y0) + log10f(y1/y0) * log10f(x * x0)/log10f(x1 * x0));
}
```

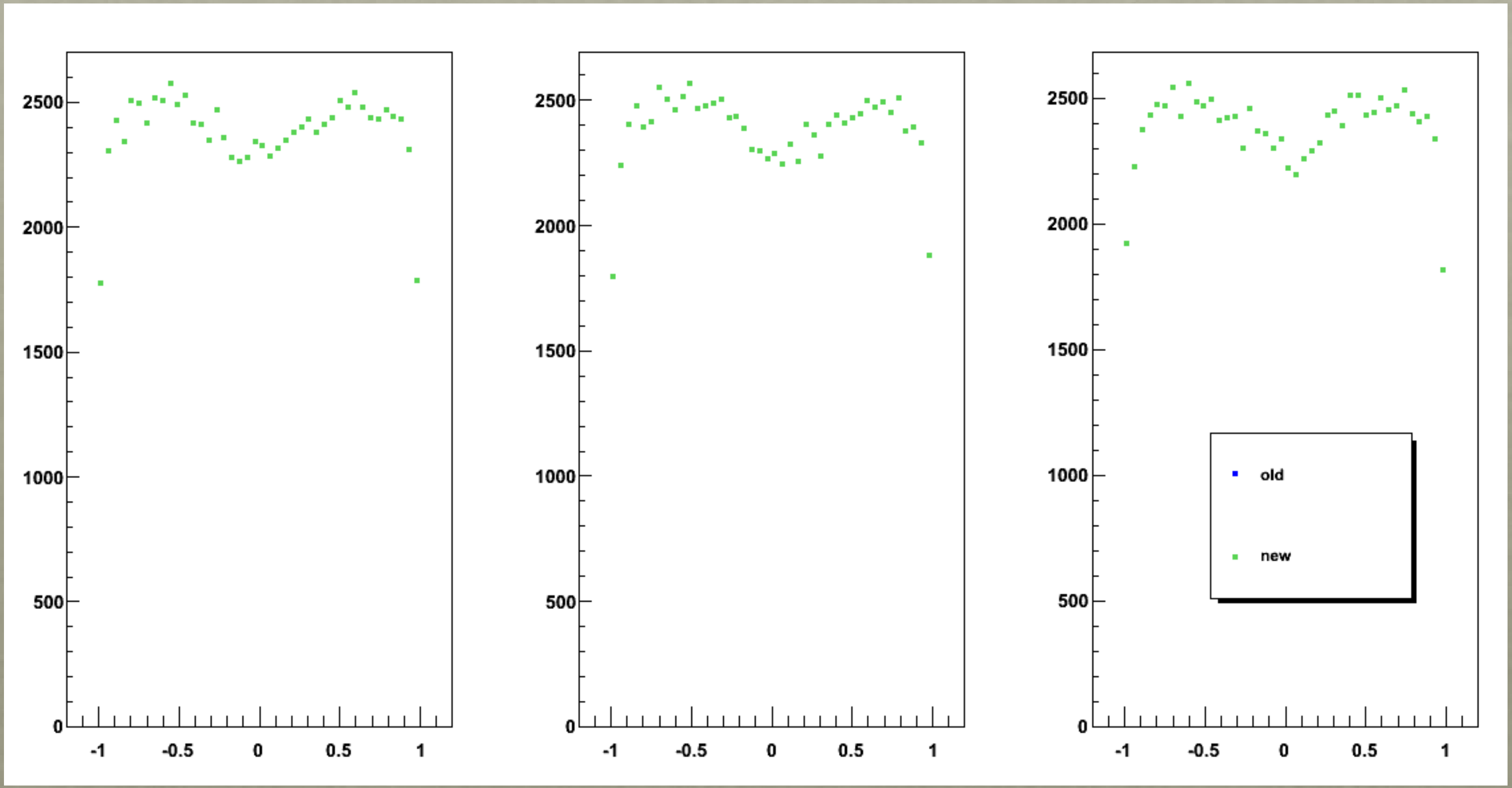
Log Log interpolation - Validation

