Exercise 1

- Make a primary generator action using either of the 2 possibilities
  - For the process “muon pair production in electron-positron collision” at 14 GeV
  - Inverse beta decay for neutrino beams of energy 5-10 MeV

tion. These are ready-to-use complete physics lists provided by the toolkit and are routinely validated and updated with each release. Geant4 provides several reference physics lists that are suited for different use cases.

NuSD uses QGSP-BERT-HP/QGSP-BIC-HP ("HP" for high-precision neutron interactions) reference physics list for the simulation of IBD events. The QGSP-BERT-HP includes standard electromagnetic and hadronic physics processes and is specifically suited for the transport of neutrons below 20 MeV down to thermal energies [12]. This modular physics list covers all the well-known processes such as ionization, Coulomb scattering, Bremsstrahlung, Compton scattering, photoelectric effect, pair production, annihilation, decay, radiative capture, fission, hadronic elastic, and inelastic scattering. This reference physics list does not include optical processes. To include them, G4OpticalPhysics, the constructor of optical processes, is instantiated and then registered in the list. This automatically adds all the optical physics processes.

2.5. Primary particles

NuSD provides two different primary particle generation classes; NuSDPrimaryGeneratorAction and NuSDGenericPrimaryGeneratorAction. The first one is used to generate IBD events, while the second one is used to simulate background events or any desired particle. These two classes are inherited from the NuSDVPrimaryGeneratorAction abstract class and are used to identify the primary particles to be generated as well as setting their initial values for energy, momentum, position, and time. The NuSDVPrimaryGeneratorAction class itself is derived from the G4VUserPrimaryGeneratorAction class and includes some additional member functions for generating random positions within the active volume of NuSD detectors. Fig. 4 shows the UML diagrams of the primary generator classes used by NuSD.

The initial position of an event can be specified in one of two ways in NuSD. The first one is to generate an event at a random point inside the detector; and the second one is to generate an event at a user-specified position. In the first option, which is accomplished by setting the command /NuSD/gun/isRandomInitPos to true, a random unit is first selected, then a random point is generated within that unit. If the command is set to false, a random unit is selected again, but this time the position is specified relative to the selected unit's center via the command /NuSD/gun/eventInitialPositon.

NuSD generates an IBD event by directly creating its products as primary particles due to the neutrino's extremely low interaction cross-section. By default, NuSD simulates reactor neutrinos (1-10 MeV energy range) and performs the following steps to determine the initial energy of the positron and neutron;

- Eq. (1) is used to sample neutrino energies. Here $P(E_{\bar{\nu}})$ is the detecting probability of a reactor neutrino at energy $E_{\bar{\nu}}$, $\Phi_i(E_{\bar{\nu}})$ is the emitted reactor neutrino energy spectrum per fission of each of the four isotopes, $\sigma(E_{\bar{\nu}})$ is the neutrino-proton interaction cross section at zeroth-order, α_i is the average fission fraction of each isotope over the reactor fuel cycle, and N is the normalization constant.
- Eq. (2) is used to sample the cosine of the scattering positron angle [7].
- Using the ROOT analysis framework, two histograms, one for the initial neutrino energy and the other for the scattering positron angle, are generated and written into a ROOT file.
- The NuSDPrimaryGeneratorAction class uses this ROOT file to get a neutrino energy and a positron scattering angle from the written histograms for each event.
- The positron energy E_{e^+} is computed from Eq. (3). Here m_e is the electron mass, M is the neutron mass, and the Δ is the mass difference of the neutron and proton.
- The neutron energy T_n is calculated from Eq. (4).

$$P(E_{\bar{\nu}}) = \frac{1}{N} \sum_{i=5,8,9,1} \alpha_i \Phi_i(E_{\bar{\nu}}) \sigma(E_{\bar{\nu}}) \tag{1}$$

$$P(\cos\theta) = \frac{1}{N} [1 - 0.1 \cos\theta] \tag{2}$$

$$E_{e^+}^{(1)} = E_{e^+}^{(0)} \left[1 - \frac{E_\nu}{M}(1 - \cos\theta)\right] - \frac{\Delta^2 - m_e^2}{2M} \tag{3}$$

$$T_n = \frac{E_\nu (E_\nu - \Delta)}{M}(1 - \cos\theta) + \frac{\Delta^2 - m_e^2}{2M} \tag{4}$$

As an alternative to the above, a NuSD user can manually set the parameters of the positron and neutron with the supplied user interface commands (see Fig. 4).

The NuSDGenericPrimaryGeneratorAction class is implemented to simulate electrons by default. The user can alter the type of particle and its initial setting using the supplied user commands.

Fig. 5 shows the initial energy distributions of the positron and neutron at the top, and their scattering angle distribution at the bottom. From the bottom of the Figure, it can be seen that the average positron direction is slightly backward and the neutron direction is purely forward.

