ELABORATO PER L'ESAME
"SIMULAZIONE MONTECARLO DI RIVELATORI DI RADIAZIONE"

Matteo Bagnalasta - A.A. 2017/2018

Esercizio 1 – Task 1

Lo scopo di questo primo esercizio è simulare un rivelatore composto da diverse sottoparti.

- ***In generale fate attenzione ai file con #include***

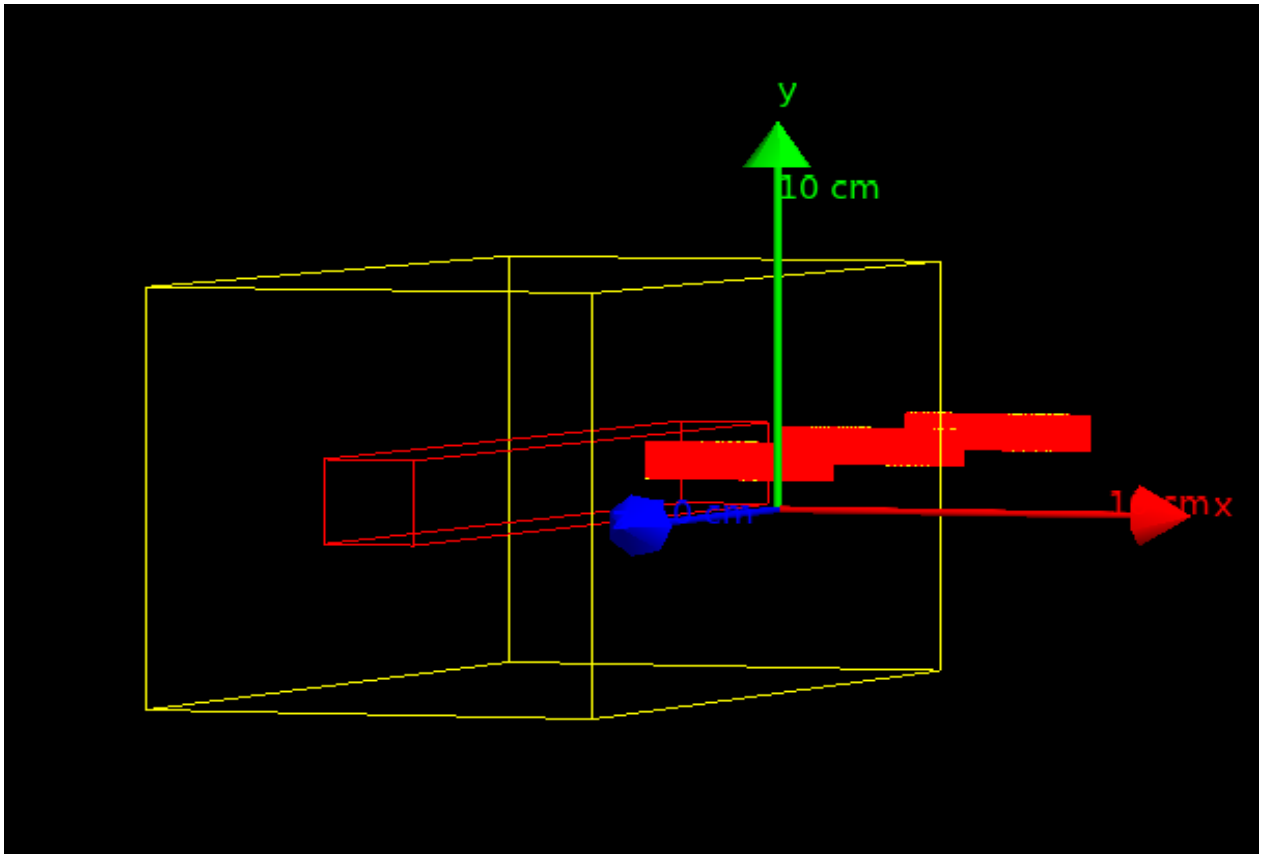
Per poter compilare correttamente l'esercizio è stato necessario includere l'header *SystemOfUnits.h,* disponibile nella folder di installazione del software CLHEP. In questo file vengono definite tutte le unità di misura poi utilizzate nell'esercizio.
L'header è stato incluso analogamente per tutti i task successivi.

- ***Modificare DetectorConstruction.cc***

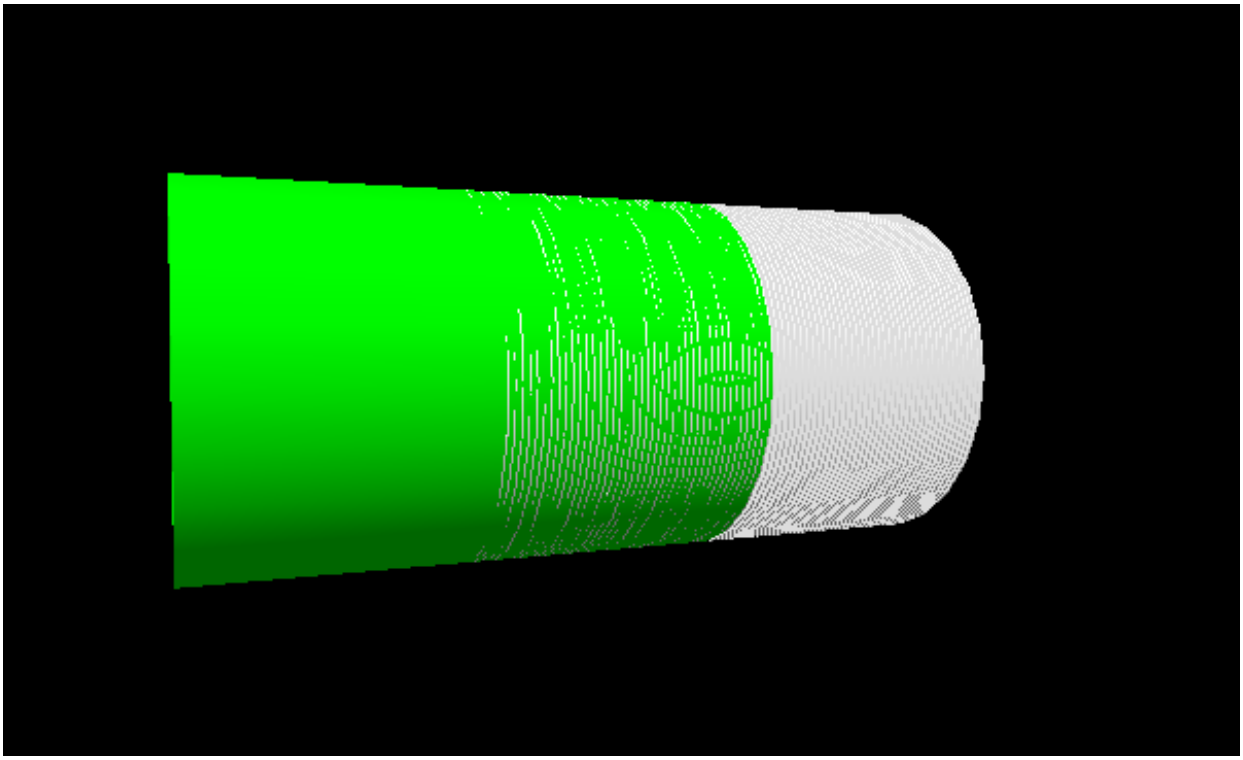
La classe *DetectorConstruction* (derivante da *G4VUserDetectorConstruction)* è una classe che deve essere fornita obbligatoriamente dall'utente. In questa classe si definiscono i materiali da utilizzare, le forme dei vari componenti del rivelatore, il loro posizionamento nello spazio e gli elementi attivi del rivelatore (i cosiddetti *Sensitive Detectors*).

Il detector si compone di più parti:

- **3 tracker di Silicio** (in rosso in figura *1.a*) composti ognuno da 600 strip, costruite con il metodo *Replica.*
- **1 calorimetro elettromagnetico** formato da due sotto-strutture in Tungstato di Piombo (i parallelepipedi in rosso e in giallo in figura *1.a*)
- **1 calorimetro adronico** costruito con 80 strati di ferro alternati a strati di argon liquido come elemento sensibile (figura *1.b*).



1.a



1.b

Si è modificato il file *DetectorConstruction.cc* costruendo uno uno strato solido del calorimetro adronico in argon liquido.
Si è per prima cosa costruito il cilindro solido *"HadCaloLayerSolid"* tramite la classe *G4Tubs*. I parametri impostati sono, come richiesto dalla consegna, raggio interno *0 mm*, raggio esterno *800 mm*, lunghezza *4 mm*, angolo iniziale *0 rad* ed angolo finale 2π.
Si è poi proceduto alla costruzione del volume logico *HadLayerLogic*, con puntatore a *HadCaloLayerSolid,* scegliendo come materiale l'Argon liquido.
Infine si è proceduto a completare la costruzione della componente del rivelatore posizionando fisicamente il layer tramite la classe *G4PVPlacement,* puntando il volume logico *HadLayerLogic* e scegliendo come volume logico madre il volume *LogicWorld*. Sempre in *G4PVPlacement* si posizionano 80 copie dello strato di argon liquido tramite il metodo Replica.

- ***Nella physics list che particelle sono abilitate? → Abilitare muoni e pioni***

La classe *PhysicsList* (derivante da *G4VUserPhysicsList*) definisce la fisica della simulazione: i tipi di particelle da simulare, i processi fisici attivi e i tagli in energia per la generazione di particelle secondarie.

Sono disponibili diverse *PhysicsLists* preimpostate che l'utente può richiamare all'interno della sua classe: in questo esercizio viene usata un'istanza di *G4EmStandardPhysics*, la *PhysicsList* standard per i processi elettromagnetici (in essa si definiscono le particelle *e-*, *e+* e qualche adrone carico). Viene inoltre definita la pseudo-particella *Geantino.*

Per abilitare muoni e pioni basta aggiungere la loro definizione nella funzione *ConstructParticle()* :

```
void PhysicsList :: ConstructParticle ()
{
G4Electron :: Electron () ;
G4Positron :: Positron () ;
G4MuonPlus::MuonPlus();
G4MuonMinus::MuonMinus();
}
```

- ***Nella PrimaryGenerationAction → Comprendere come è fatta la sorgente primaria***

La classe *PrimaryGeneratorAction* permette all'utente di impostare le caratteristiche delle particelle iniziali in un evento.
Geant4 mette a disposizione due classi per la generazione degli eventi: G4ParticleGun e G4GeneralParticleSource.
In questo esercizio *G4VPrimaryGenerator* è implementato con il metodo G4GeneralParticleSource (*GPS*). Le caratteristiche della sorgente primaria sono:

- la sorgente primaria è composta da $\pi+$
- la distribuzione energetica è monocromatica a *2 GeV*
- la sorgente è posizionata nel punto *(0cm, 0cm, -80cm)*
- la deviazione standard del profilo di posizione del beam è di 0.1 mm, sia per la direzione x che per quella y (area circolare)
- la direzione del momento è l'asse *z*
- la distribuzione angolare è *"beam2d"*
- la deviazione standard del profilo direzionale del beam è di *0.1 mrad*, sia per la direzione *x* che per quella y
- l'asse di riferimento per la distribuzione angolare è l'asse *(-1,0,0)*

- ***Guardare file vis.mac***

Una simulazione Geant4 viene eseguita dall'utente passando al componente G4UIManager una macro contenente una serie di comandi. Molti componenti di Geant4 possono essere infatti sia configurati all'interno del codice o direttamente con comandi macro.
In questo esercizio la macro presente è composta dai seguenti comandi:

- *`/control/verbose 2 , /run/verbose 2`*
  Due comandi per settare i valori di default del verbose.
- *`/vis/open OGLSX 600x600-0+0`*
  Comando per iniziare una Uisession per un specifico sistema grafico.
- *`/vis/drawVolume`*
  Comando per disegnare la geometria.
- *`/vis/viewer/set/viewpointThetaPhi 90 180 deg , /vis/viewer/zoom 0.8`*
  Due comandi per settare alcuni parametri di visualizzazione.
- *`/vis/scene/endOfEventAction accumulate`*
  Comando per accumulare tutti gli eventi di una data run.
- *`/run/beamOn 10`*
  Comando per iniziare una run di 10 eventi processati sequenzialmente.

Esercizio 2 – Task 2

In questo esercizio sono presentati gli strumenti per analizzare l'interazione (*Hit*) delle particelle con le parti sensibili del detector (i *SensitiveDetector*).
Viene inoltre definita un'interfaccia utente per il detector attraverso la classe *DetectorMessenger* che permette di controllare dalla macro la posizione del secondo livello del telescopio.

Detector →Che cosa è?

Il detector è un telescopio a tre livelli, composto da tre sensori piani di silicio lunghi *10 mm* e larghi *300 µm*. I sensori sono composti ognuno da 48 strip di silicio con pitch di *50 µm,* create tramite la classe *G4PVReplica*.

Come accennato nel precedente esercizio, alcune parti del detector sono considerate attive, ovvero in grado di registrare informazioni sulla particella che li attraversa. Queste componenti vengono chiamate *SensitiveDetectors*. Le informazioni registrate vengono chiamate Hit e caratterizzano una singola interazione della particella con il materiale.
In questo specifico esercizio, terminata la costruzione dei volumi viene instanziato e registrato il *SensitiveDetector* ed in ogni Hit si memorizza l'id della strip, il piano del telescopio e l'energia depositata, e il tipo di particella (primaria o secondaria). Ogni Hit viene poi salvata nella *HitsCollection* dell'evento in corso per poter essere riutilizzata successivamente.
In Geant4 è possibile definire dei *Digitizer* (che derivano da *G4VDigitizer Module*) per elaborare le *HitsCollection* di ogni evento e produrre delle *Digi Collection*. Le Hit rappresentano le quantità fisiche della risposta del detector, i Digi rappresentano l'informazione elaborata. In questo modo è possibile simulare ad esempio il rumore elettronico e la conversione energia/carica.

Task2a

- ***In generale fate attenzione ai file con #include***

In *CrosstalkGenerator.hh* vengono inclusi I file *"CLHEP/Matrix/SymMatrix.h"* e *"CLHEP/Matrix/Vector.h"*. Per poter compilare correttamente l'esercizio è necessario aver eseguito il build di Geant4 selezionando l'opzione di compilazione "*use_system_CLHEP*", con software CLHEP di versione pari o superiore alla *2.3.3.0*.