```
float loglog_result = loglog(5.8000E-06, 5.0000E-06, 5.3,6.0000E-06 , 6.2);
printf("loglog =  %e\n", loglog_result);
```

→ loglog = 6.021795

Old Code : unique L_{Mie} = 6.021795

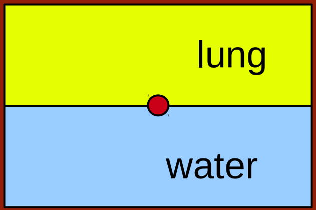
New Code : Mie scattering length from Table + LogLog interpolation



Optical Imaging :
Status of the GPU code

Fresnel Process at Boundaries

Simple two materials benchmark to test Gate GPU code



Isotropic photon source of 5.8eV

Anisotropy is 0.6 for both water and lung.

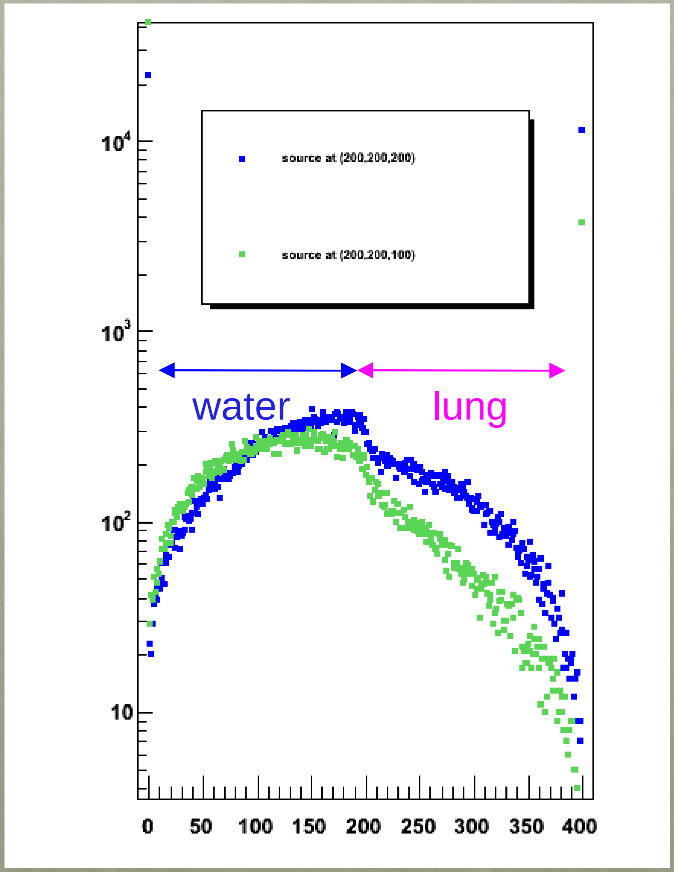
Most att. material : lung

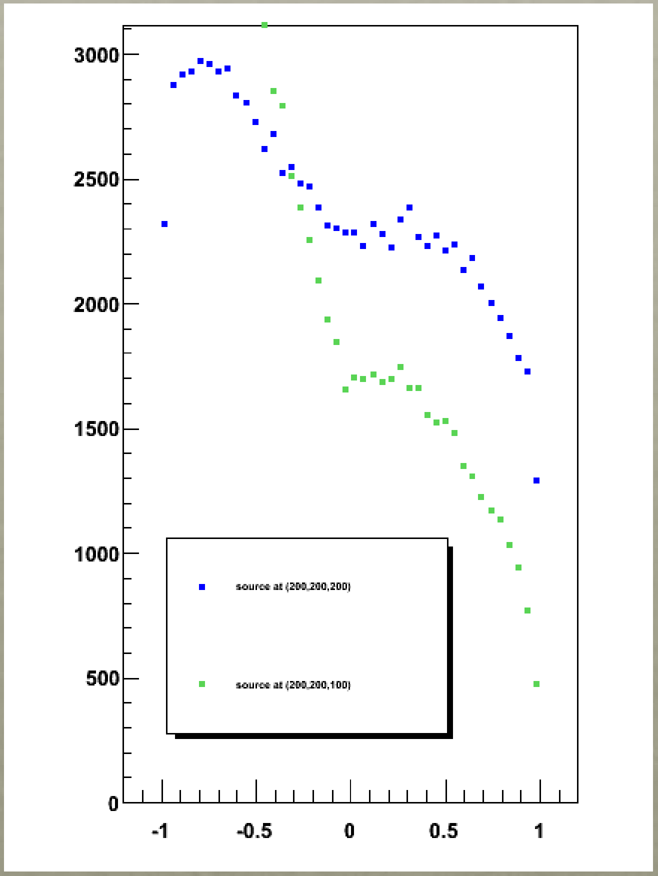
Input table for scattering lengths + LogLog interpolation :

Water = 6.021795mm
Lung = 3.009411mm

Source position :
(200,200,200)mm
(200,200,100)mm

ONLY Mie scattering physics process

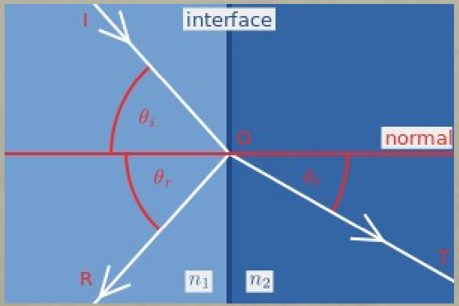




Implementation of the Fresnel Reflectance (following the MCML code)

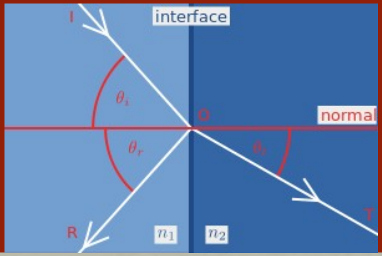
The fraction of the incident light that is reflected from the interface is given by the reflectance R and the fraction that is refracted by the transmittance T=(1-R).

