GENIE-GEANT4 Interface

Ritesh Kumar Pradhan
Central University of Karnataka

---

1 GENIE

https://hep.ph.liv.ac.uk/~costasa/genie/index.html

To install GENIE, all steps can be found at https://hep.ph.liv.ac.uk/~costasa/genie/get_started.html.

The GENIE use manual can be found at https://genie-docdb.pp.rl.ac.uk/DocDB/0000/000002/007/man.pdf.

GENIE computes cross sections splines for a target and given particles. For ICAL simulation, the cross section file can be generated by following command,

```
$ gmkspl -p 14,-14 -t 1000260560 -o ical-cross-sections.xml -e 10 -n 150
```

This will generate cross section splines for ν_μ-iron & $\bar{\nu}_\mu$-iron interactions from 0.01 GeV to 10 GeV with 150 knots and save it as *ical-cross-sections.xml*.

Now, generation of events for INO-ICAL geometry and realistic flux using the cross section spline can be done by following command,

```
$ gevgen_atmo -f FLUKA:inohondaflux_energyweight.dat[14] -g icalgeometry.root -n 100000 -E
    1,10 -o output -r 1 -seed 17187 --cross-sections ical-cross-sections.xml
```

This will generate 100000 ν_μ events with the ICAL geometry with energy ranging from 1 GeV to 10 GeV using the INO Honda flux and the cross section generated from GENIE. The output file will be saved as *output.1.ghep.root*.

We can convert the output data file to ntuple format using following command,

```
$ ntpc -ioutput.1.ghep.root -f gst -o output.root
```

Now the output data file is *output.root* which contains the informations about the out-coming particles from the neutrino interactions, which will be given to Geant4.

2 Interfacing with GEANT4

We can give all informations from data file to GEANT4 in the *PrimaryGeneratorAction.cc* file. Include required headers in the *PrimaryGeneratorAction.hh* file. Read the data file and get the tree and branches from the root file using the following codes,

```
//reading the data file
input = TFile::Open("withflux31.root", "READ");
gst = (TTree*)input->Get("gst");

entry = gst->GetEntries(); //total number of entries in the tree

//interaction and scattering
gst->SetBranchAddress("cc",&cc);
gst->SetBranchAddress("nc",&nc);
gst->SetBranchAddress("qel",&qes);
gst->SetBranchAddress("mec",&mec);
gst->SetBranchAddress("dis",&dis);
gst->SetBranchAddress("coh",&coh);
gst->SetBranchAddress("res",&res);

//data for leptons
gst->SetBranchAddress("iev",&id);
gst->SetBranchAddress("fspl",&pdg);
gst->SetBranchAddress("El",&E);
gst->SetBranchAddress("pxl",&px);
gst->SetBranchAddress("pyl",&py);
gst->SetBranchAddress("pzl",&pz);

//vertex
gst->SetBranchAddress("vtxx",&vx);
gst->SetBranchAddress("vtxy",&vy);
gst->SetBranchAddress("vtxz",&vz);
```

Get the Event ID from Geant4 and get the data from root file such that the event id in root file should be same as event id from Geant4.

```
G4int eventid = anEvent->GetEventID(); //G4 Event ID
// G4cout<<"eventid is "<<eventid<<G4endl;

gst->GetEntry(eventid);  //getting entry corresponding to eventID
```

We defined two guns; one for leptons and another one for hadrons.
Set the particle gun definition for leptons.

```
//for leptons

fParticleGun  = new G4ParticleGun(1);
G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
G4ParticleDefinition* particle  = particleTable->FindParticle(pdg);
fParticleGun->SetParticleDefinition(particle);

double p = sqrt(px*px+py*py+pz*pz);  //momentum
fParticleGun->SetParticleMomentumDirection(G4ThreeVector(px/p,py/p,pz/p));

fParticleGun->SetParticleEnergy(E *GeV);
fParticleGun->SetParticleMomentum(p *GeV);
fParticleGun->SetParticlePosition(G4ThreeVector(vx *m,vy *m ,vz*m));
fParticleGun->GeneratePrimaryVertex(anEvent);
```