- ***PrimaryGeneratorAction -> Svolgere esercizio specifico***

Nel costruttore *PrimaryGeneratorAction::PrimaryGeneratorAction()*, si deve solamente implementare il nome della particella e l'energia della stessa:

```
PrimaryGeneratorAction::PrimaryGeneratorAction()
  : outfile(0)
{
  gun  = new G4ParticleGun(1);

  // complete particle name and energy  (do not forget the energy unit)
  G4ParticleDefinition* pion = G4ParticleTable::GetParticleTable()->FindParticle("pi+");
  gun->SetParticleDefinition(pion);

  gun->SetParticleEnergy(1.0*GeV);
```

In *PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)* viene invece richiesto di generare prima una singola particella:

```
  G4double x0 = 0.*cm, y0 = 0.*cm, z0= 0.0*cm;
  G4cout<<"GeneratePrimaries : new event "<<G4BestUnit(G4ThreeVector(x0,y0,z0),"Length")<<G4endl;

  gun->SetParticlePosition(G4ThreeVector(x0,y0,z0));
  gun->SetParticleMomentumDirection(G4ThreeVector(0.,0.,1.));
  gun->GeneratePrimaryVertex(anEvent);
```

viene chiesto poi di generare una sorgente rettangolare di dimensioni *0.1 x 2* cm. Per farlo si fa uso di numeri casuali uniformemente, richiamati tramite la classe *G4UniformRand:*

```
G4double z0 = 0.*cm, x0 = 0.*cm, y0 = 0.*cm;
x0 = -0.05 + 2*0.05*G4UniformRand();
y0 = -1.0+ 2*G4UniformRand();
gun->SetParticlePosition(G4ThreeVector(x0,y0,z0));
gun->SetParticleMomentumDirection(G4ThreeVector(0.,0.,1.));
gun->GeneratePrimaryVertex(anEvent);
```

- ***Physics List: Includere Pioni positivi e negativi***

Per includere pioni positivi e negativi nella Physics list, analogamente a quanto fatto nel precedente esercizio, basta aggiungere la loro definizione in void *PhysicsList::ConstructParticle() :*

```
void PhysicsList :: ConstructParticle ()
{
 G4PionPlus::PionPlusDefinition();
 G4PionMinus::PionMinusDefinition();
}
```

Task2b

- ***PrimaryGeneratorAction → Utilizzare comandi UI***

Il task 2b chiede di utilizzare la classe *G4GeneralParticleSource*, invece che la classe *G4ParticleGun*, per la generazione di eventi primari:

```
G4GeneralParticleSource *gps = new G4GeneralParticleSource();

gps->GetCurrentSource()->GetEneDist()->SetMonoEnergy(2.0*GeV);
gps->GetCurrentSource()->GetPosDist()->SetCentreCoords(G4ThreeVector(0.0*cm, 0.0*cm, 0.0*cm));
gps->GetCurrentSource()->GetAngDist()->SetParticleMomentumDirection(G4ThreeVector(0.,0.,1.));
```

Viene creata una sorgente primaria di eventi monocromatici a 2 GeV, centrata in (*0cm*, *0cm*, *0cm*) e con momento diretto nella direzione dell'asse Z.

Per modificare I parametri dell'evento primario tramite interfaccia grafica (creata tramite la classe *G4UIManager*) si possono utilizzare i comandi:

- `gps/energy` per modificarne l'energia.
- `gps/direction` per modificarne la direzione.
- `gps/position` per modificarne la posizione.

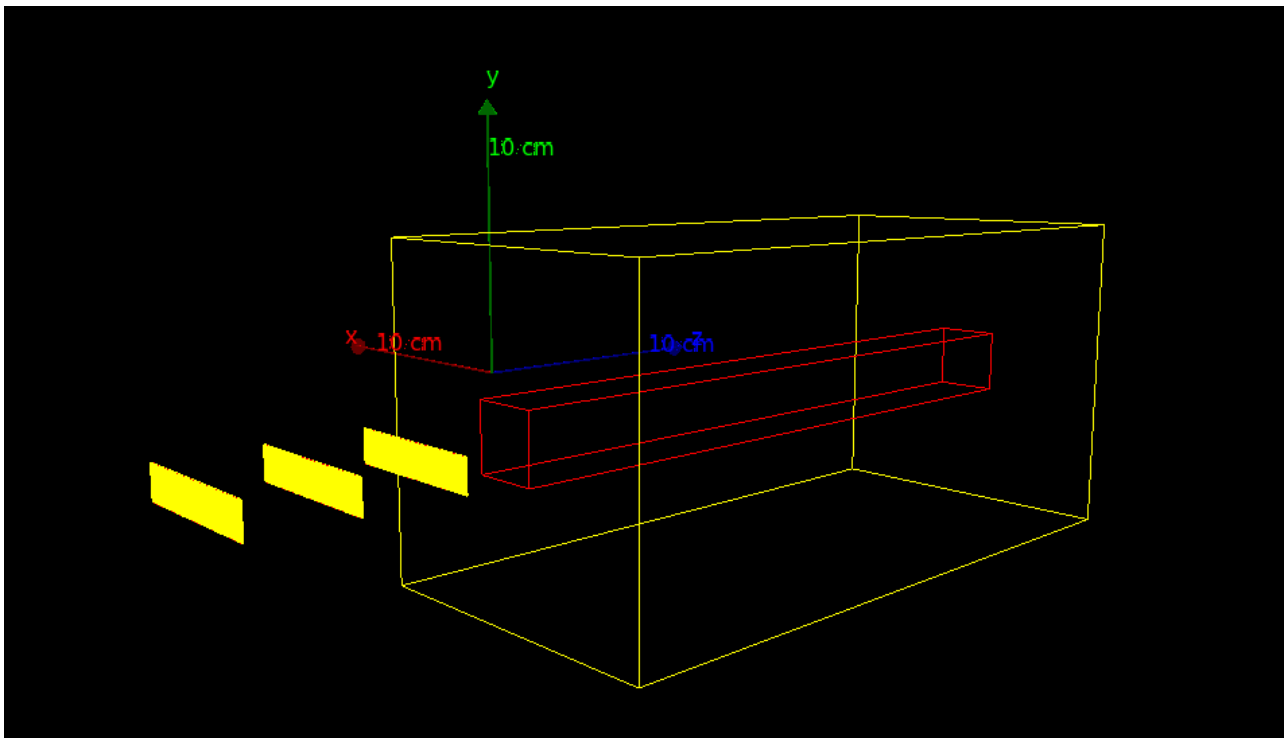
Esercizio 3 – Task 3

Lo scopo di questo esercizio è quello di ustilizzare ulteriori **UserActions** per registrare le informazioni ottenute durante la simulazione. Non si utilizzeranno più le *Hit* e i *SensitiveDetector,* ma il meccanismo della *SteppingAction.* Inoltre per la creazione delle tracce si farà uso della classe *StackingAction.*

Detector → Che cosa sono?

Il detector si compone di più sottoparti:

- **3 tracker di Silicio** (in giallo in figura *3.a)* composti ognuno da 600 strip, costruite con il metodo *Replica* come nell'esercizio 1.
- **1 calorimetro elettromagnetico** in tungstato di piombo composto da due sotto-geometrie di diverse dimensioni (i parallelepipedi in rosso e giallo in figura *3.a*).



Task3a

- ***Main: Capire dove vengono chiamate le nuove user action classes***

Il Main di una simulazione Geant4 ha una struttura predefinita necessaria per caricare i diversi componenti del framework e inizializzare la simulazione secondo le classi definite dall'utente.
Prima di tutto è necessario instanziare un **RunManager**, l'oggetto che si occupa di gestire tutte le fasi della simulazione:

```
G4RunManager * runManager = new G4RunManager();
```

Vengono poi instanziate le **classi** obbligatorie create dall'utente, rispettivamente per la **creazione del detector,** della **physics list** e dell'**evento primario**:

```
G4VUserDetectorConstruction* detector = new
DetectorConstruction();
runManager->SetUserInitialization(detector);

G4VUserPhysicsList* physics = new PhysicsList();
runManager→SetUserInitialization(physics);

G4VUser PrimaryGeneratorAction*gen_action = new
PrimaryGeneratorAction() ;
runManager -> SetUserAction(gen_action) ;
```

Si istanziano poi le classi

- **Stacking Action,** per permettere all'utente di gestire la priorità nella simulazione delle tracce primarie e secondarie:

```
StackingAction* aStackingAction = new
StackingAction();
runManager->SetUserAction(aStackingAction);
```

- **Stepping Action,** per permettere all'utente di registrare le informazioni necessarie delle particelle:

```
SteppingAction* aSteppingAction = new
SteppingAction();
runManager->SetUserAction(aSteppingAction);
```

- **Event Action**, per eseguire del codice prima e dopo ogni evento:

```
EventAction* anEventAction = new
EventAction();
runManager->SetUserAction( anEventAction );
```

- **runAction**, per eseguire del codice prima e dopo ogni run**:**

```
RunAction* aRunAction = new RunAction();
runManager->SetUserAction( aRunAction );
```

- ***Descrivere RunAction, EventAction, SteppingAction, StackingAction e Analisys***

RunAction

In *void RunAction::BeginOfRunAction(const G4Run* aRun)* viene instanziata la run *"aRun"* tramite la funzione *Analysis::GetInstance() → PrepareNewRun(aRun).*

In *void RunAction::EndOfRunAction(const G4Run* aRun)* si conclude la run richiamando la funzione *EndOfRun(aRun).*

EventAction

In *void EventAction::BeginOfEventAction(const G4Event* anEvent)* viene instanziato l'evento "*anEvent"* tramite la funzione *Analysis::GetInstance() → PrepareNewEvent(anEvent).*

In *void EventAction::EndOfEventAction(const G4Event* anEvent)* si conclude l'evento richiamando la funzione *EndOfEvent(anEvent).*

SteppingAction

L'utente può fornire una funzione UserSteppingAction(const G4Step * theStep) che viene chiamata ad ogni step per registrare le informazioni necessarie.

In questo caso si controlla il deposito di energia per assicurarsi che lo step sia avvenuto dentro il calorimetro elettromagnetico.

Si richiede il *copy number* (il calorimetro elettromagnetico ha *copy number* 10 o 11). Si trova quindi la posizione z calcolando un numero casuale tra *pre* e *post step point*, in tal modo si rende più continuo il profilo dell'istogramma.

Infine, tramite la funzione *Analysis::GetInstance()->AddEDepEM(edep, z, volCopyNum)* si salva la deposizione di energia per l'istogramma.

StackingAction

Geant4 permette all'utente di gestire la priorità nella simulazione delle tracce primarie e secondarie attraverso la UserStackingAction .

Tramite la funzione *ClassifyNewTrack(const G4Track * aTrack)* tutti gli stack si classificano come *urgent.* Inoltre se la traccia è secondaria, tramite la funzione *Analysis::GetInstance()→AddSecondary(..),* questa viene aggiunta all'insieme delle tracce secondarie in *Analysis.* Se invece la traccia è primaria, tramite la funzione *Analysis::GetInstance()→SetBeam(..),* si ricava l'energia cinetica della traccia.

Analisys

La classe Analysis contiene diversi metodi per ricevere informazioni dalle varie UserAction implementate dall'utente.

La funzione *void Analysis::PrepareNewEvent* inizializza le variabili relative all'evento corrente e *void Analysis::PrepareNewRun* inizializza quelle relative alla run corrente. *void Analysis::PrepareNewRun* crea inoltre il ROOT file e 3 istogrammi. I tre istogrammi vengono riempiti rispettivamente con l'energia totale depositata normalizzata rispetto all'energia del fascio, l'energia depositata nel calorimetro centrale normalizzata rispetto all'energia del fascio, il profilo energetico lungo il calorimetro (nella direzione *z*).

La funzione *void Analysis::EndOfEvent* accumula gli eventi durante la run e riempie con queste informazioni gli istogrammi (energia totale e energia nel calorimetro centrale).

Nella funzione *void Analysis::EndOfRun,* dopo aver fatto alcuni print out informativi sulla run (ad es. Quali particelle compongono il fascio, il numero di γ ecc..) , si scrive file ROOT e infine si chiude.

La funzione *void Analysis::AddSecondary* conta quanti γ, elettroni e positroni secondari sono stati prodotti nella run.

La funzione *void Analysis::AddEDepEM* riempie l'istogramma del profilo energetico lungo il calorimetro (nella direzione z).

La funzione *void Analysis::SetBeam* setta il tipo di particelle primarie e l'energia del fascio.

- ***Cosa si intende per touchable?***

Un touchable è un'entità geometrica che ha un ruolo univoco nella descrizione del detector. È rappresentato da una classe astratta che può essere implementata

in diversi modi. Ogni modo deve essere tale da permettere di ottenere la trasformazione (*GetTranslation(depth)* e *GetRotation(depth)*) o il solido (*GetSolid(depth), GetVolume(depth), GetReplicaNumber(depth)* e *GetCopyNumber(depth)*) descritto dal touchable.

- ***Che particelle sono abilitate? E che processi?***

In *void PhysicsList::ConstructParticle()* sono abilitate le pseudo-particelle *G4Geantino* e *G4ChargedGeantino* e le particelle che fanno parte della *emPhysicsList* (γ , *e+*, *e-* e qualche adrone carico).
L'unico processo abilitato in *void PhysicsList::ConstructProcess()* è il trasporto (che per la *emPhysicsList* consiste nei processi di produzione di coppie, Compton scattering, effetto fotoelettrico, Rayleighyg scattering, reazioni nucleari per i gamma, ionizzazione, Coulomb scattering, Bremsstrahlung e annichilazione del fotone).