```
// Compute the Fresnel reflectance
__device__ float RFresnel(float n_incident, /* incident refractive index.*/
                          float n_transmit, /* transmit refractive index.*/
                          float c_incident_angle, /* cosine of the incident angle. 0<a1<90 degrees. */
                          float *c_transmission_angle_Ptr) /* pointer to the cosine of the transmission angle. a2>0. */
{
  float r;

  if(n_incident==n_transmit) {                    /** matched boundary. **/
    *c_transmission_angle_Ptr = c_incident_angle;
    r = 0.0;
  }
  else if(c_incident_angle>COSZERO) {   /** normal incident. **/
    *c_transmission_angle_Ptr = c_incident_angle;
    r = (n_transmit-n_incident)/(n_transmit+n_incident);
    r *= r;
  }
  else if(c_incident_angle<COS90D)  {   /** very slant. **/
    *c_transmission_angle_Ptr = 0.0;
    r = 1.0;
  }
  else  {                /** general. **/
    float sa1, sa2;      /* sine of the incident and transmission angles. */
    float ca2;

    sa1 = sqrt(1-c_incident_angle*c_incident_angle);
    sa2 = n_incident*sa1/n_transmit;
    if(sa2>=1.0) {        /* double check for total internal reflection. */
      *c_transmission_angle_Ptr = 0.0;
      r = 1.0;
    }
    else  {
      float cap, cam;   /* cosines of the sum ap or difference am of the two */
                        /* angles. ap = a_incident+a_transmit am = a_incident - a_transmit. */
      float sap, sam;   /* sines. */