Now, set the gun definition for hadrons.

```
//2nd gun definition for hadrons

int n=gst->GetLeaf("nf")->GetValue();   //number of hadrons in one event

//getting data for hadrons

for (int j =0 ;j<n;j++){
int pdgh=gst->GetLeaf("pdgf")->GetValue(j);  //pdg of final state hadrons
double pxh=gst->GetLeaf("pxf")->GetValue(j);
```

```
double pyh=gst->GetLeaf("pyf")->GetValue(j);
double pzh=gst->GetLeaf("pzf")->GetValue(j);
double Eh=gst->GetLeaf("Ef")->GetValue(j);

fParticleGun1  = new G4ParticleGun(n);
G4ParticleTable* particleTable1 = G4ParticleTable::GetParticleTable();
if(pdgh<10002605)     //excluding iron (there is iron in final hadronic system of coh
    scattering)
{
G4ParticleDefinition* particle1  = particleTable1->FindParticle(pdgh);
fParticleGun1->SetParticleDefinition(particle1);

double ph = sqrt(pxh*pxh+pyh*pyh+pzh*pzh);  //momentum
fParticleGun1->SetParticleMomentumDirection(G4ThreeVector(pxh/ph,pyh/ph,pzh/ph));

fParticleGun1->SetParticleEnergy(Eh *GeV);
fParticleGun1->SetParticleMomentum(ph *GeV);
}
fParticleGun1->SetParticlePosition(G4ThreeVector(vx *m,vy *m ,vz*m));
fParticleGun1->GeneratePrimaryVertex(anEvent);
```

3 Defining New Commands in GEANT4

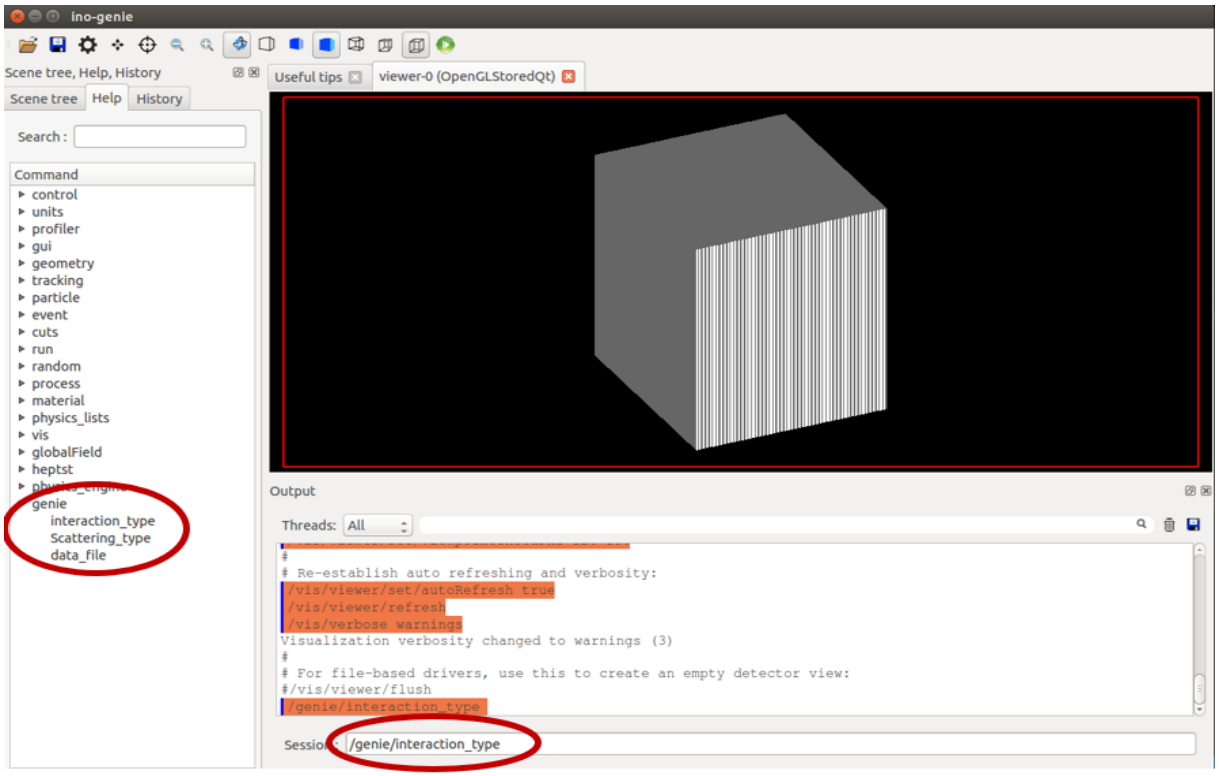
Three new commands are defined in GEANT4 so that one can shoot particle for a particular interaction type (cc or nc) and a scattering type (qes, mec, res, dis, coh).
New commands can be defined by using G4GenericMessenger as follow,

```
#include "G4GenericMessenger.hh"

G4GenericMessenger *fMessenger;

//creating messengers for input
fMessenger = new G4GenericMessenger(this,"/genie/");
fMessenger->DeclareProperty("interaction_type",interaction_type,"interaction type");
fMessenger->DeclareProperty("Scattering_type",Scattering_type,"scattering type");
fMessenger->DeclareProperty("data_file",data_file,"genie data file");
```