- ***Cosa fa la simulazione se si lancia un run?***

Descritto nella seconda risposta, nella sezione relativa ad *Analisys.*

- ***Salvare dei dati significativi a vostra scelta***

Per salvare dei dati bisogna aggiungere dei print-out nella funzione *void Analysis::EndOfRun* in *analisys*.
Ad esempio, se si vuole conoscere il numero medio di protoni prodotti bisogna inizializzare a 0 la variabile *n_proton* in *void Analysis::PrepareNewRun.* Si deve poi aggiungere un contatore in *void Analysis::AddSecondary :*
if (part == G4Proton::ProtonDefinition()) { ++n_proton; }.
A questo punto si punto si può aggiungere il print-out desiderato in *EnfOfRun:*

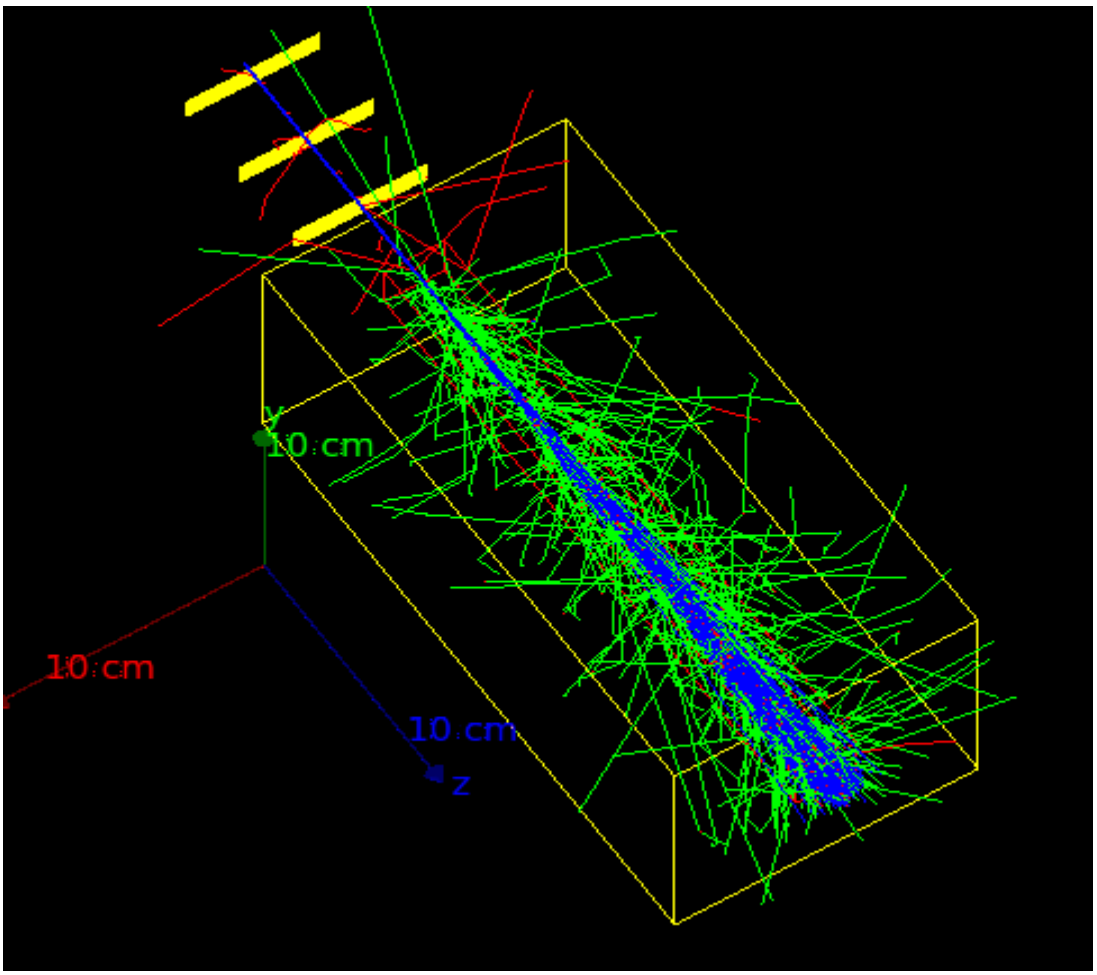
```
G4cout<< "Average number of p:" << (G4double)n_proton
/ (G4double)numEvents << G4endl;
```

Se invece si vogliono conoscere le particelle secondarie bisogna aggiungere in *EnfOfRun:*

```
G4cout<< " Secondaries: "<< thisEventSecondaries <<
G4endl;
```

Plot dei risultati

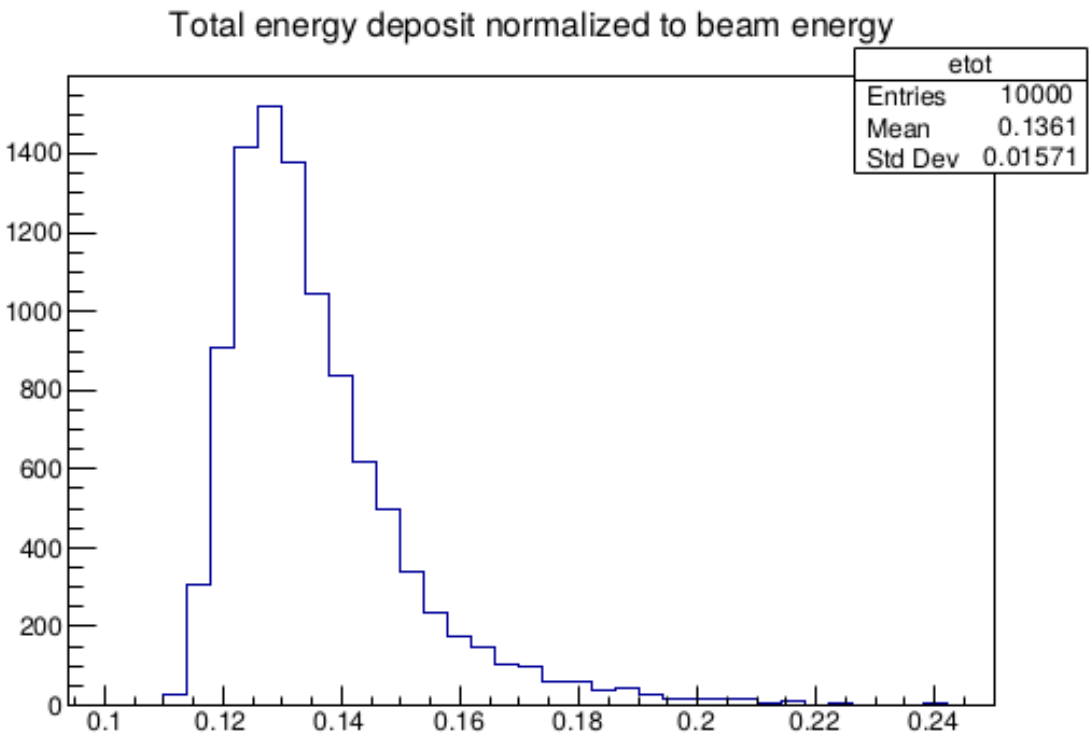
Si esegue una simulazione con 10^4 π+ da 2 GeV. La *PhysicsList* attivata è quella elettromagnetica standard.



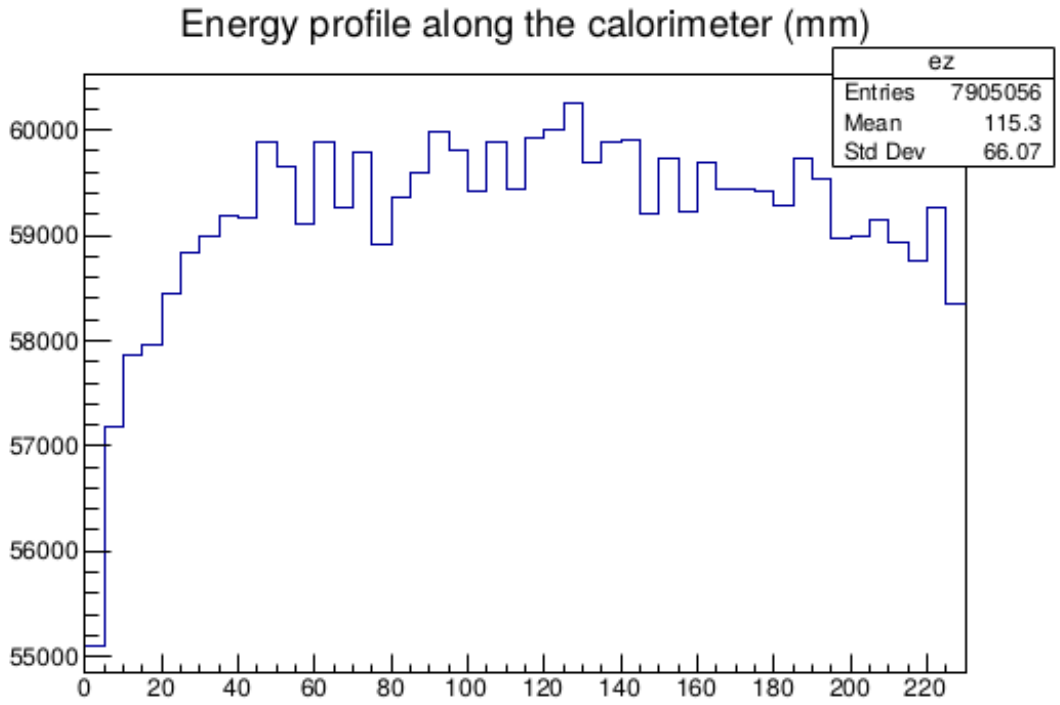
3a *Simulazione di* 10^4 π+ *con energia 2GeV (in verde i gamma e in rosso gli elettroni)*

```
Beam of pi+ kinetic energy: 2 GeV
Event processed:          10000
Average number of gamma: 62.7871
Average number of e-    : 361.667
Average number of e+    : 0.8126
```

Sommario degli eventi generati nella run



3a.a *Energia totale depositata dai π+*

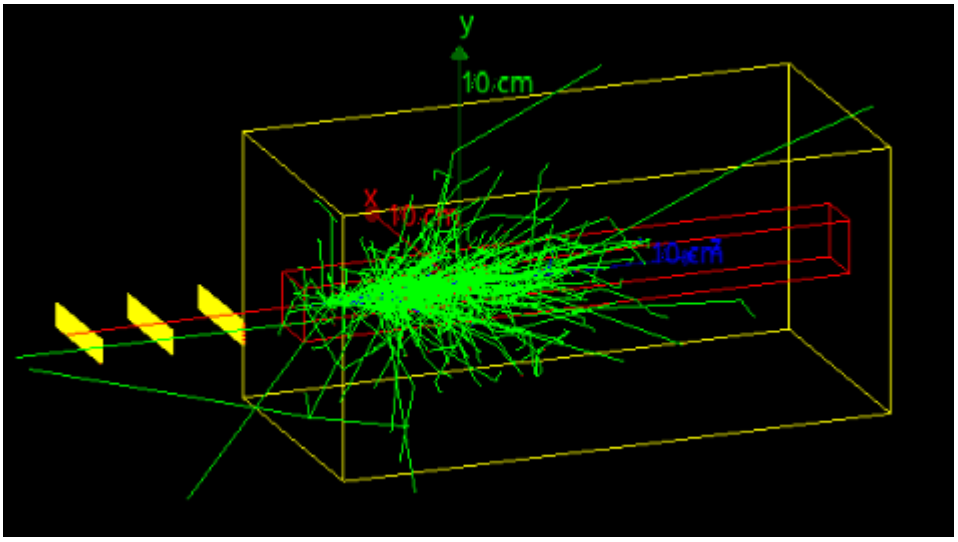


3a.b *Energia dei π+ depositata lungo il calorimetro*

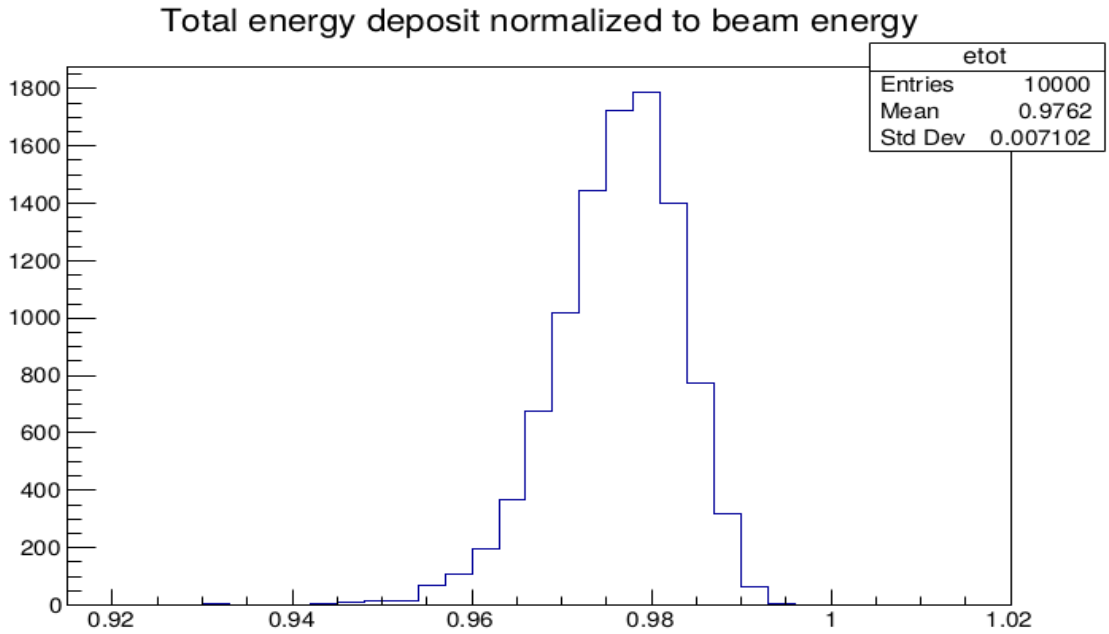
Dall'output della simulazione si nota che i pioni che interagiscono con il calorimetro elettromagnetico producono gamma (principalmente per eccitazione del mezzo e perdita radiativa) ed elettroni secondari (raggi delta).

Dai due grafici si nota che la deposizione media di energia è bassa (figura 3.a) e distribuita lungo l'intero calorimetro (figura 3.b). Dunque la maggior parte dei pioni sopravvive al calorimetro elettromagnetico. Per il completo assorbimento sarebbe necessario quindi un calorimetro adronico.