      *c_transmission_angle_Ptr = ca2 = sqrt(1-sa2*sa2);

      cap = c_incident_angle*ca2 - sa1*sa2; /* c+ = cc - ss. */
      cam = c_incident_angle*ca2 + sa1*sa2; /* c- = cc + ss. */
      sap = sa1*ca2 + c_incident_angle*sa2; /* s+ = sc + cs. */
      sam = sa1*ca2 - c_incident_angle*sa2; /* s- = sc - cs. */
      r = 0.5*sam*sam*(cam*cam+cap*cap)/(sap*sap*cam*cam);
    }
  }
  return(r);
}
```



Validation: Rfresnel(1.2, 1.4, 0.7, &ca_transmission) = 0.00953686 OK!

Fresnel Process Implementation: Reflection and Refraction



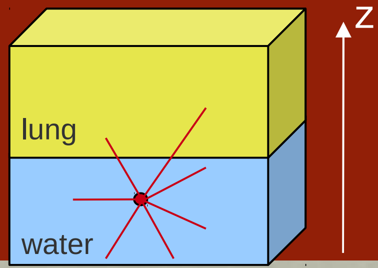

```
__device__ float3 Fresnel_process(StackGamma stackgamma, unsigned int id, unsigned short int *mat_i_Ptr, unsigned short int mat_t) {

  float uz = stackgamma.dz[id]; /* z directional cosine. */
  float uz1;     /* cosines of transmission angle. */
  float r=0.0;   /* reflectance */
  float ni = mat_Rindex[*mat_i_Ptr];
  float nt = mat_Rindex[mat_t];

  /* Get r. */
  if( uz <= 0.7) /* 0.7 is the cosine of the critical angle of total internal reflection */
    r=1.0;                /* total internal reflection. */
  else r = RFresnel(ni, nt, uz, &uz1);


  if (Brent_real(id, stackgamma.table_x_brent, 0) > r) {        /* transmitted */
      stackgamma.dx[id] *= ni/nt;
      stackgamma.dy[id] *= ni/nt;
      stackgamma.dz[id] = uz1;
    }
  else {                                        /* reflected. */
    stackgamma.dz[id] = -uz;
}
      return make_float3(stackgamma.dx[id], stackgamma.dy[id], stackgamma.dz[id]);