This will create three new commands as shown in the figure,



Filter the event id in the root file as per input and put it in an array so that we can take entries for those ids corresponding to Geant4 event id. Example: for pair cc and qes, the id's from root file are 9,14,18,23, etc. So, when you generate 2 events in Geant4, the 1st event will take entry for 9th id and 2nd event will take entry for 14th id from the root file. This is done by the following code,

```
//filtering
counter=0;
for (int i=0;i<entry;i++)
{
  gst->GetEntry(i);
  if (interaction_type=="cc"){interaction=cc;}
  if (interaction_type=="nc"){interaction=nc;}
  if (Scattering_type=="qes"){scattering=qes;}
  if (Scattering_type=="mec"){scattering=mec;}
  if (Scattering_type=="dis"){scattering=dis;}
  if (Scattering_type=="coh"){scattering=coh;}
  if (Scattering_type=="res"){scattering=res;}
  if (interaction && scattering)
  {
    counter++;
//  cout<<id<<endl;
    arr[counter]=id;   //array for filtered ID
  }
}
```

4 Example

One can shoot particle for interaction type cc and scattering type qes by the following command,

```
/genie/interaction_type cc
/genie/scattering_type qes
/run/beamOn 1
```

A charged-current quasi-elastic event in the detector is shown in the figure 1. There is a bent path, that's for muon as it's a charged-current event and other particles are p, n, π^0 after the neutrino interaction.

A neutral-current quasi-elastic event in the detector is shown in the figure 2. There is a straight line going away from the vertex without any interaction, that's for ν_μ as it's a neutral-current event and other particles are protons and neutrons after the neutrino interaction.

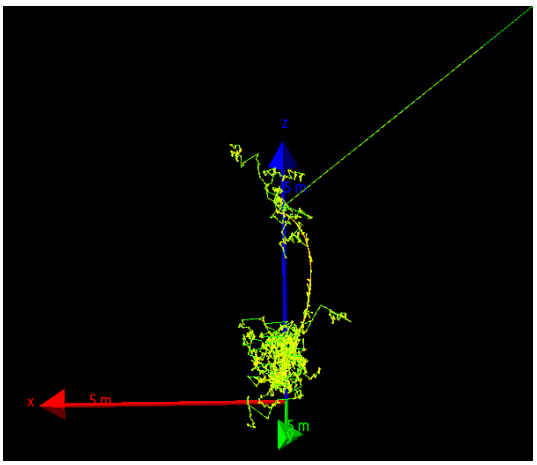


Figure 1: A charged-current Quasi-elastic event in the detector.

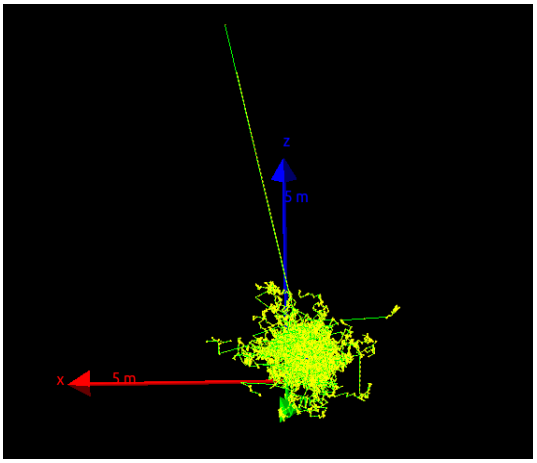


Figure 2: A neutral-current Quasi-elastic event in the detector.