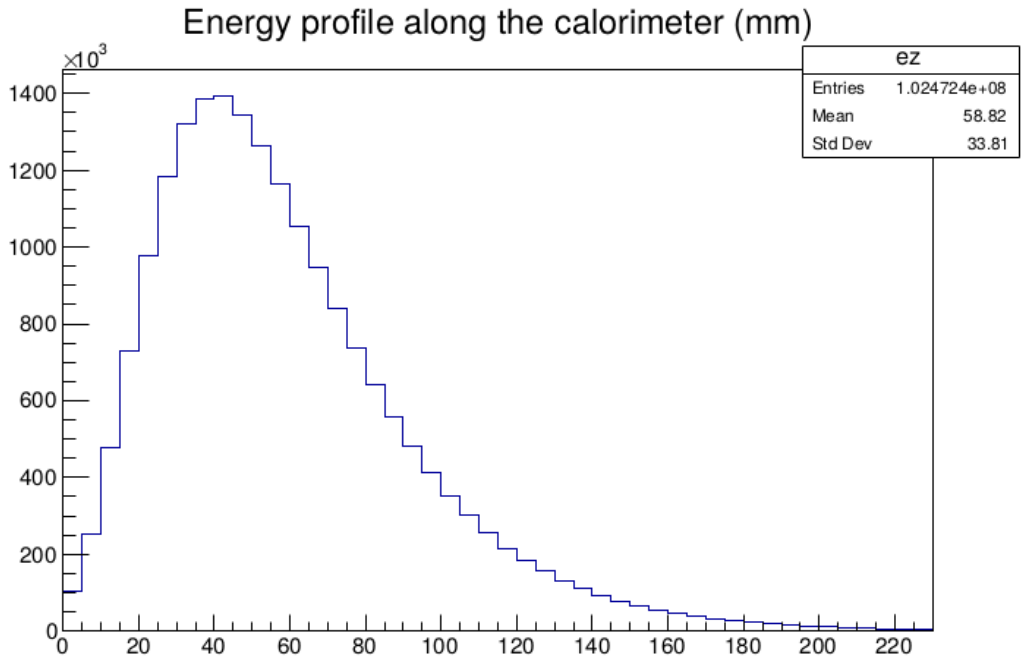
Per mettere a confronto le diverse caratteristiche di deposizione dell'energia per particelle leggere si simulano ora 10^4 **elettron**i.



3a.c *Simulazione di* 10^4 *elettroni con energia 2GeV (in verde i gamma e in rosso gli elettroni)*



3a.d *Energia totale depositata dagli elettroni da 2 GeV*



3a.e*Energia depositata lungo il calorimetro per e− da 2 GeV*

Dall'output della simulazione si apprende che gli elettroni producono una cascata elettromagnetica di gamma, elettroni e positroni (per perdita radiativa e produzione di coppie).

Si nota ora che le particelle primarie e secondarie depositano in media quasi tutta la loro energia lungo il calorimetro elettromagnetico (97.6 % dell'energia iniziale). La deposizione di energia da parte degli elettroni non è uniforme come per i pioni, ma presenta un picco a circa 45 mm di profondità. La perdita di energia avviene principalmente per processi di Coulomb scattering e perdita radiativa.

Task3b

- ***Detector → Cosa c'è di nuovo? A cosa serve?***

Rispetto al task 3a, sono ora presenti due calorimetri adronici, uno in ferro e uno in argon liquido composto da 80 layers, costruiti tramite il metodo Replica. All'interno del calorimetro adronico viene aggiunto un campo magnetico di 3.5 mT. Questo viene fatto per studiare il decadimento del muone e la violazione di parità dell'interazione debole.

In quanto calorimetri adronici, il loro scopo è quello di far depositare l'energia degli adroni al loro interno. Dai grafici della sezione precedente si evince che particelle pesanti di 2 GeV non sono fermate dal calorimetro elettromagnetico.

- ***Physics List: Aggiungere definizione di particelle***

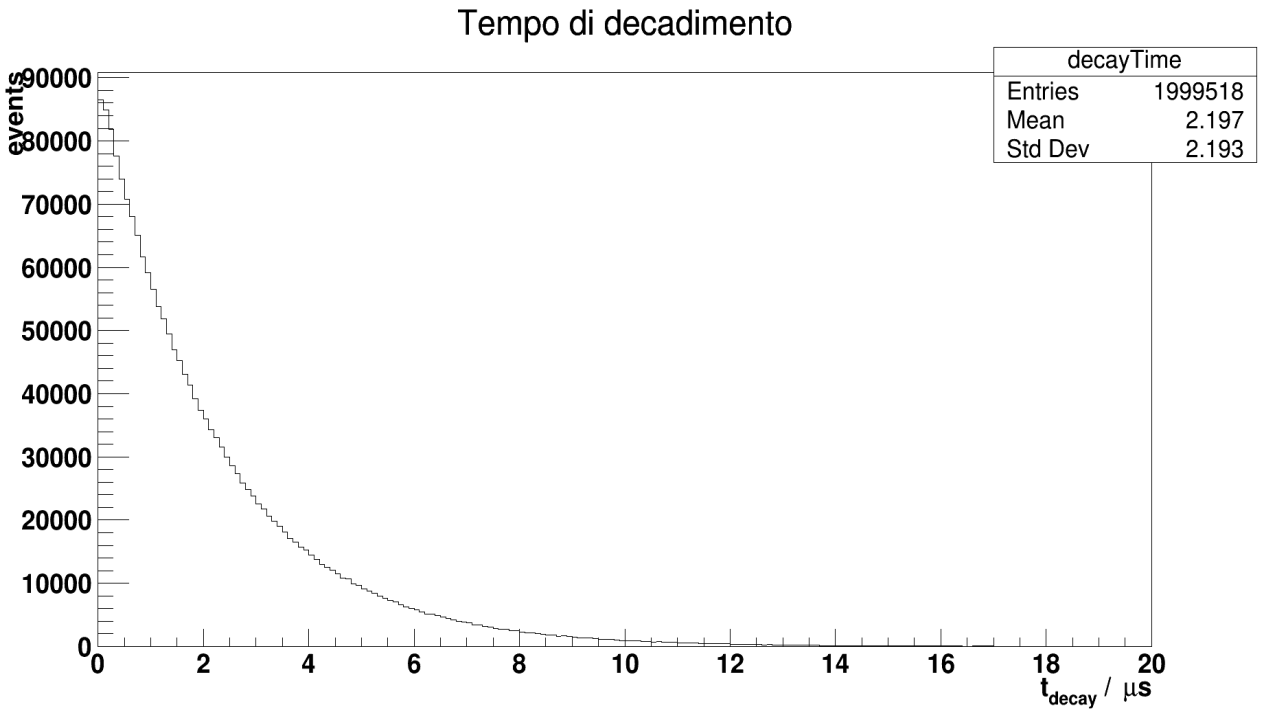
Per aggiungere, come richiesto, la definizione di mu+ e mu-, basta aggiungere nella funzione *void PhysicsList::ConstructParticle():*

```
void PhysicsList :: ConstructParticle ()
{
 G4PionPlus::MuonPlus();
 G4PionMinus::MuonMinus();
}
```

In questo esercizio non si è utilizza la *PhysicsList* elettromagnetica standard ma si sono instanziate particelle e processi manualmente. È importante notare che sono attivati i processi di **trasporto** (che per la *emPhysicsList* consiste nei processi di produzione di coppie, Compton scattering, effetto fotoelettrico, Rayleighyg scattering, reazioni nucleari per i gamma, ionizzazione, Coulomb scattering, Bremsstrahlung e annichilazione del fotone), **multiscattering**, ma soprattutto di **decadimento**, sia a riposo che in volo.

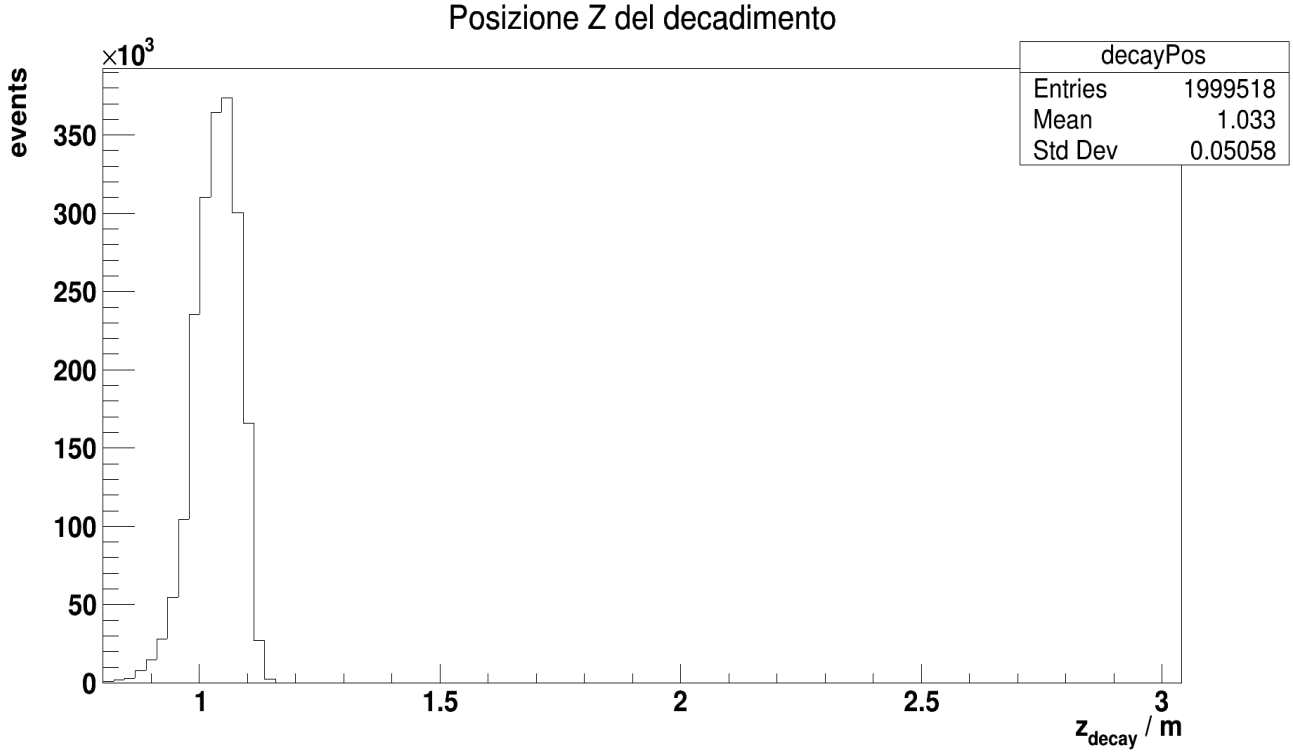
- ***Mostrare qualche istogramma di output***

Si è eseguita la simulazione di circa $2 \cdot 10^6$ μ- da 1 GeV.
Ci si aspetta ovviamente di osservare il decadimento del muone con la conseguente creazione di un elettrone, un antineutrino elettronico e un neutrino muonico.



3b.a Distribuzione del tempo di decadimento dei muoni

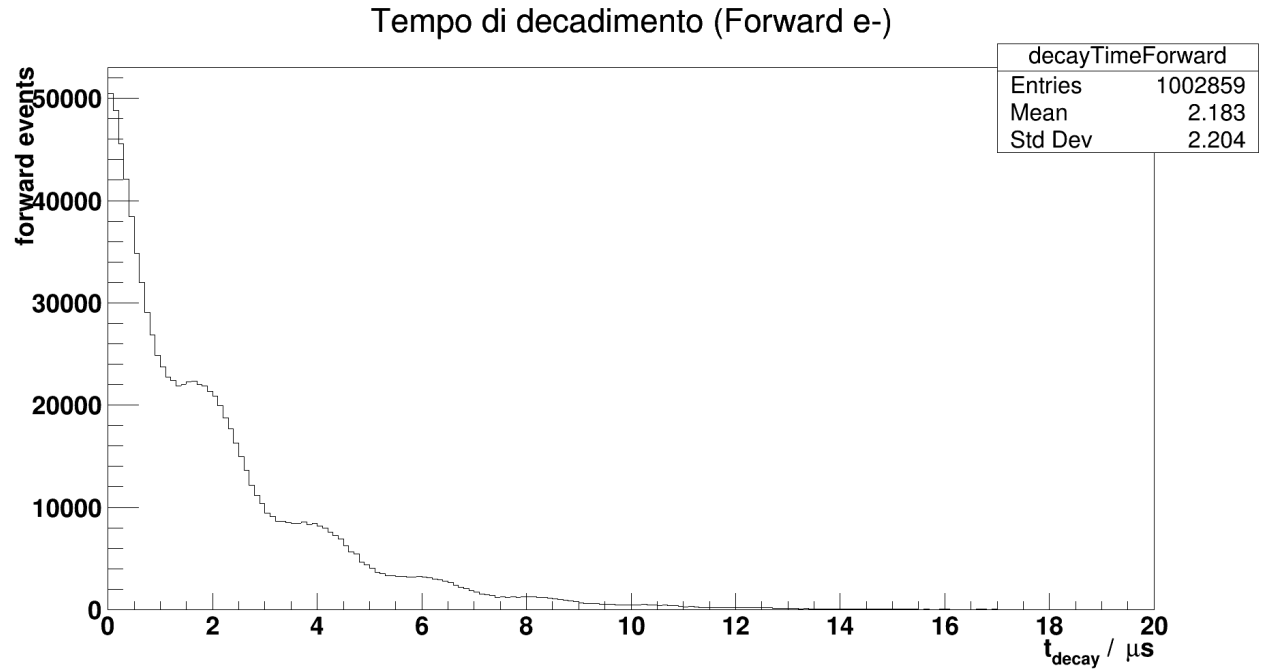
Dal grafico del tempo di decadimento, tramite un fit esponenziale, si ricava la vita media del muone: $\tau = 2.161 \pm 0.006\ \mu s$.