}  // vesna - Fresnel Process
```

← Calculation of the Reflectance.

Total Internal Reflection : $\theta > \theta_c$ no light can pass through the surface. All the light is reflected.

← Code returns the photon NEW direction.

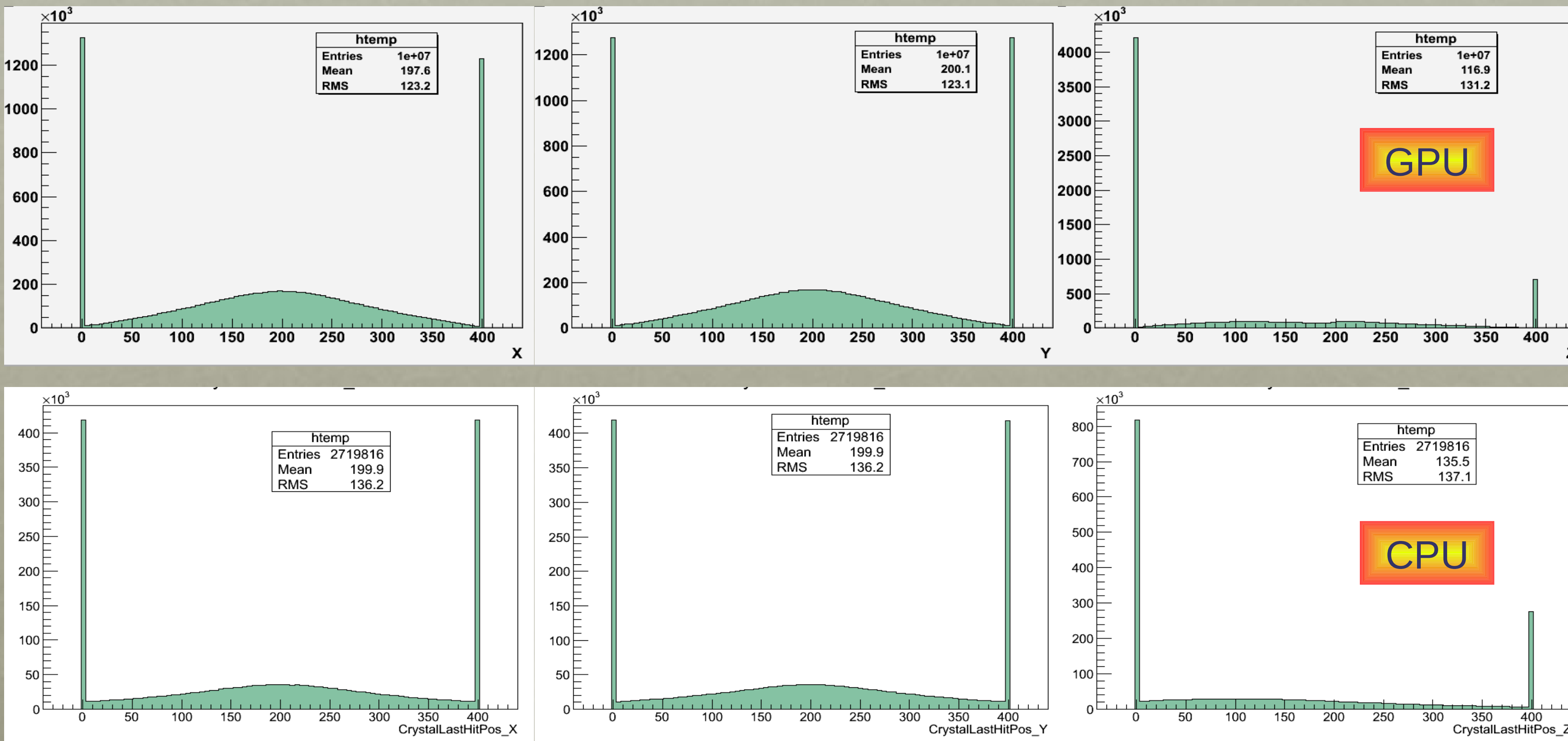
Problem:
When using the woodcock tracking implemented in the GPU standalone code, we apply the Fresnel equations at the next particle position which is <u>not</u> the surface between 2 materials. We need to modify the tracking so that the new particle position where we apply the Fresnel process is on the interface between two materials.

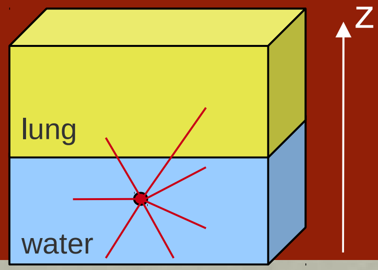


- voxelized phantom : 100x100x100
- voxel size = 4mm
- most_att_mat = lung (tracking)
- 5.8eV optical photons
- statistics : 10M

- Isotropic source at (200,200,100)mm
- NO total internal reflection
- R_{index}^{water} = 1.2 and R_{index}^{lung} = 1.4
- Anisotropy = 0.6 for both materials
- Different Mie scattering properties

\+ loglog interpolation