Exercise 2

- Generate Compton scattering by modifying the Geant4 code for incident photons of 0.5 MeV, 1.0 MeV and 1.5 MeV and compare the distribution with those obtained from exercise 2 of yesterday.

- Steps to be taken

    - Make a process which activates only electron, gamma

    - Add only Compton scattering and transportation as processes

    - Physics list. Is to be modified to register only this class

Exercise 3

Add a new particle XXX of mass 2000 GeV and charge +1 to the simulation process

Add ionisation and transportation for this new particle

Propagate this new particle in the example code

Steps to be taken

- Create a new class for the particle definition
- Create a class to add the processes of the new particle
- Register this class in the physics list
- Modify the macro to generate 20 events for this particle and also for $\mu+$ and compare the energy loss in silicon tracker between the two cases

```
#ifndef G4Electron_h
#define G4Electron_h 1

#include "globals.hh"
#include "G4ios.hh"
#include "G4ParticleDefinition.hh"

// ######################################################################
// ###                          ELECTRON                              ###
// ######################################################################

class G4Electron : public G4ParticleDefinition
{
 private:
   static G4Electron* theInstance;
   G4Electron(){}
   ~G4Electron(){}

 public:
   static G4Electron* Definition();
   static G4Electron* ElectronDefinition();
   static G4Electron* Electron();
};

#endif
```

```
#include "G4Electron.hh"
#include "G4PhysicalConstants.hh"
#include "G4SystemOfUnits.hh"
#include "G4ParticleTable.hh"

// ######################################################################
// ###                          ELECTRON                              ###
// ######################################################################
G4Electron* G4Electron::theInstance = 0;

G4Electron* G4Electron::Definition()
{
  if (theInstance !=0) return theInstance;
  const G4String name = "e-";
  // search in particle table]
  G4ParticleTable* pTable = G4ParticleTable::GetParticleTable();
  G4ParticleDefinition* anInstance = pTable->FindParticle(name);
  if (anInstance ==0)
  {
  // create particle
  //
  //    Arguments for constructor are as follows
  //               name             mass          width         charge
  //             2*spin           parity  C-conjugation
  //          2*Isospin       2*Isospin3       G-parity
  //               type    lepton number  baryon number   PDG encoding
  //             stable         lifetime    decay table
  //             shortlived      subType    anti_encoding

  // use constants in CLHEP
  //  static const double electron_mass_c2 = 0.51099906 * MeV;

    anInstance = new G4ParticleDefinition(
                 name,  electron_mass_c2,       0.0*MeV,    -1.*eplus,
        1,                  0,              0,
        0,                  0,              0,
      "lepton",                 1,              0,          11,
     true,              -1.0,           NULL,
            false,                 "e"
           );
    // Bohr Magnetron
   G4double muB =  -0.5*eplus*hbar_Planck/(electron_mass_c2/c_squared) ;

   anInstance->SetPDGMagneticMoment( muB * 1.00115965218076 );

  }
  theInstance = reinterpret_cast<G4Electron*>(anInstance);
  return theInstance;
}

G4Electron*  G4Electron::ElectronDefinition()
{
  return Definition();
}

G4Electron*  G4Electron::Electron()
{
  return Definition();
}
```

```
void XXX::ConstructParticle() {
  G4Electron::Electron();
}

void XXX::ConstructProcess() {

  G4ParticleDefinition* particle = G4Electron::Electron();

  G4PhysicsListHelper* ph = G4PhysicsListHelper::GetPhysicsListHelper();
  G4LossTableManager* man = G4LossTableManager::Instance();

  G4eIonisation* eioni = new G4eIonisation();

  G4eMultipleScattering* msc = new G4eMultipleScattering;
  G4UrbanMscModel* msc1 = new G4UrbanMscModel();
  G4WentzelVIModel* msc2 = new G4WentzelVIModel();
  msc1->SetHighEnergyLimit(highEnergyLimit);
  msc2->SetLowEnergyLimit(highEnergyLimit);
  msc->SetEmModel(msc1);
  msc->SetEmModel(msc2);


  // single scattering
  G4eCoulombScatteringModel* ssm = new G4eCoulombScatteringModel();
  G4CoulombScattering* ss = new G4CoulombScattering();
  ss->SetEmModel(ssm);
  ss->SetMinKinEnergy(highEnergyLimit);
  ssm->SetLowEnergyLimit(highEnergyLimit);
  ssm->SetActivationLowEnergyLimit(highEnergyLimit);

  ph->RegisterProcess(msc, particle);
  ph->RegisterProcess(eioni, particle);
  ph->RegisterProcess(new G4eBremsstrahlung(), particle);
  ph->RegisterProcess(ss, particle);
}
```

Exercise for the Future

- Create a new process and add to Geant4 library
-