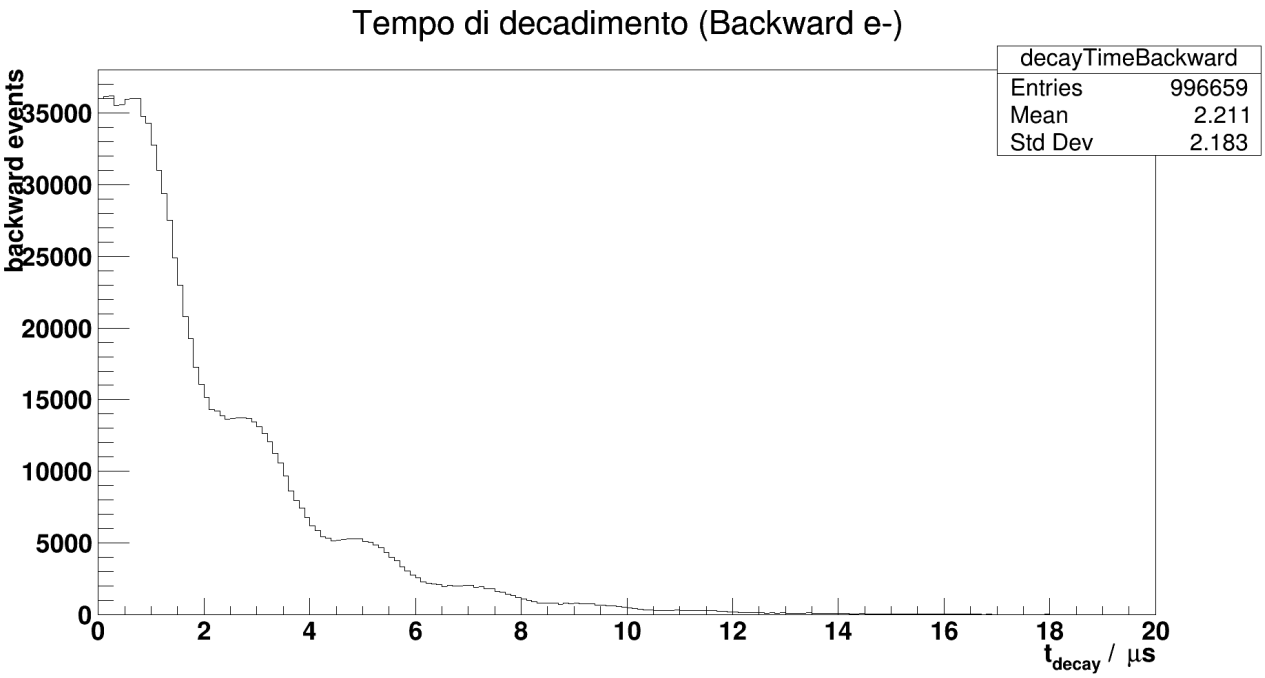
3b.b Posizione del decadimento lungo la direzione Z del rivelatore

Osservando l'evoluzione nel tempo dei muoni con elettrone emesso in avanti o indietro, si nota una periodicità nel profilo di decadimento. Questa corrisponde alla precessione dello spin a causa del campo magnetico: gli elettroni tendono

infatti ad essere emessi nella direzione opposta allo spin del muone (violazione di parità dell'interazione debole).



3b.c *Distribuzione temporale del tempo di decadimento del muone con elettrone emesso in avanti*



3b.d *Distribuzione temporale del tempo di decadimento del muone con elettrone emesso in avanti*

Esercizio 4 - Task 4 File

In questo esercizio si modifica la simulazione precedente aggiungendo un *Sensitive Detector* agli strati di argon liquido del calorimetro elettromagnetico per memorizzare l'energia depositata in ogni strato.

Task4a

- ***Main: Che Physics List viene usata? Descrivetela in breve***

In questo esercizio la PhysicsList non è definita dallo user. Nel main viene attivata la *physics list* predefinita di *Geant4:*

G4VUserPhysicsList physics = new QGSP_BERT().*

"QGSP_BERT()" è una *physics list* di riferimento, in cui sono attivati:

1. tutti i processi elettromagnetici standard
2. cascate ti Bertini fino ai 9.5GeV (che in genere produce più neutroni e protoni secondari del modello LEP, con un accordo maggiore con i dati sperimentali)
3. il modello QGS per le alte energie (E>20GeV)
4. il modello LEP (*Low Energy Parameterized models*) per energie intermedie

- ***DetectorConstruction: completare parte su sensitive detector***

```
//Create the logical value for the LAr layer
G4LogicalVolume* hadLayerLogic = new G4LogicalVolume(hadLayerSolid,lar,"HadLayerLogic",0);

//--------------
// Exercise 1 Task4a
//--------------
//Create a SD
//We need to create a SD and attach it to the active layer of the HAD calorimeter: The LAr logic volume

HadCaloSensitiveDetector* sensitive = new HadCaloSensitiveDetector("/HadClo");
G4SDManager* sdman = G4SDManager::GetSDMpointer();
sdman->AddNewDetector( sensitive );
hadLayerLogic->SetSensitiveDetector(sensitive)
```

Per creare un Sensitive Detector ed appenderlo allo staot attivo del calorimetro adronico composto da argon liquido bisogna innanzitutto creare un oggetto del tipo *HadCaloSensitiveDetector.*
Si punta quindi a questo oggetto tramite il G4*SDManager* :

```
G4SDManager *sdman = G4SDManager::GetSDMpointer();
sdman->AddNewDetector( sensitive );
```

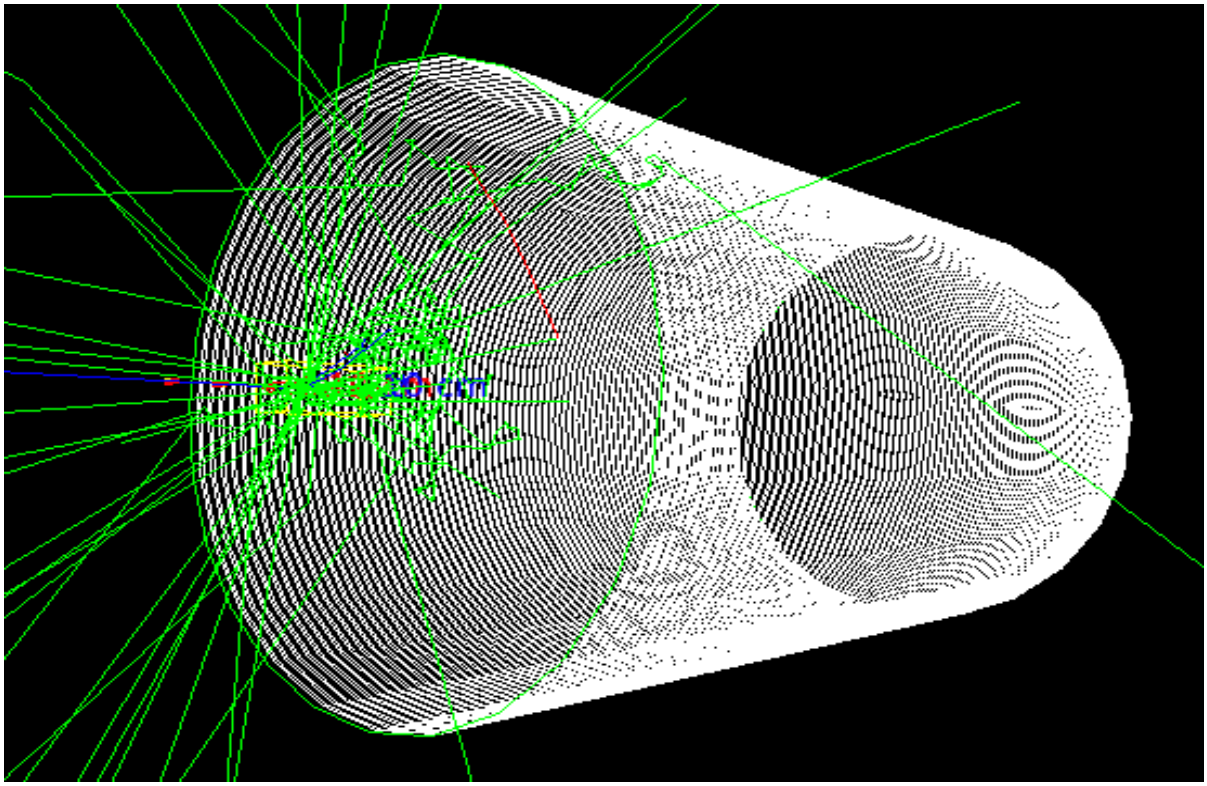
Infine si appende il SD al volume logico del layer attivo:

```
hadLayerLogic→SetSensitiveDetector(sensitive);
```

- ***Obiettivo: lanciare simulazione e salvare output a schermo***

Viene ora lanciata una simulazione di un π+ con energia pari a 2 GeV.

Si riporta l'**output Grafico** della simulazione:



4.a *Simulazione di 1 π+ con energia pari a 2 GeV (in verde i gamma e in rosso gli elettroni)*

Il print-out della simulazione fornisce informazioni sui diversi processi che delle particelle coinvolte.

Qui sotto sono riportati i diversi processi definiti per i **protoni** a diverse energie per quanto riguarda i fenomeni di **ionizzazione**, **perdita radiativa** e **produzione di coppie:**

```
hIoni:   for  proton    SubType= 2
      dE/dx and range tables from 100 eV  to 100 TeV in 84 bins
      Lambda tables from threshold to 100 TeV, 7 bins per decade, spline: 1
      finalRange(mm)= 0.1, dRoverRange= 0.2, integral: 1, fluct: 1,
linLossLimit= 0.01
      ===== EM models for the G4Region  DefaultRegionForTheWorld ======
              Bragg :  Emin=        0 eV    Emax=        2 MeV
         BetheBloch :  Emin=        2 MeV   Emax=      100 TeV

hBrems:   for  proton    SubType= 3
      dE/dx and range tables from 100 eV  to 100 TeV in 84 bins
      Lambda tables from threshold to 100 TeV, 7 bins per decade, spline: 1
      ===== EM models for the G4Region  DefaultRegionForTheWorld ======
              hBrem :  Emin=        0 eV    Emax=      100 TeV

hPairProd:   for  proton    SubType= 4
      dE/dx and range tables from 100 eV  to 100 TeV in 84 bins
      Lambda tables from threshold to 100 TeV, 7 bins per decade, spline: 1
      Sampling table 17x1001; from 7.50618 GeV to 100 TeV
      ===== EM models for the G4Region  DefaultRegionForTheWorld ======
           hPairProd :  Emin=        0 eV    Emax=      100 TeV
```

Si mostrano poi dati riguardanti i processi adronici per il **neutrone**: s**cattering elastico**, **scattering inelastico** e **cattura**:

```
  ============================================================
                   HADRONIC PROCESSES SUMMARY (verbose level 1)
  ------------------------------------------------------------
                             Hadronic Processes for neutron

    Process: hadElastic
          Model:              hElasticCHIPS: 0 eV  ---> 100 TeV
  Cr_sctns:         G4NeutronElasticXS: 0 eV  ---> 100 TeV
       Cr_sctns:             GheishaElastic: 0 eV  ---> 100 TeV

    Process: neutronInelastic
          Model:                        QGSP: 12 GeV ---> 100 TeV
          Model:                        FTFP: 9.5 GeV ---> 25 GeV
          Model:             BertiniCascade: 0 eV  ---> 9.9 GeV
  Cr_sctns:       G4NeutronInelasticXS: 0 eV  ---> 100 TeV
       Cr_sctns:        Barashenkov-Glauber: 0 eV  ---> 100 TeV
       Cr_sctns:        Barashenkov-Glauber: 0 eV  ---> 100 TeV
       Cr_sctns:           GheishaInelastic: 0 eV  ---> 100 TeV

    Process: nCapture
          Model:                nRadCapture: 0 eV  ---> 100 TeV
  Cr_sctns:         G4NeutronCaptureXS: 0 eV  ---> 100 TeV
       Cr_sctns:           GheishaCaptureXS: 0 eV  ---> 100 TeV
```

Per ogni regione del detector vengono poi printati i cut in energia e range:

```
Index : 0     used in the geometry : Yes
Material : G4_AIR
Range cuts        :  gamma  700 um     e-  700 um     e+  700 um
proton 700 um
Energy thresholds :  gamma  990 eV     e-  990 eV     e+  990 eV
proton 70 keV
Region(s) which use this couple :
DefaultRegionForTheWorld
Index : 1     used in the geometry : Yes
Material : G4_Si
Range cuts        :  gamma  700 um     e-  700 um     e+  700 um
proton 700 um
Energy thresholds :  gamma  5.87535 keV    e-  424.726 keV    e+
410.692 keV proton 70 keV
Region(s) which use this couple :
DefaultRegionForTheWorld
```

Infine, nel sommario della run sono riportati gli eventi secondari e l'energia media depositata nei calorimetri adroni e elettromagnetici:

```
================
Summary for run: 0
Event processed: 1
Average number of secondaries: 878
Average energy in EM calo: 1.06155 GeV
Average energy in Had calo: 11.2515 MeV
================
```

Si nota che la deposizione di energia maggiore avviene nel calorimetro elettromagnetico.