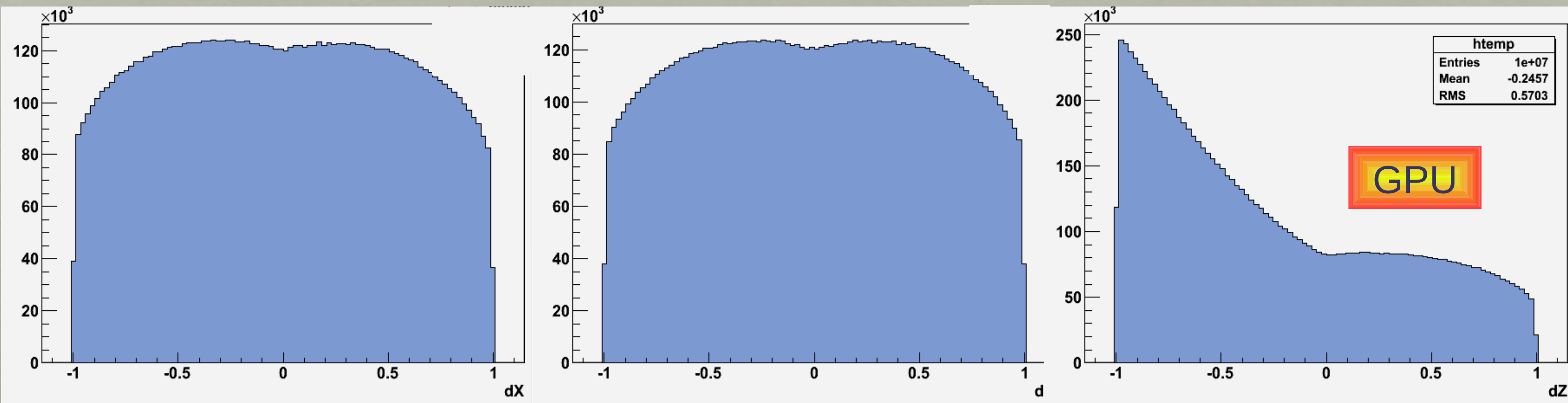


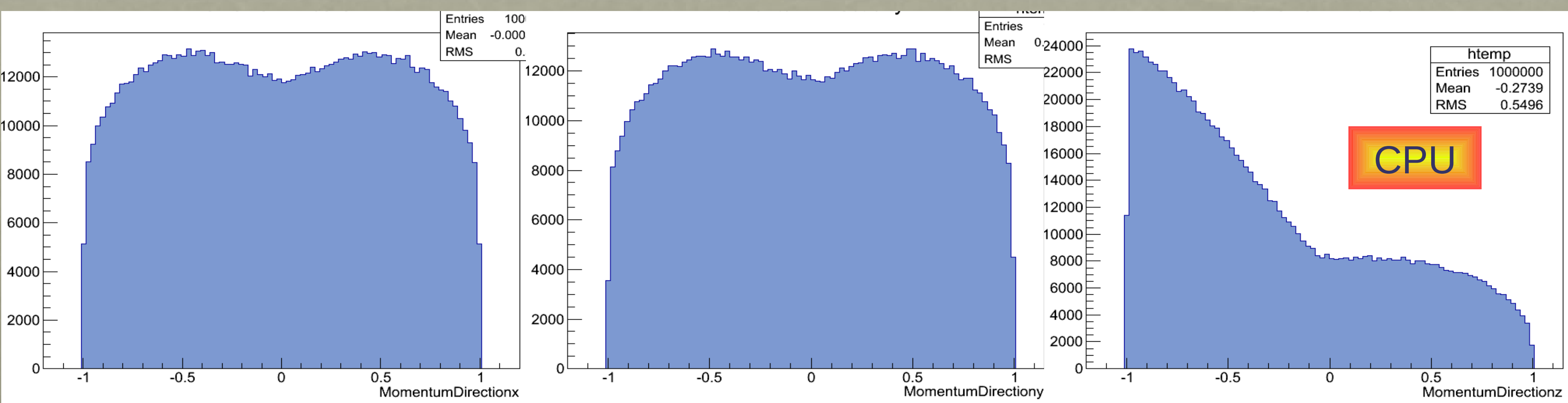


- voxelized phantom : 100x100x100
- voxel size = 4mm
- most_att_mat = lung (tracking)
- 5.8eV optical photons
- statistics : 10M

- Isotropic source at (200,200,100)mm
- NO total internal reflection
- R_{index}^{water} = 1.2 and R_{index}^{lung} = 1.4
- Anisotropy = 0.6 for both materials
- Different Mie scattering properties

+ loglog interpolation





Conclusion and Next Step

- Improvements in the simulation of the Mie scattering process:
  - Table of scattering lenghts
  - LogLog interpolation

- Implementation of the Fresnel Reflectance

- Implementation of the Fresnel process : Reflection and Refraction

- First attempt for a validation of the GPU code against GATE CPU :

Distributions of the optical photon position and direction show a similar « behavior » between GPU and CPU code but we do not validate yet.

Main missing piece for a correct validation is the particle tracking.