Task4b

- ***Svolgere esercizio su StackingAction***

La prima parte dell'esercizio richiede di aumentare il contatore dei gamma e dei neutroni solo se la loro energia è maggiore di quella di soglia (*30 MeV*).
Per farlo basta controllare che la particella sia effettivamente un γ o un neutrone, controllare la sua energia cinetica e quindi aumentare il contatore se l'energia è sopra soglia :

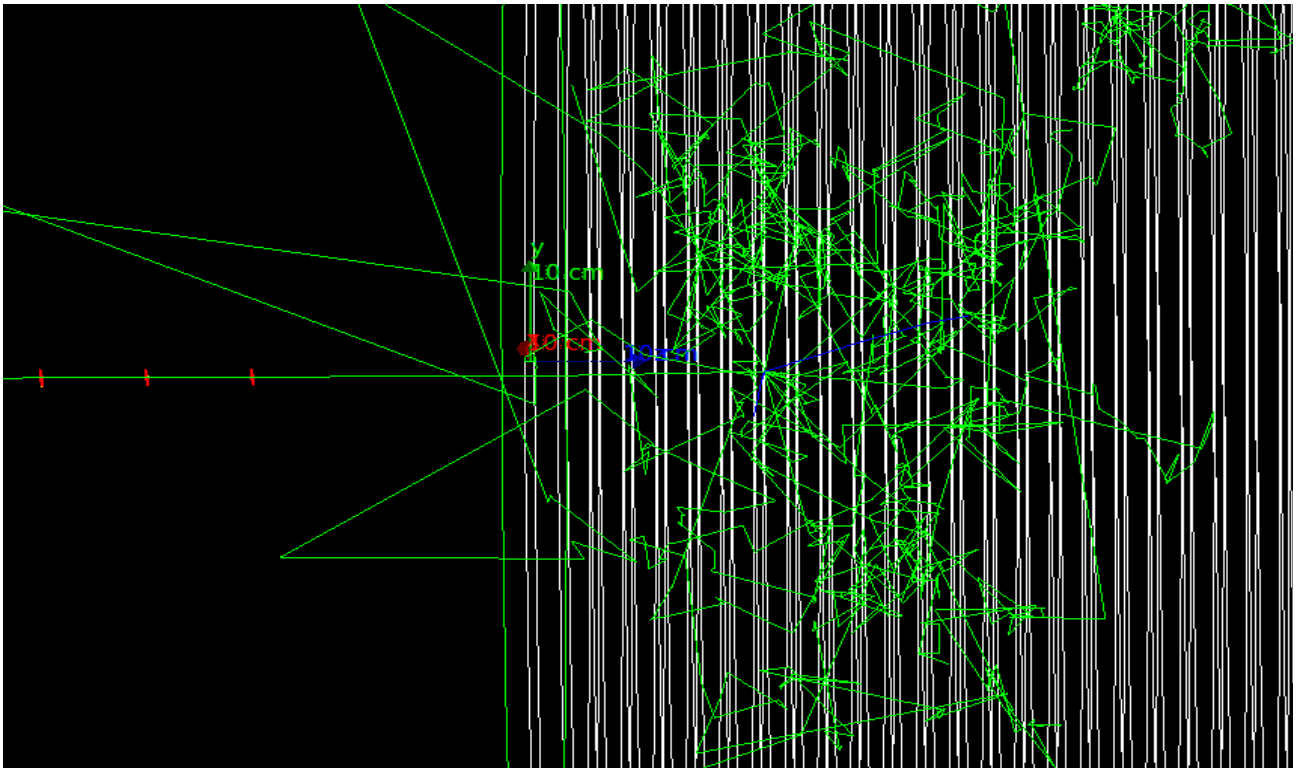
```
if ( particleType == G4Gamma::GammaDefinition() ){
G4double ken_gam = aTrack->GetKineticEnergy();
if ( ken_gam > thresh ){
analysis->AddGammas(1);
  }
}

if ( particleType == G4Neutron::NeutronDefinition() ){
G4double ken_neut = aTrack->GetKineticEnergy();
if ( ken_neut > thresh ){
analysis->AddNeutrons(1);
  }
}
```

La seconda parte dell'esercizio richiede di controllare che la particella sia un γ e in tal caso sopprimerla. Per farlo basta impostare lo stato *fKill* tramite *result=fKill* una volta controllato che la particella sia effettivamente un γ.

- ***Obiettivo: lanciare simulazione e salvare output a schermo***

Viene ora lanciata una simulazione di un **neutrone** con energia pari a 1GeV.

Si riporta l'**output Grafico** della simulazione:



4.b Simulazione *di un neutrone di energia 1 GeV (in verde i neutroni scatterati e in blu i gamma)*

Per testare la corretta implementazione del conteggio dei neutroni e dei gamma si è lanciata una simulazione di 10^3 **neutroni** primari da 50 GeV:

```
Run Summary Neutron 50 GeV
Number of events processed : 1000
==================
Event processed: 1000
Average number of secondaries: 7266
Average energy in Had calo: 1.23271 GeV
Average number of gammas: 38
Average number of neutrons: 128
==================
```

Come atteso, nella simulazione sono presenti reazioni nucleari con conseguente produzione di gamma e neutroni.

Task4c

- ***Hit: Aggiungere print-out delle hits***

Per aggiungere il print-out delle hits bisogna aggiungere nella funzione *void HadCaloSensitiveDetector::EndOfEvent* del file *HadCaloSensitiveDetector.cc* la linea di codice:

```
hitCollection→PrintAllHits();
```

Per aggiungere il print-out del layer number e dell'energia depositata bisogna aggiungere nella funzione *void HadCaloHit::Print()* nel file *HadCaloHit.cc* :

```
G4cout << "Energy Deposited in layer " << layerNumber
<< "is " << G4BestUnit(eDep, "Energy") << G4endl;
```

- ***Sensitive detector: includere hits***

Lavorando nella funzione
void HadCaloSensitiveDetector::Initialize(G4HCofThisEvent HCE)*

innanzitutto si crea la *hit collection*, richiamando il nome del Sensitive Detector (tramite la funzione *GetName()*) e della *collection* (tramite il vettore collection alla posizione 0, in quanto unica).

Infine, per aggiungere le hits si deve ottenere l'ID di *collectionName[0]* e quindi appendere le hit alla *Hit Collection* tramite la funzione *AddHitsCollection.*

```
hitCollection = new HadCaloHitCollection( GetName(), collectionName[0] );
static G4int HCID = -1;
if (HCID<0) HCID = GetCollectionID(0);
HCE->AddHitsCollection(HCID, hitCollection);
```

Per testare la corretta implementazione del print-out delle hits si lancia una run di 100 π+ da *2 GeV*:

```
Starting Run: 0
Starting Event: 0
Energy Deposited in layer 0 is 1.40622 MeV
Energy Deposited in layer 1 is 2.07927 MeV
Energy Deposited in layer 2 is 1.39722 MeV
Energy Deposited in layer 3 is 2.20579 MeV
Energy Deposited in layer 4 is 3.71753 MeV
Energy Deposited in layer 5 is 24.4079 MeV
Energy Deposited in layer 6 is 5.50504 MeV
```

4.c *Print-out delle hits nei primi 6 layers*

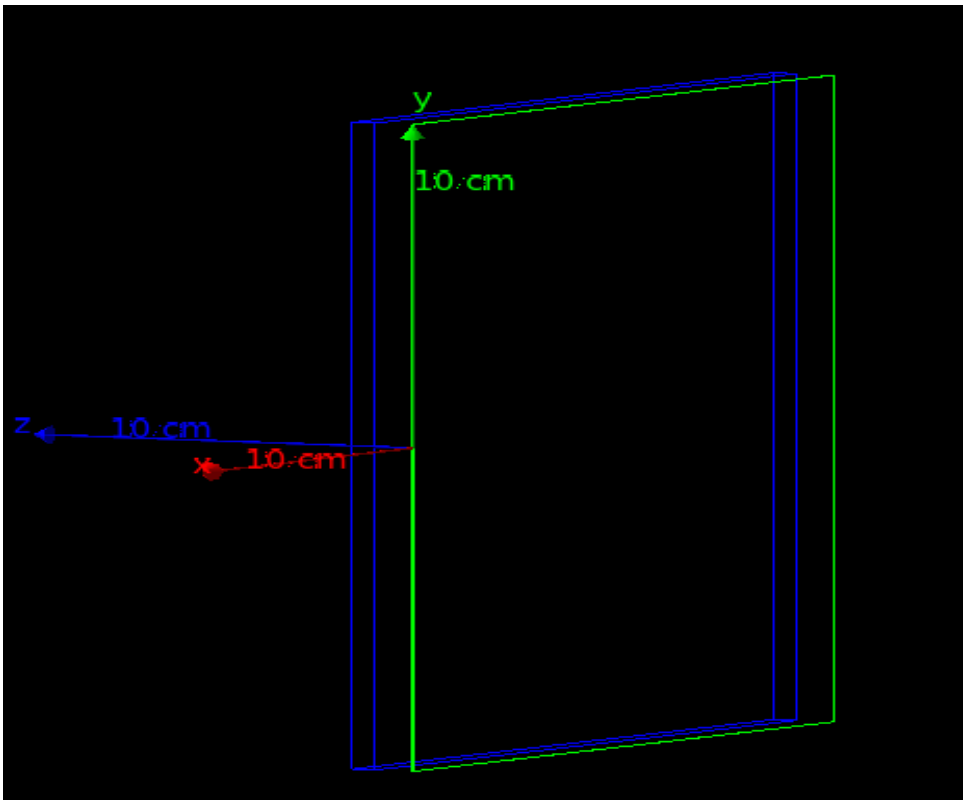
Andando a controllare l'energia depositata su tutti gli 80 strati si nota che i π+ depositano tutta la loro energia cinetica entro il 50esimo strato.

Esercizio 5 – Task 6

Questo esercizio verte su un rivelatore di neutroni veloci costituito da un sottile strato di materiale per la conversione dei neutroni e da un rivelatore a gas ad Argon posto a 1 cm di distanza.

Si utilizza un materiale ricco di idrogeno per sfruttare il fenomeno dello scattering elastico dei neutroni veloci sui protoni: il Polietilene (CH2) è un materiale adatto.

- ***Che detector è? Quale è il mezzo attivo?***



5.a

Il detector è composto da più parti:

- **PE Converter**, un parallelepipedo in Polietilene di misure *20cm * 20cm * 100um* (in verde in figura *4.a*);
- **Argon Detector (rivelatore a gas)**, un parallelepipedo in Argon di misure *20cm * 20cm * 3mm* (in blu in figura *4.a*)

Il mezzo attivo è l'argon Detector. Il mezzo attivo è stato creato appendendo un Sensitive Detector al volume logico dell'Argon detector.

- ***Descrizione dettagliata delle classi***

DetectorConstruction.cc

Nella funzione *void DetectorConstruction::DefineMaterials()* vengono definiti i materiali *NIST.* Si richiamano innanzitutto l'idrogeno ed il carbonio, e si costruisce quindi il polietilene (CH2) attribuendogli una data densità (*0.935 g/cm3*). Si costruisce poi l'Argon, ancora definendo una data densità (*0.001784 g/cm3*).

In *void DetectorConstruction::ComputeParameters()* vengono definiti i valori di default della geometria.

In *G4VPhysicalVolume* DetectorConstruction::Construct()* si costruisce il mondo (composto di materiale *G4_Galactic*) e lo si posizione nel punto *(0,0,0).*

In *G4VPhysicalVolume* DetectorConstruction::Construct_PEConv_ArDet()* si costruiscono il converter in polietilene e il detector di argon precedentemente descritti e li si posizionano rispettivamente in *(0 , 0 , 0)* ed in *(0 , 0 , 1cm + 1mm + 50um).* Si appende infine il SD al volume logico del detector di argon in modo tale da farlo divenire il volume attivo.

La funzione *void DetectorConstruction::UpdateGeometry()* consente la pulizia della vecchia geometria e la costruzione di una nuova.

SensitiveDetector.cc

Nel costruttore si dichiara il nome della *HitCollection* che si utilizzerà in seguito.

Nella funzione *G4bool SensitiveDetector::ProcessHits(G4Step *step, G4TouchableHistory *)* si ricava il *copy number* del sensitive volume. Risalendo alla traccia tramite *step→GetTrack(),* si ricava l'energia cinetica della traccia, il suo ID, la particella che la compone, l'ID della madre, la sua posizione Z, il suo *startime* e infine si l'energia depositata nello step.

Se l'energia depositata è maggiore dell'energia della particella primaria (*2 MeV*) viene restituito errore, altrimenti la hit viene inserita nella *Hit Collection.*

Nella funzione *void SensitiveDetector::Initialize(G4HCofThisEvent* HCE)* viene inizializzato l'*HitColloection.*

Infine in *void SensitiveDetector::EndOfEvent(G4HCofThisEvent*)* si fa il print-out delle hit.

DetectorMessenger.cc

Nel costruttore si possono definire nuovi comandi, ma in questo caso non ne è stato definito nessuno.

PrimaryGeneratorAction.cc

GPS viene scelto come metodo per generare eventi primari.

- La sorgente primaria è composta da neutroni
- la distribuzione energetica è monocromatica a *2 MeV*
- la sorgente è posizionata in *(0cm, 0cm, -5m)*
- la deviazione standard del profilo di posizione del beam è di 0.001 mm, sia per la direzione *x* che per quella *y*
- la direzione del momento è l'asse *z*
- la distribuzione angolare è *"beam2d"*
- la deviazione standard del profilo direzionale del beam è di *0.001 mrad*, sia per la direzione *x* che per quella *y*
- l'asse di riferimento per la distribuzione angolare è l'asse *(-1,0,0)*

RunAction.cc

Nel costruttore si inizializza il *RootSaver* tramite
eventAction → SetRootSaver(..) .

Nella funzione *void RunAction::BeginOfRunAction(const G4Run* aRun)* viene creato un nuovo Ttree per ogni run. Viene inoltre creato un istogramma ROOT con lo scopo di immagazzinare la distribuzione di energia di un dato evento.

Nella funzione *void RunAction::EndOfRunAction(const G4Run* aRun)* si salvano i dati della run nell'istogramma, si chiude il ROOT file e si chiude il Ttree.

EventAction.cc

Nella funzione *void EventAction::AssignHisto(TH1F* histo)* si assegna l' istogramma, creato in *RunAction,* all'evento.

Nella funzione *void EventAction::BeginOfEventAction(const G4Event* anEvent)* si richiama il SD tramite la funzione *EventAction::GetSensitiveDetector(G4String detname)* e si trova l'ID per la *hit collection* del SD tramite *GetCollectionID(..)* .
Nella funzione *void EventAction::EndOfEventAction(const G4Event* anEvent)* si trova la *hits collection* dell'evento e, una volta ottenuta, si fa un loop delle hit e si aggiunge l'energia a un array. In pratica si trasforma la collezione delle hits in un array contenente le energie delle hits. Si riempie quindi l'istogramma dell'energia depositata (*edep/keV*).

StackingAction.cc

Nella funzione *G4ClassificationOfNewTrack StackingAction::ClassifyNewTrack(const G4Track * aTrack)* si definiscono con status *fUrgent* le tracks primarie e con status *fWaiting* le tracks secondarie.

TrackParticle.cc

TrackParticle.cc definisce le funzioni che restituiscono l'energia depositata, il tipo di particella, l'id della particella madre, l'energia depositata totale, il nome della particella, la posizione z della traccia e lo start time.

TrackParentParticle.cc

Nel costruttore di *TrackParentParticle.cc* vengono fatti i print-out delle informazioni iniziali della particella madre.

Nella funzione *void TrackParentParticle::SetIntValues(G4int* IntArray)* e *void TrackParentParticle::SetDoubleValues(G4double* DoubleArray)* le informazioni della particella madre sono attaccate a due array, rispettivamente di valori interi e di valori double. Vengono inoltre fatti dei print-out delle informazioni immagazzinate.

RootSaver.cc

Nella funzione *void RootSaver::CreateTree* si crea un nuovo ROOT file e si apre per la scrittura, se il file esiste già si sovrascrive. Si crea un nuovo Ttree e si definiscono i *Branches (ntracks, id, mum, type ...)* del tree.

Nella funzione *void RootSaver::CloseTree()* si scrive il TTree nel file ROOT, che poi viene chiuso.

In *void RootSaver::AddEvent* si itera su un numero massimo di tracce (*1000*) e si ricavano alcune informazioni per ogni traccia (ID, *energia depositata*, particella madre, tipo di particella..). Con queste informazioni vengono riempiti i branches del rootTree e vengono fatti i print-out dei valori della particella attuale e della particella madre.

- ***Cambiare da materiale del catodo CH2 a Alluminio, Carbonio, Oro e Rame***

- **Da linea di codice**

Innanzitutto, nella funzione *void DetectorConstruction::DefineMaterials()*, si devono definire I materiali inizializzandoli nell'header e nel file .cc, per poi richiamarli dalla lista *G4NIST Material :*

```
private:

  //! \name Materials
  //@{

  G4Material* vacuum;
  G4Material* PE_Mat;
  G4Material* Ar_Mat;
  G4Material* alluminio;
  G4Material* oro;
  G4Material* rame;
  G4Material* carbonio;
```

Definizione nell'header

```
DetectorConstruction::DetectorConstruction()
  : vacuum(0)
  , Ar_Mat(0)
  , PE_Mat(0)
  , alluminio(0)
  , rame (0)
  , oro(0)
  , carbonio(0)
  , logicWorld(0)
  , halfWorldLength(0.5*km)
```

Inizializzazione nel file .cc

```
//esercizio 4-cambiare materiali

//Alluminio
alluminio = man->FindOrBuildMaterial("G4_Al");
//carbonio
carbonio = man->FindOrBuildMaterial("G4_C");
//Oro
oro = man->FindOrBuildMaterial("G4_Au");
//Rame
rame = man->FindOrBuildMaterial("G4_Cu");
```

Materiali richiamati dalla lista G4NIST Material

Una volta definiti i materiali si può procedere ad inserire il materiale desiderato al momento della costruzione del volume logico del catodo.

Alluminio

```
    logicPEConv =
new G4LogicalVolume(solidPEConv,        // its solid
        alluminio,          //its material --> PE Converter
        "logic_PEConv");  //its name
```

Carbonio

```
    logicPEConv =
new G4LogicalVolume(solidPEConv,        // its solid
        carbonio,          //its material --> PE Converter
        "logic_PEConv");  //its name
```

Oro

```
    logicPEConv =
new G4LogicalVolume(solidPEConv,        // its solid
        oro,          //its material --> PE Converter
        "logic_PEConv");  //its name
```

Rame

```
    logicPEConv =
new G4LogicalVolume(solidPEConv,        // its solid
        rame,         //its material --> PE Converter
        "logic_PEConv");  //its name
```

- **Tramite linea di comando**

Si utilizza la classe *DetectorMessenger* per modificare il materiale del catodo attraverso la linea di comando:

```
detDir = new G4UIdirectory ("/detector/catmat" );
detDir->SetGuidance("detector cathod material" );

setCathodeMaterial = new G4UIcmdWithAString ( "/detector/catmat
/setMaterial" , this );
setCathodeMaterial->SetGuidance("Enter the material for the
cathode");
setCathodeMaterial->SetParameterName("material" , true );
setCathodeMaterial->AvailableForStates( G4State_Idle );
```

- ***Lanciare neutroni di diversa energia***
- **Da linea di codice**

  Per cambiare l'energia dei neutroni è necessario cambiare il parametro *eneDist->SetMonoEnergy* in *PrimaryGeneretarAction.cc* nella funzione *G4VPrimaryGenerator* PrimaryGeneratorAction::InitializeGPS() :*

```
// particle type
G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
G4ParticleDefinition* neutron = particleTable->FindParticle("neutron");
gps->GetCurrentSource()->SetParticleDefinition(neutron);

// set energy distribution
G4SPSEneDistribution *eneDist = gps->GetCurrentSource()->GetEneDist() ;
eneDist->SetEnergyDisType("Mono"); // or gauss
eneDist->SetMonoEnergy(2.0*MeV);
```

Neutroni da 2 MeV

```
// particle type
G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
G4ParticleDefinition* neutron = particleTable->FindParticle("neutron");
gps->GetCurrentSource()->SetParticleDefinition(neutron);

// set energy distribution
G4SPSEneDistribution *eneDist = gps->GetCurrentSource()->GetEneDist() ;
eneDist->SetEnergyDisType("Mono"); // or gauss
eneDist->SetMonoEnergy(3.0*MeV);
```

Neutroni da 3 MeV

- **Tramite linea di comando**

cambiare l'energia dei neutroni da linea di comando basta lanciare il comando:

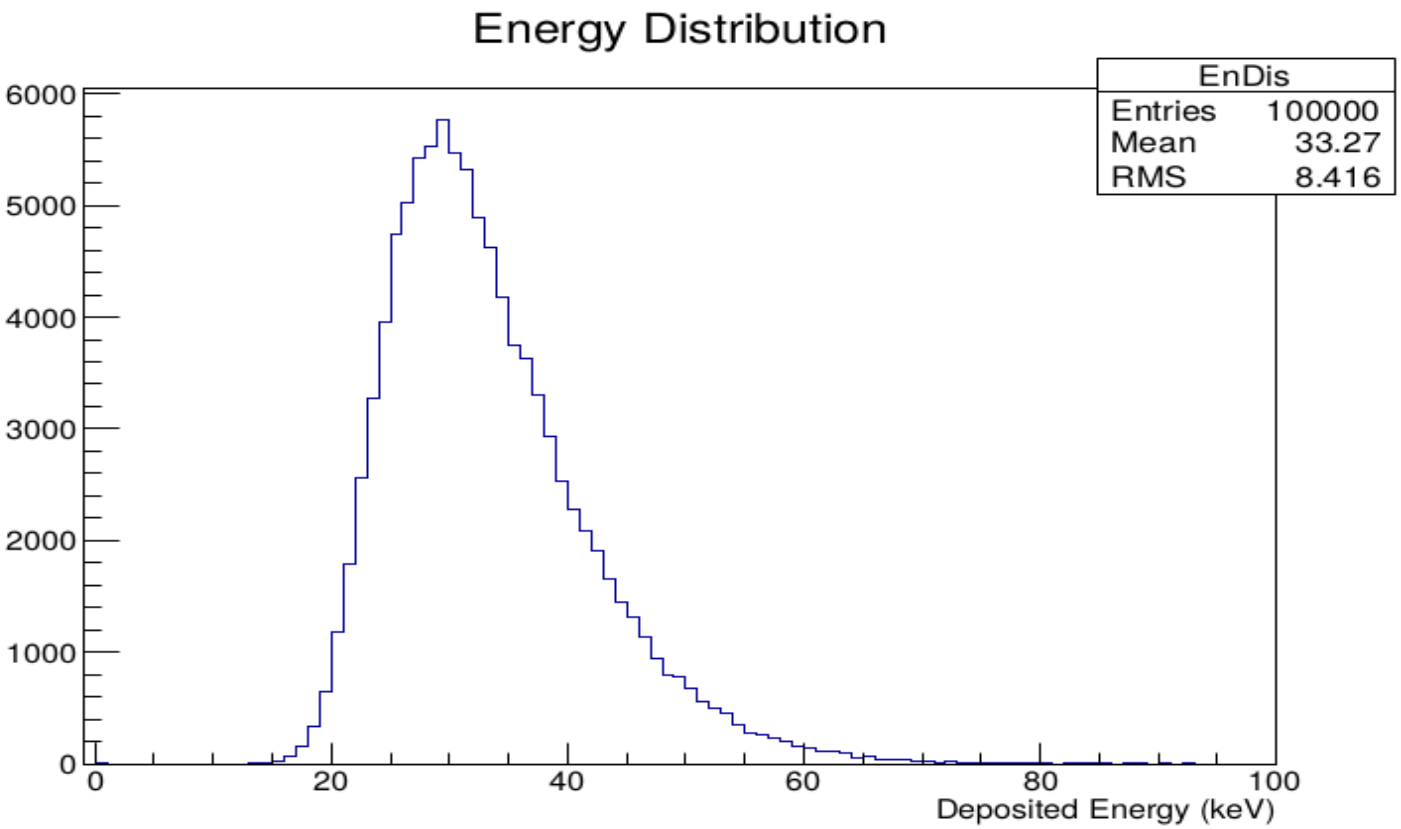
```
/gps /particle neutron
/gps /energy 2 MeV
/gps /energy 3 MeV
```

Energia depositata da diverse particelle nel detector

Si sono lanciate innanzitutto due run da 10^5 eventi ciascuna.

La traccia primaria della prima run è composta da protoni mentre quella della seconda da elettroni.

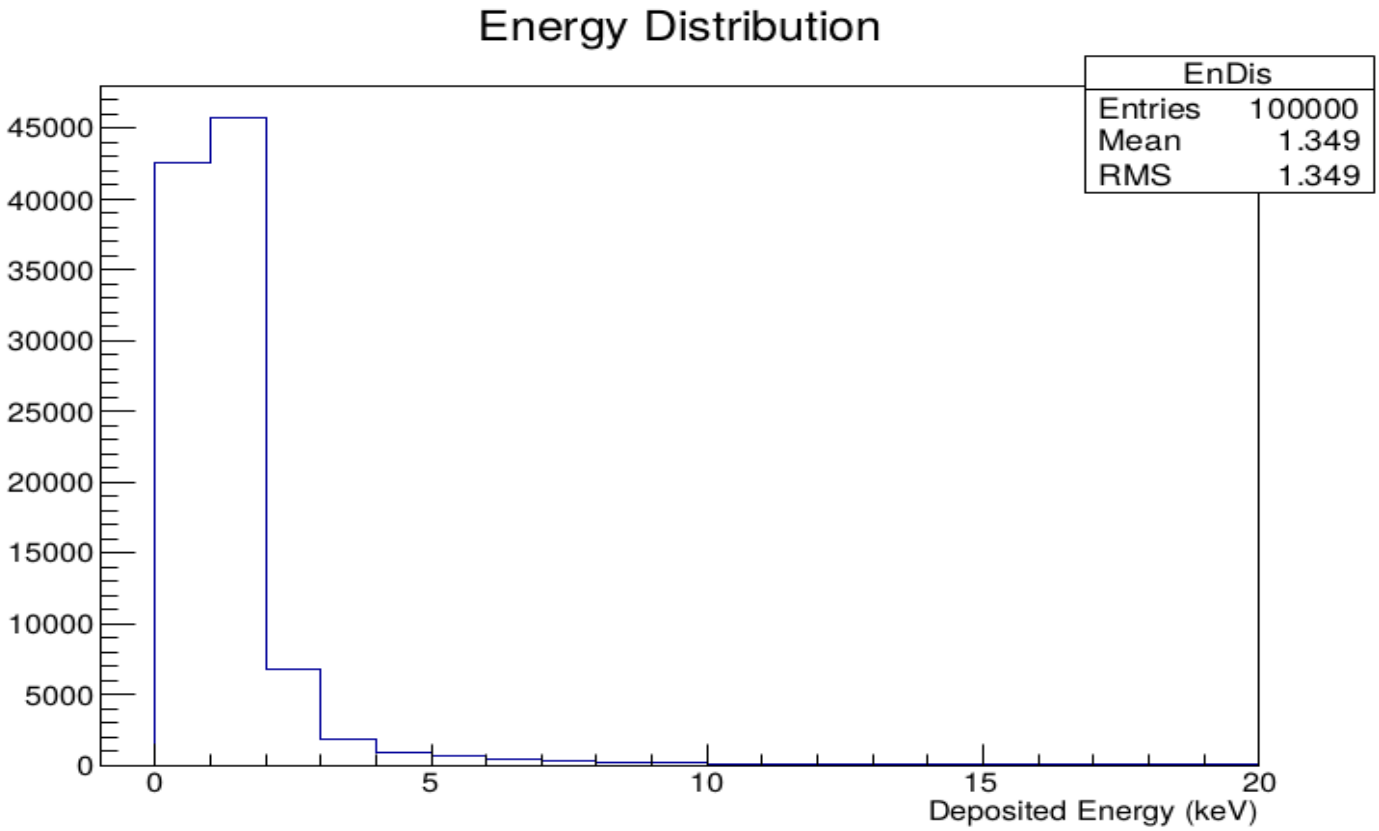
- **10^5 protoni** da **10 MeV :**



5.b

Dall'istogramma si nota immediatamente che l'energia depositata dalle particelle segue la distribuzione di Landau, tipica dell'interazione di particelle pesanti con un assorbitore sottile. La forma della distribuzione dell'energia è dovuta all'emissione di elettroni delta, in seguito all'interazione di particelle pesanti di alta energia con l'assorbitore. Gli elettroni delta sono particelle secondarie con abbastanza energia da viaggiare per una distanza significativa e produrre ulteriore ionizzazione.

- **10^5 elettroni** da **10 MeV :**



5.c

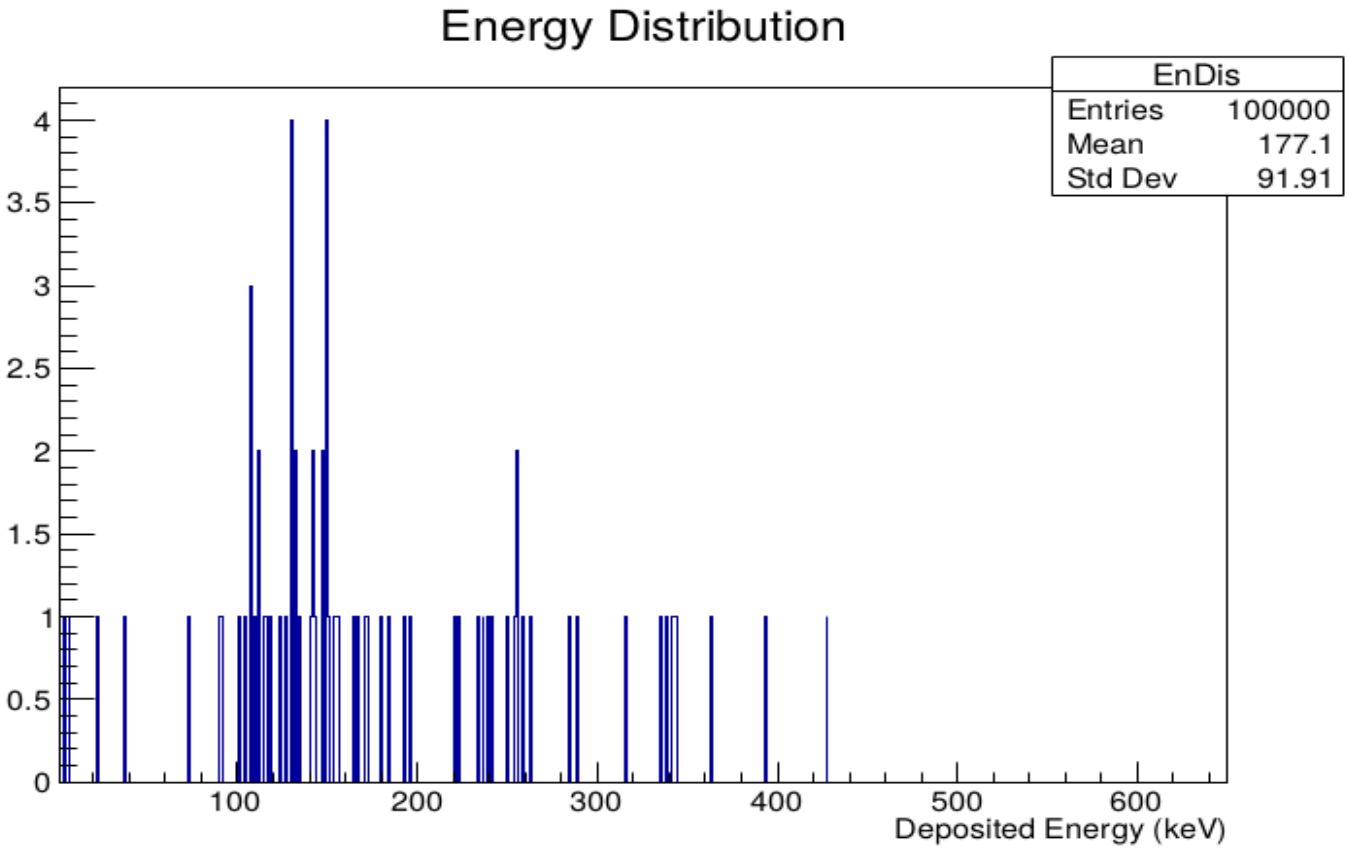
L’energia media depositata dagli elettroni è, come atteso, significativamente minore rispetto a quella depositata dai protoni nello stesso mezzo.

Le particelle secondarie prodotte nella simulazione sono fotoni, elettroni e positroni, infatti, quando un elettrone incide su un assorbitore inizia una cascata elettromagnetica, in quanto gli effetti di perdita radiativa e produzione di coppie producono elettroni e fotoni ad energia minore.

Si eseguono ora diverse run, sempre composte da 10^5 eventi, ma con neutroni come particelle primarie.

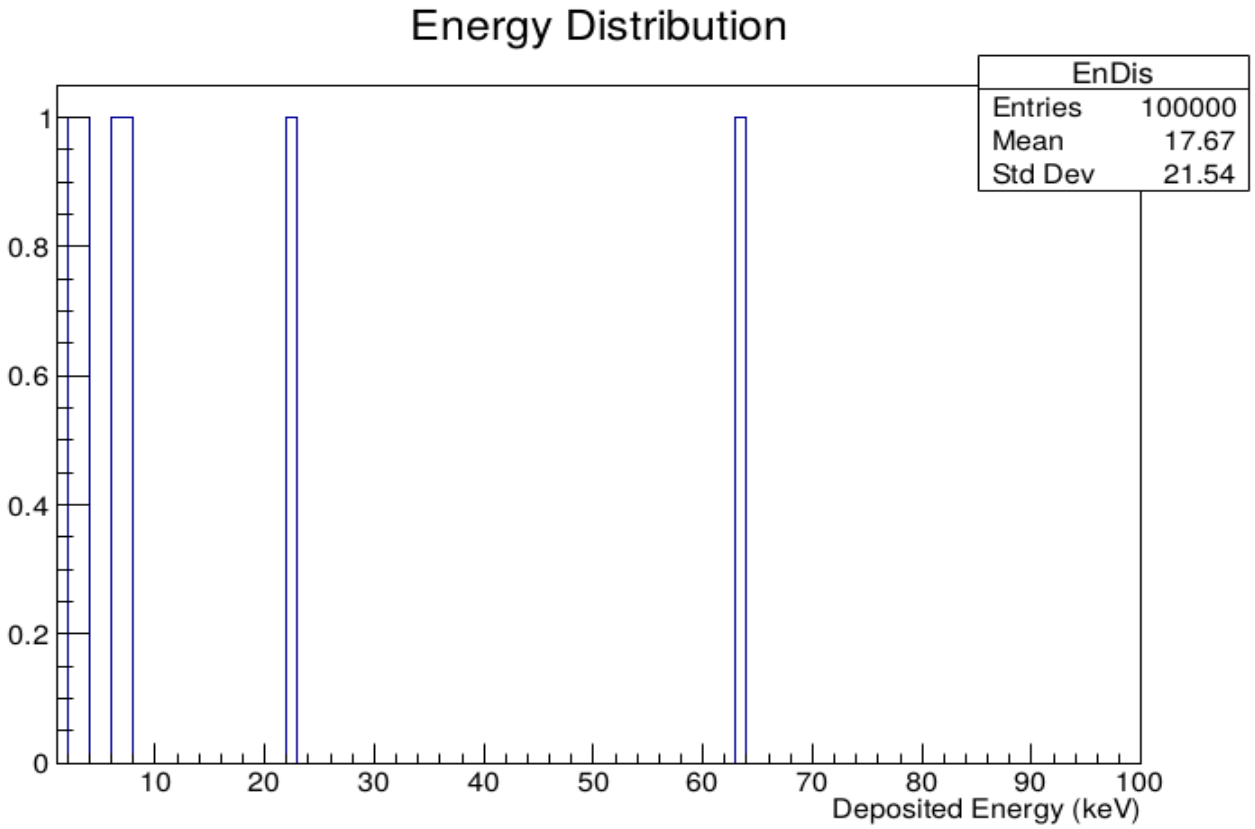
Si riportano, a scopo esemplificativo, due istogrammi della deposizione di energia nel detector, in due condizioni di energia della traccia primaria differenti.

- **10^5 neutroni da 2.5 MeV**



5.d

- **10^5 neutroni da 100 keV**

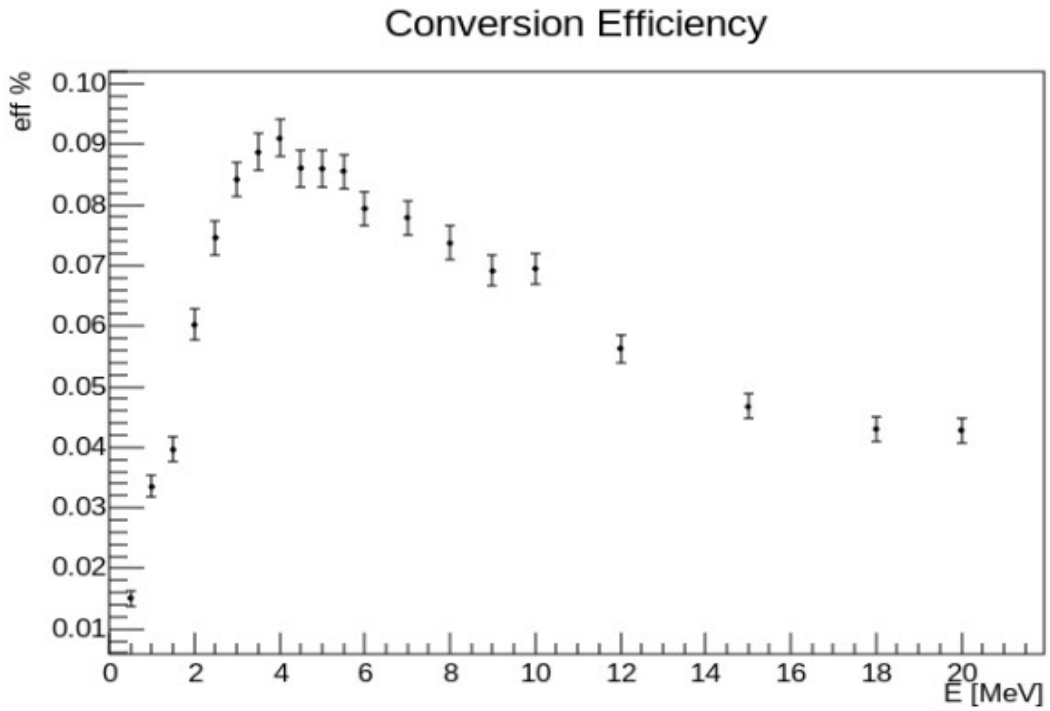


5.e

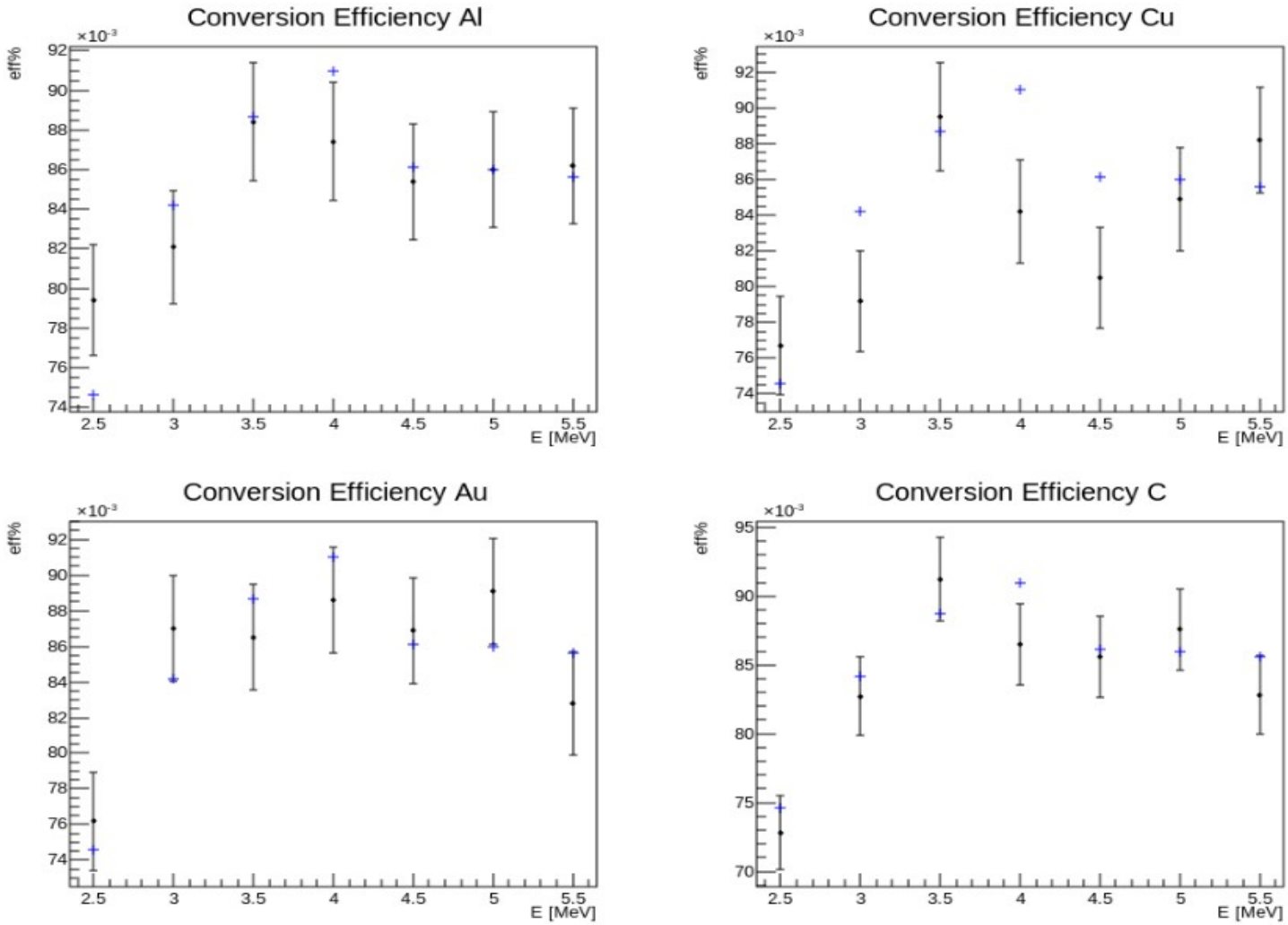
L’interazione del neutrone con il catodo alla produzione di protoni, neutroni deviati dallo scattering ed elettroni secondari. I protoni e gli elettroni secondari quindi interagiscono e ionizzano gli atomi del detector.

Già dai primi due istogrammi di distribuzione dell’energia depositata, è chiaro che il numero di particelle secondarie prodotte dipende dall’energia del neutrone primario.

Si procede quindi a fare uno studio sistematico dell’efficienza di conversione del catodo in polietilene, oro, alluminio, rame e carbonio. Si sono effettuati diversi run da 10^5 neutroni di energia compresa tra i 500 KeV e i 20 MeV.



5.f *Efficienza di conversione di neutroni del polietilene*



5.g *Confronto efficienza di conversione con polietilene (punti in blu)*